Data Handling and Descriptive Statistics

Il-Chul Moon
Dept. of Industrial and Systems Engineering
KAIST

icmoon@kaist.ac.kr

Value Chain of Data Analysis

- Ultimate goal of data science
  - Creating a value added to the real world via understanding data
  - Then, how to gain value from data understanding?
    - i.e. New knowledge and discovered patterns → new service, new operation
    - i.e. Real-time analysis and optimization → less cost and time

- How to understand data?

Simple Visualization
Simple Metric

Complex AI Model
Latent factors

Data Handling

Detour) Dataset Introduction: Credit Score

- This research aimed at the case of customers' default payments in Taiwan
- Dependent variable :
  - a binary variable, default payment (Yes = 1, No = 0), as the response variable.
- Independent variable:
  - X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
  - X2: Gender (1 = male; 2 = female).
  - X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
  - X4: Marital status (1 = married; 2 = single; 3 = others).
  - X5: Age (year).
  - X6 - X11: History of past payment.
    - We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005.
    - The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
  - X12-X17: Amount of bill statement (NT dollar).
    - X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
  - X18-X23: Amount of previous payment (NT dollar)
    - X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.
- Yeh, I. C., & Lien, C. H. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Systems with Applications, 36(2), 2473-2480.
- Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Dataset

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15
2	ID	LIMIT_BA	SEX	EDUCATI	MARRIAG	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AM	BILL_AM	BILL_AM	BILL_AM
3	1	20000	2	2	1	24	2	2	-1	-1	-2	-2	3913	3102	689	0
4	2	120000	2	2	2	26	-1	2	0	0	0	2	2682	1725	2682	3272
5	3	90000	2	2	2	34	0	0	0	0	0	0	29239	14027	13559	14331
6	4	50000	2	2	1	37	0	0	0	0	0	0	46990	48233	49291	28314
7	5	50000	1	2	1	57	-1	0	-1	0	0	0	8617	5670	35835	20940
8	6	50000	1	1	2	37	0	0	0	0	0	0	64400	57069	57608	19394
9	7	500000	1	1	2	29	0	0	0	0	0	0	367965	412023	445007	542653
10	8	100000	2	2	2	23	0	-1	-1	0	0	-1	11876	380	601	221
11	9	140000	2	3	1	28	0	0	2	0	0	0	11285	14096	12108	12211
12	10	20000	1	3	2	35	-2	-2	-2	-2	-1	-1	0	0	0	0
13	11	200000	2	3	2	34	0	0	2	0	0	-1	11073	9787	5535	2513
14	12	260000	2	1	2	51	-1	-1	-1	-1	-1	2	12261	21670	9966	8517
15	13	630000	2	2	2	41	-1	0	-1	-1	-1	-1	12137	6500	6500	6500

- All in numbers
  - Some are categorical variables in coding: SEX, EDU, MARRIAGE, PAY_X…
  - Some are numeric variables
    - AGE
    - BILL_AMOUNT
  - ID is a unique identifier, which does not convey the semantic meaning in its number
- How to load the dataset and understand the numbers?

Detour) *pandas* library

- pandas
  - is an open-source library for data analysis
  - aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.
- Easy to handle a tabular dataset with pandas
  - Dataset read/write/indexing….
    - pandas provides Input/Output interfaces for
      - JSON, CSV, XML, MS Excel, HDF5, SAS, SPSS, Python Pickle, SQL…..
  - Dataset manipulation
    - join, merge, concatenate, compare, reshape, pivot….
  - Dataset conversion
    - Can interoperate with NumPy, Python List and Dictionary…
- Working environment
  - Using the library of
    - matplotlib 3.6.1, numpy 1.23.4, pandas 1.5.0, xlrd 2.0.1

Dataset Read with *pandas*

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15
2	ID	LIMIT_BA	SEX	EDUCATI	MARRIAG	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AM	BILL_AM	BILL_AM	BILL_AM
3	1	20000	2	2	1	24	2	2	-1	-1	-2	-2	3913	3102	689	0
4	2	120000	2	2	2	26	-1	2	0	0	0	2	2682	1725	2682	3272
5	3	90000	2	2	2	34	0	0	0	0	0	0	29239	14027	13559	14331

Need to adjust the file path by yourself

```python
import pandas as pd

objData = pd.read_excel('../data/ucidataset-credit score/default of credit card clients.xls', skiprows=[
 1,2])
print(objData)
```

- Sample code
- API document
  - https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html

pandas.read_excel

```python
pandas.read_excel(io, sheet_name=0, header=0, names=None,
index_col=None, usecols=None, squeeze=None, dtype=None, engine=None,
converters=None, true_values=None, false_values=None, skiprows=None,
nrows=None, na_values=None, keep_default_na=True, na_filter=True,
verbose=False, parse_dates=False, date_parser=None, thousands=None,
decimal='.', comment=None, skipfooter=0, convert_float=None,
mangle_dupe_cols=True, storage_options=None)
```

[source]

- Sample outputs

```
       Unnamed: 0      X1  X2  X3  X4  X5  ...    X19    X20   X21    X22   X23  Y
0               2  120000   2   2   2  26  ...   1000   1000  1000      0  2000  1
1               3   90000   2   2   2  34  ...   1500   1000  1000   1000  5000  0
2               4   50000   2   2   1  37  ...   2019   1200  1100   1069  1000  0
3               5   50000   1   2   1  57  ...  36681  10000  9000    689   679  0
4               6   50000   1   1   2  37  ...   1815    657  1000   1000   800  0
...           ...     ...  ..  ..  ..  ..  ...    ...    ...   ...    ...   ... ..
29994       29996  220000   1   3   1  39  ...  20000   5003  3047   5000  1000  0
29995       29997  150000   1   3   2  43  ...   3526   8998   129      0     0  0
29996       29998   30000   1   2   2  37  ...      0  22000  4200   2000  3100  1
29997       29999   80000   1   3   1  41  ...   3409   1178  1926  52964  1804  1
29998       30000   50000   1   2   1  46  ...   1800   1430  1000   1000  1000  1
```

DataFrame

Concept of DataFrame in *pandas*

	Unnamed: 0	X1	X2	X3	X4	X5	...	X19	X20	X21	X22	X23	Y
0	2	120000	2	2	2	26	...	1000	1000	1000	0	2000	1
1	3	90000	2	2	2	34	...	1500	1000	1000	1000	5000	0
2	4	50000	2	2	1	37	...	2019	1200	1100	1069	1000	0
3	5	50000	1	2	1	57	...	36681	10000	9000	689	679	0
4	6	50000	1	1	2	37	...	1815	657	1000	1000	800	0
...	...	...	..	..	..	..	...	...	...	...	...	...	..
29994	29996	220000	1	3	1	39	...	20000	5003	3047	5000	1000	0
29995	29997	150000	1	3	2	43	...	3526	8998	129	0	0	0
29996	29998	30000	1	2	2	37	...	0	22000	4200	2000	3100	1
29997	29999	80000	1	3	1	41	...	3409	1178	1926	52964	1804	1
29998	30000	50000	1	2	1	46	...	1800	1430	1000	1000	1000	1

- objData = pd.read_excel('../data/ucidataset-credit score/default of credit card clients.xls', skiprows=[1,2])
  - The type of "objData" is "DataFrame"
- DataFrame is the most commonly utilized structure
  - Can be seen as a dictionary of column data
  - Dictionary of columns
  - Structured to access the row by using "iloc"
- DataFrame is built upon "Series" data structure
  - Could be accessed like list
  - Could be accessed like dict

```
print(objData['X1'])
print(type(objData['X1']))
print(objData.iloc[0])
print(type(objData.iloc[0]))
print(objData['X1'][0])
print(type(objData['X1'][0]))
```

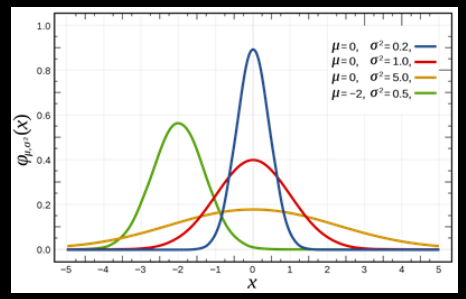
```
[29999 rows x 25 columns]
0        120000
1         90000
2         50000
3         50000
4         50000
          ...
29994    220000
29995    150000
29996     30000
29997     80000
29998     50000
Name: X1, Length: 29999, dtype: int64
<class 'pandas.core.series.Series'>
```

```
Unnamed: 0         2
X1            120000
X2                 2
X3                 2
X4                 2
X5                26
X6                -1
X7                 2
X8                 0
X9                 0
X10                0
X11                2
X12             2682
X13             1725
X14             2682
X15             3272
X16             3455
X17             3261
X18                0
X19             1000
X20             1000
X21             1000
X22                0
X23             2000
Y                  1
Name: 0, dtype: int64
<class 'pandas.core.series.Series'>
```

```
120000
<class 'numpy.int64'>
```

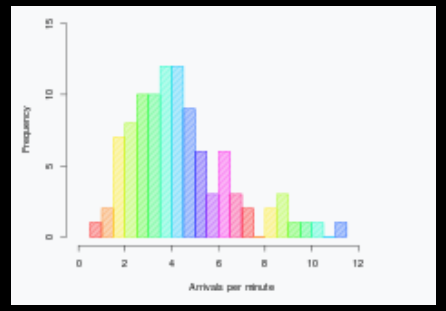
```
dic = {'a':1,'b':2,'c':3}
objSeries = pd.Series(dic)
print(objSeries)
print(objSeries['a'])
print(objSeries[1:3])
```

```
<class 'numpy.int64'>
a    1
b    2
c    3
dtype: int64
1
b    2
c    3
dtype: int64
```



© wikipedia

Historgam

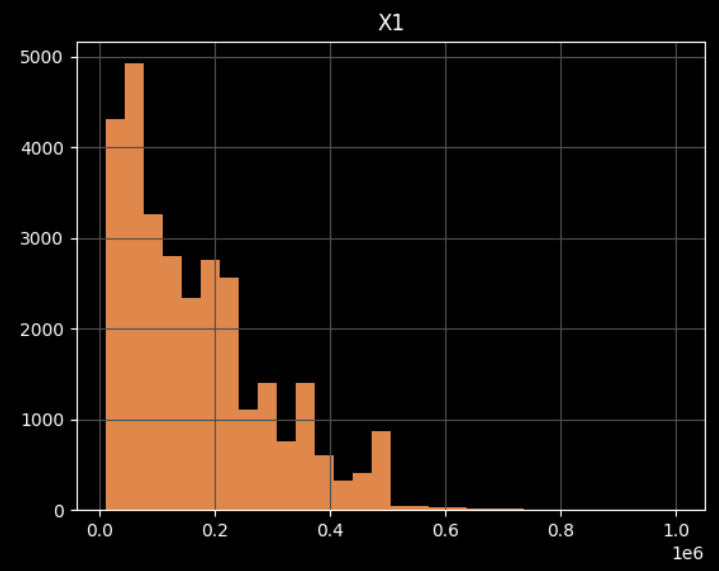


- One of the most fundamental method to understand a data distribution
  - Provides an empirical density
  - An approximate representation of the distribution of numerical data
- Drawing a histogram
  - Setup the "bin"s of dataset
    - The bins are often equal size in terms of the range
  - Count the occurrence of data instances
    - The bins have corresponding counts from the data occurrence of the bin
  - Draw a barchart of the count by the bins
- Count of data instances ≈ Density of data distribution
- Example)
  - Draw a histogram of the height of children
    - 125, 135, 110, 100, 117, 130, 145, 90, 150, 133

Histogram from DataFrame

```
import pandas as pd
import matplotlib.pyplot as plt
objData = pd.read_excel('../data/ucidataset-credit score/default of credit card clients.xls', skiprows=[
 1,2])
objHist = objData.hist(column='X1',bins=30)
plt.show()
```

- *pandas* and *matplotlib* provides histogram
  - *matplotlib* is a charting library
- Histogram by pandas
  - You can setup the bin number
    - If you omit the bin number, pandas will set the number for you
  - "plt.show()" is needed
    - pandas relies upon matplotlib for drawing
- What can you tell from the histogram?
  - In terms of distribution and density

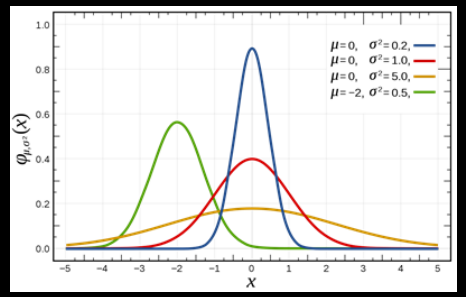


Bin Size Selection

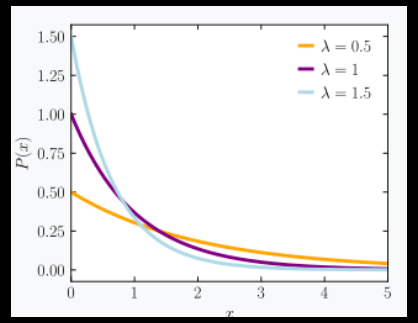
- Which bin size would be better?
  - How to setup the bin size?
  - Noise vs. Complexity
- Some systematic selection by formula
  - Square-root choice : $k = \lceil \sqrt{n} \rceil$
  - Sturge's formula : $k = \sqrt{\log_2 n} + 1$
  - There are many more formulas….
- What's the ultimate goal of drawing histogram?
  - Identifying the data distribution

© wikipedia

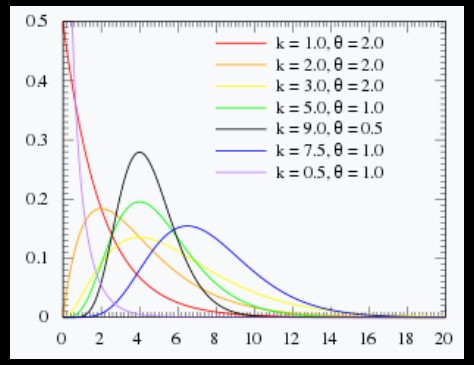
Normal Dist



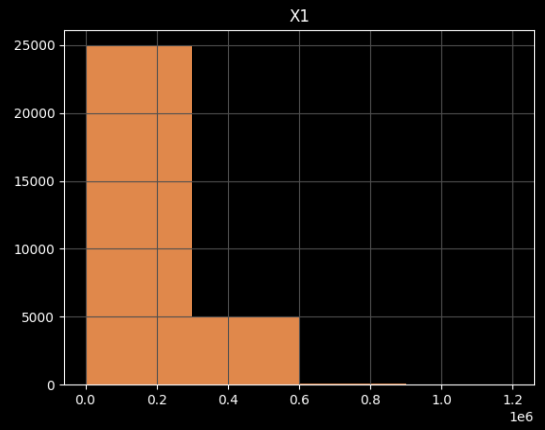
Exp. Dist



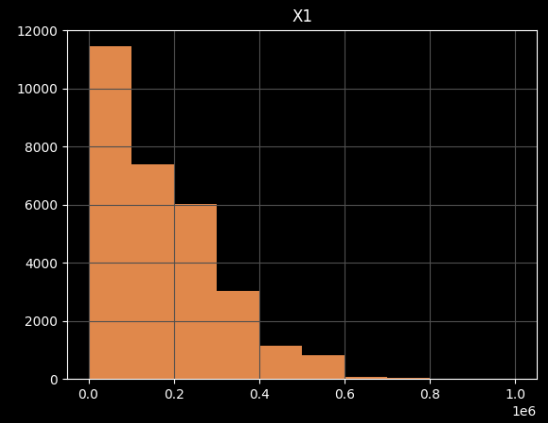
Gamma Dist



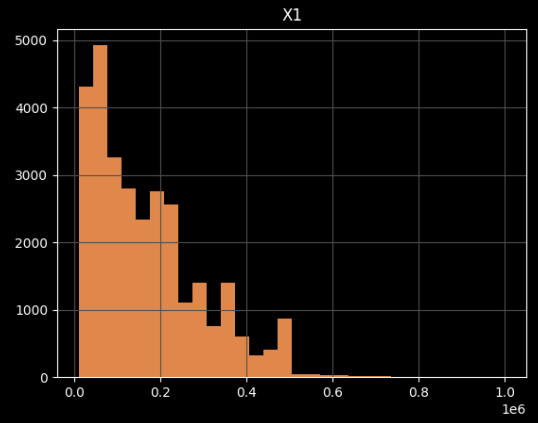
4 Bins



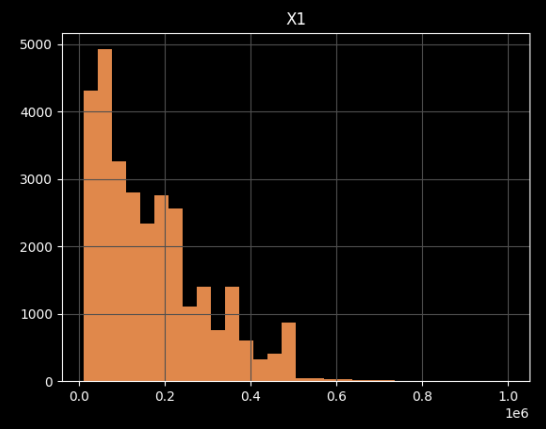
11 Bins



30 Bins

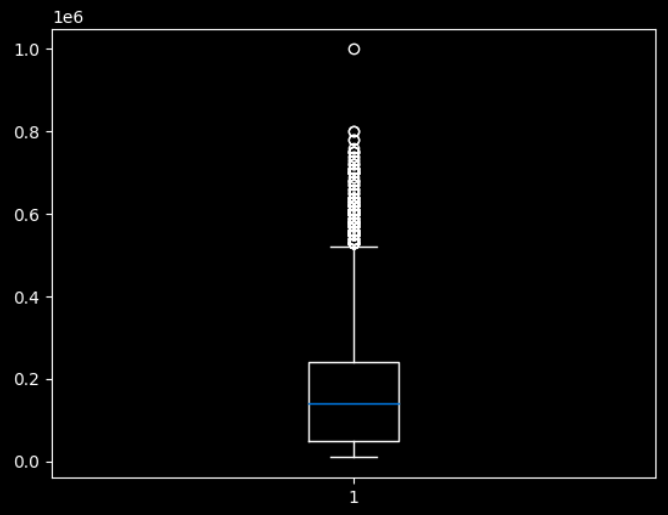


Descriptive Statistics



- How to represent the data in a few numbers?
  - Initial description of data
  - Summary statistics
  - Variables as well as visualization, i.e. histogram, box plot
- Some examples
  - Mean
  - Median
  - Variance, Standard deviation
  - Quantiles

```
plt.figure()
plt.boxplot(objData['X1'])
plt.show()
```

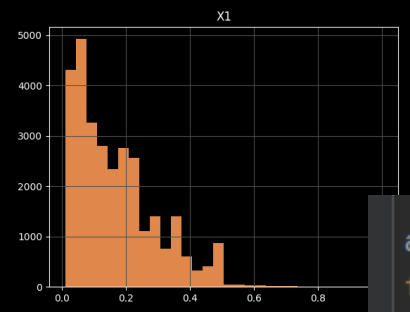


Q1 and Q3 are visualized

$IQR = Q_3 - Q_1$

Max Boundary = Q3+1.5IQR

Min Boundary = Q1-1.5IQR



Empirical Mean and Variance

```
avg = 0
for i in range(len(objData['X1'])):
    avg = avg + objData['X1'][i]
avg = avg / float(len(objData['X1']))
print(avg)

variance = 0
for i in range(len(objData['X1'])):
    variance = variance + math.pow(objData['X1'][i] - avg, 2)
variance = variance / float(len(objData['X1']))
print(variance)
std_deviation = math.sqrt(variance)
print(std_deviation)
```

Method 1

```
avg = 0
avg_squared = 0
for i in range(len(objData['X1'])):
    avg = avg + objData['X1'][i]
    avg_squared = avg_squared + math.pow(objData['X1'][i], 2)
avg = avg / float(len(objData['X1']))
avg_squared = avg_squared / float(len(objData['X1']))
variance = avg_squared - math.pow(avg,2)
std_deviation = math.sqrt(variance)
print(avg)
print(variance)
print(std_deviation)
```

Method 2

- Empirical Mean: $E_n(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$
- Mean Squared Variance (or, Mean Squared Deviation): $\mathrm{var}_n(X) = E_n\{[X - E_n(X)]^2\} = \frac{1}{n}\sum_{i=1}^{n}[x_i - E_n(X)]^2$
  - $= \frac{1}{n}\sum_{i=1}^{n}\left[x_i^2 - 2x_iE_n(X) + E_n(X)^2\right] = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - 2E_n(X)\frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n}\sum_{i=1}^{n} E_n(X)^2$
    $= \frac{1}{n}\sum_{i=1}^{n} x_i^2 - 2E_n(X)\frac{1}{n}nE_n(X) + E_n(X)^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - E_n(X)^2$ $\qquad \mathrm{var}_n(X) = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - E_n(X)^2$
- Sample Variance: $S^2(X) = \frac{1}{n-1}\sum_{i=1}^{n}[x_i - E_n(X)]^2$
  - So, the sample variance can be unbiased estimator.

$$\mathrm{var}_n(X) = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - E_n(X)^2$$

$$\mathrm{var}_n(X) + E_n(X)^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2$$

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n} x_i^2 - \frac{1}{n(n-1)}\left(\sum_{i=1}^{n} x_i\right)^2$$

Unbiased Estimator

- Our dataset is a sample instance from large population.
  - The descriptive statistics from samples may be biased by the sampling effect
  - To nullify the bias, we need a guarantee that
    - $N \to \infty$, $s_N \to s_\infty$
    - i.e. As $N \to \infty$, $S^2(X) = \frac{1}{n-1}\sum_{i=1}^{n}[x_i - E_n(X)]^2 \to \sigma^2$
      - σ: the population variance (true variance), μ: the population mean (true mean)
- Let's calculate the ***expected*** sample variance
  - $E(S^2) = E\left(\frac{1}{n-1}\sum_{i=1}^{n} X_i^2 - \frac{1}{n}\frac{1}{(n-1)}(\sum_{i=1}^{n} X_i)^2\right)$

$$= \frac{1}{n-1}\sum_{i=1}^{n} E(X_i^2) - \frac{1}{n}\frac{1}{(n-1)}E\left(\left(\sum_{i=1}^{n} X_i\right)^2\right)$$

$$= \frac{1}{n-1}\sum_{i=1}^{n}[\mathrm{var}_n(X) + E_n(X)^2] - \frac{1}{n}\frac{1}{(n-1)}\left[\mathrm{var}_n\left(\sum_{i=1}^{n} X_i\right) + E_n\left(\sum_{i=1}^{n} X_i\right)^2\right]$$

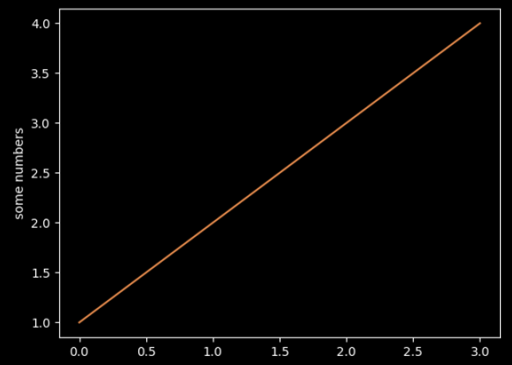
$$= \frac{1}{n-1}\sum_{i=1}^{n}[\sigma^2 + \mu^2] - \frac{1}{n}\frac{1}{(n-1)}[n\sigma^2 + (n\mu)^2] = \frac{n}{n-1}(\sigma^2 + \mu^2) - \frac{1}{(n-1)}[\sigma^2 + n\mu^2] = \sigma^2$$

- Hence, as $N \to \infty$, $S^2(X) = \frac{1}{n-1}\sum_{i=1}^{n}[x_i - E_n(X)]^2 \to \sigma^2$
- **<u>Use unbiased estimator as descriptive statistics!</u>**

Detour) matplotlib

- *matplotlib* is a library for data visualizations in Python
  - Many papers depend upon matplotlib for their result visualizations
  - Interactions are possible
  - Many customization options
- pyplot
  - An interface to matplotlib library
  - Figure GUI manager
  - https://matplotlib.org/stable/api/pyplot_summary.html#module-matplotlib.pyplot

```python
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



matplotlib.pyplot.plot

matplotlib.pyplot.plot(args, scalex=True, scaley=True, data=None, **kwargs*)** [source]

Plot y versus x as lines and/or markers.

Call signatures:

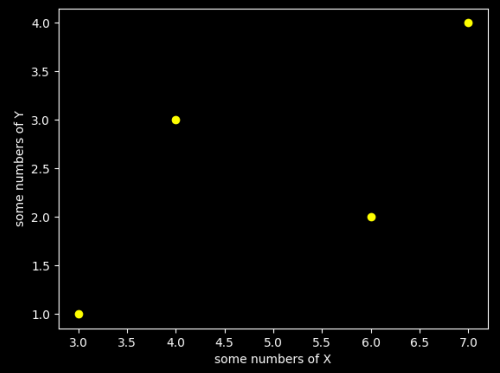
```
plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

The coordinates of the points or line nodes are given by *x*, *y*.

The optional parameter *fmt* is a convenient way for defining basic formatting like color, marker and linestyle. It's a shortcut string notation described in the *Notes* section below.

```
>>> plot(x, y)        # plot x and y using default line style and color
>>> plot(x, y, 'bo')  # plot x and y using blue circle markers
>>> plot(y)           # plot y using x as index array 0..N-1
>>> plot(y, 'r+')     # ditto, but with red plusses
```

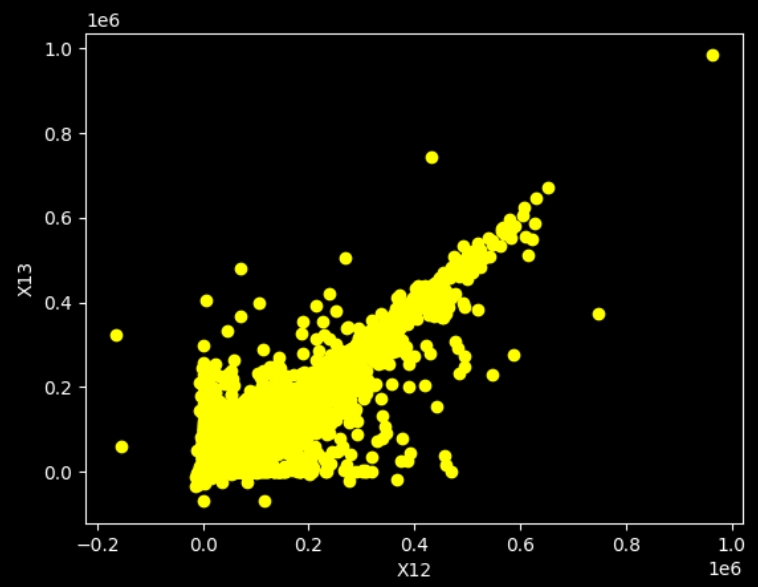
```python
import matplotlib.pyplot as plt
plt.plot([3, 6, 4, 7], [1, 2, 3, 4], 'bo')
plt.xlabel('some numbers of X')
plt.ylabel('some numbers of Y')
plt.show()
```



Scatter Plot

- Scatter plot is used to show
  - Two continuous variable-pair instances
  - Useful for visualizing the dependency between the two variables
- Dependence ≠ Correlation
  - Dependence
    - X is depends on Y
      - in its value determination or sampling
    - Direction without correlation relation type
  - Correlation
    - X is (linearly) correlated with Y
    - X is (nonlinearly) correlated with Y
    - Correlation relation type without direction

```
plt.plot(objData['X12'],objData['X13'],'bo')
plt.xlabel('X12')
plt.ylabel('X13')
plt.show()
```



X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005

X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
X2: Gender (1 = male; 2 = female).
X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
X4: Marital status (1 = married; 2 = single; 3 = others).

Pivot Table

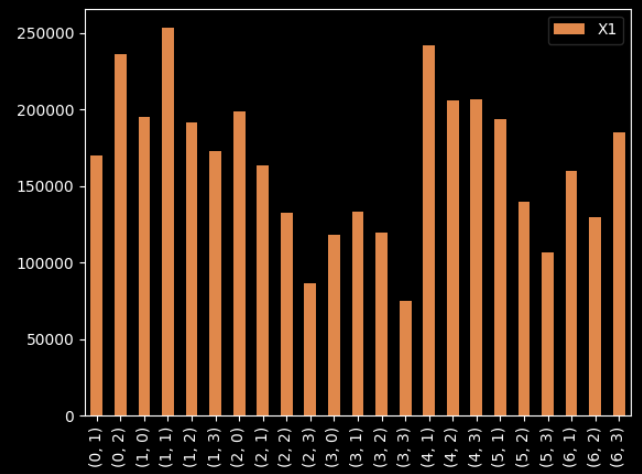
```
objData = pd.read_excel('../data/ucidataset-credit score/default of credit card clients.xls', skiprows=[
 1,2])
print(objData)

objPivotSum = pd.pivot_table(objData, values='X1', index=['X3','X4'], aggfunc=np.sum)
print(objPivotSum)
objPivotAvg = pd.pivot_table(objData, values='X1', index=['X3','X4'], aggfunc=np.mean)
print(objPivotAvg)

objPivotAvg.plot(kind='bar')
plt.show()
```

```
                X1
X3 X4
0  1       680000
   2      2360000
1  0       780000
   1    942090000
   2   1302630000
   3      8640000
2  0      1190000
   1   1118376000
   2    929690000
   3     14010000
3  0      5210000
   1    381450000
   2    227867680
   3      7720000
4  1     12570000
   2     13980000
   3       620000
5  1     29026000
   2     17740000
   3       320000
```

- *pivot_table* will create another dataframe of pandas
  - *values* is the value to be aggregated by pivoting
  - *index* is the column set to identify a group
  - *aggfunc* is the function to be used for the aggregation
  - DataFrame from pivot_table supports "plot" in conjunction with matplotlib
- Pivoting assumes the column index to be categorical variables, so the group can be identified discretely



X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
X2: Gender (1 = male; 2 = female).
X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
X4: Marital status (1 = married; 2 = single; 3 = others).

One-hot Encoding

- Categorical variable requires a special treatment for data processing
  - Code ≠ Value
- X2 : Gender (1=male; 2=female)
  - There should be no ordering or value in being either male or female
  - If we just accept the code as the value, we are creating an ordinal information
    - Order comes from 1 and 2 in Z
- To avoid this confusion and to support the learning in the downstream, we convert the categorical variable to be "one-hot" encoded
  - i.e. X2
    - Male : (0, 1)
    - Female : (1, 0)

```
objDataOneHot = pd.get_dummies(objData,columns=['X2','X3','X4'])
print(objDataOneHot)
```

	Unnamed: 0	X1	X5	X6	X7	X8	...	X3_5	X3_6	X4_0	X4_1	X4_2	X4_3
0	2	120000	26	-1	2	0	...	0	0	0	0	1	0
1	3	90000	34	0	0	0	...	0	0	0	0	1	0
2	4	50000	37	0	0	0	...	0	0	0	1	0	0
3	5	50000	57	-1	0	-1	...	0	0	0	1	0	0
4	6	50000	37	0	0	0	...	0	0	0	0	1	0
...	...	...	..	..	..	..	...	...	...	...	...	...	...
29994	29996	220000	39	0	0	0	...	0	0	0	1	0	0
29995	29997	150000	43	-1	-1	-1	...	0	0	0	0	1	0
29996	29998	30000	37	4	3	2	...	0	0	0	0	1	0
29997	29999	80000	41	1	-1	0	...	0	0	0	1	0	0
29998	30000	50000	46	0	0	0	...	0	0	0	1	0	0