Preprocessing

Eunseo, Lee
Chaeyun, Jang

Index

- Basic skills, Missing data, preprocessing (statsmodel package/scikit-learn package)

Basic skills

- Pandas
- skimming:df.Info(),df.head(),df.describe(),df.columns(),df.value_counts()
- Converting name of row/column: df_c=df.rename(columns={'국어':kor',' 영어',:eng…})
- Finding certain row : df['국어']
- sorting: df. Sort_values('국어',inplace=True,(ascending=True))
- Find certain row with special condition: temp=df.query('국어>70')
- delete: df.drop(df.index[0:5])/df.drop(df.columns[4],axis=1)
- addition: df.loc['짱구']=[90,80,70,30]
- Converting : df.colums=df.reindex(columns=['A','B','C','D']

- https://youngq.tistory.com/39?category=764297

Missing data

- 실수형(real number)->NaN(not a number)
- 정수형(integer)->real number
- 날짜/시간(datetime) ->NaT (parse date)
- 문자열(string) ->빈 문자열(null string)/데이터 없음(NA: not avaiable)
- Df=pd.read_csv(csv_data,dtype={x1:pd.Int64Dtype(),parse_dates=[3])
- Parse: To split a file or other input into pieces of data that can be easily stored or manipulated

Processing missing data

- Find-df.isnull()/df.isna(): find the location of missing value
- Delete- df.dropna(thres=N,axis=N):thres: threshold of #Na
- Axis:1- delete column/0: delete row
- Replace: scikit-learn:simpleImputer package

Find null data

	x1	x2	x3	x4	x5
0	1	0.1	1.0	2019-01-01	A
1	2	NaN	NaN	2019-01-02	B
2	3	NaN	3.0	2019-01-03	C
3	NaN	0.4	4.0	2019-01-04	A
4	5	0.5	5.0	2019-01-05	B
5	NaN	NaN	NaN	2019-01-06	C
6	7	0.7	7.0	NaT	A
7	8	0.8	8.0	2019-01-08	B
8	9	0.9	NaN	2019-01-09	C

```
df.isnull()
```

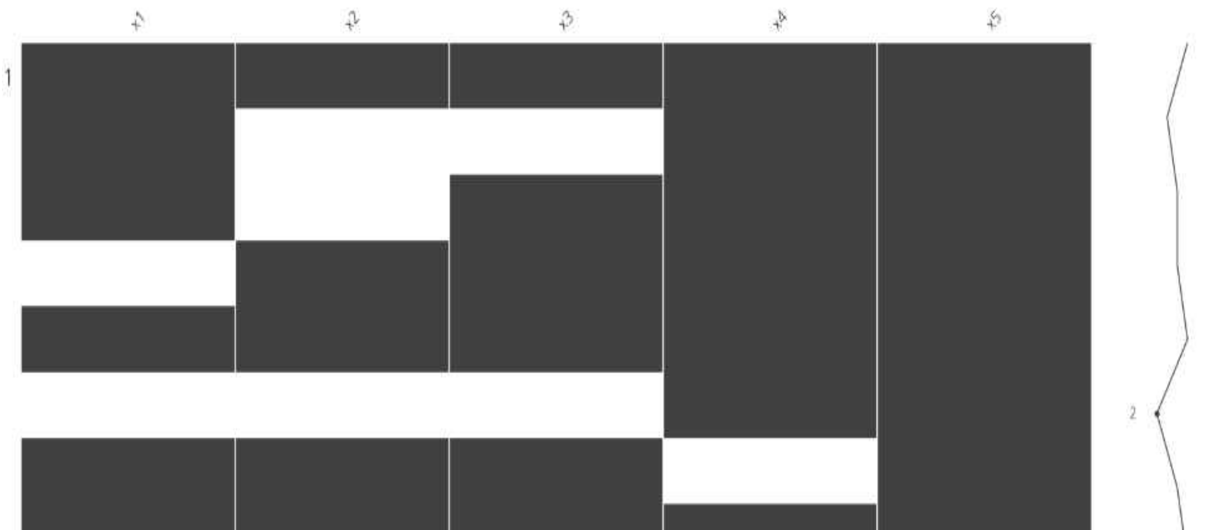
	x1	x2	x3	x4	x5
0	False	False	False	False	False
1	False	True	True	False	False
2	False	True	False	False	False
3	True	False	False	False	False
4	False	False	False	False	False
5	True	True	True	False	False
6	False	False	False	True	False
7	False	False	False	False	False
8	False	False	True	False	False

```
df.isnull().sum()
```

```
x1    2
x2    3
x3    3
x4    1
x5    0
dtype: int64
```

```
import missingno as msno

msno.matrix(df)
plt.show()
```



Delete null data

```
df = sns.load_dataset("titanic")
df.tail()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town
886	0	2	male	27.0	0	0	13.00	S	Second	man	True	NaN	Southampton
887	1	1	female	19.0	0	0	30.00	S	First	woman	False	B	Southampton
888	0	3	female	NaN	1	2	23.45	S	Third	woman	False	NaN	Southampton
889	1	1	male	26.0	0	0	30.00	C	First	man	True	C	Cherbourg
890	0	3	male	32.0	0	0	7.75	Q	Third	man	True	NaN	Queenstown

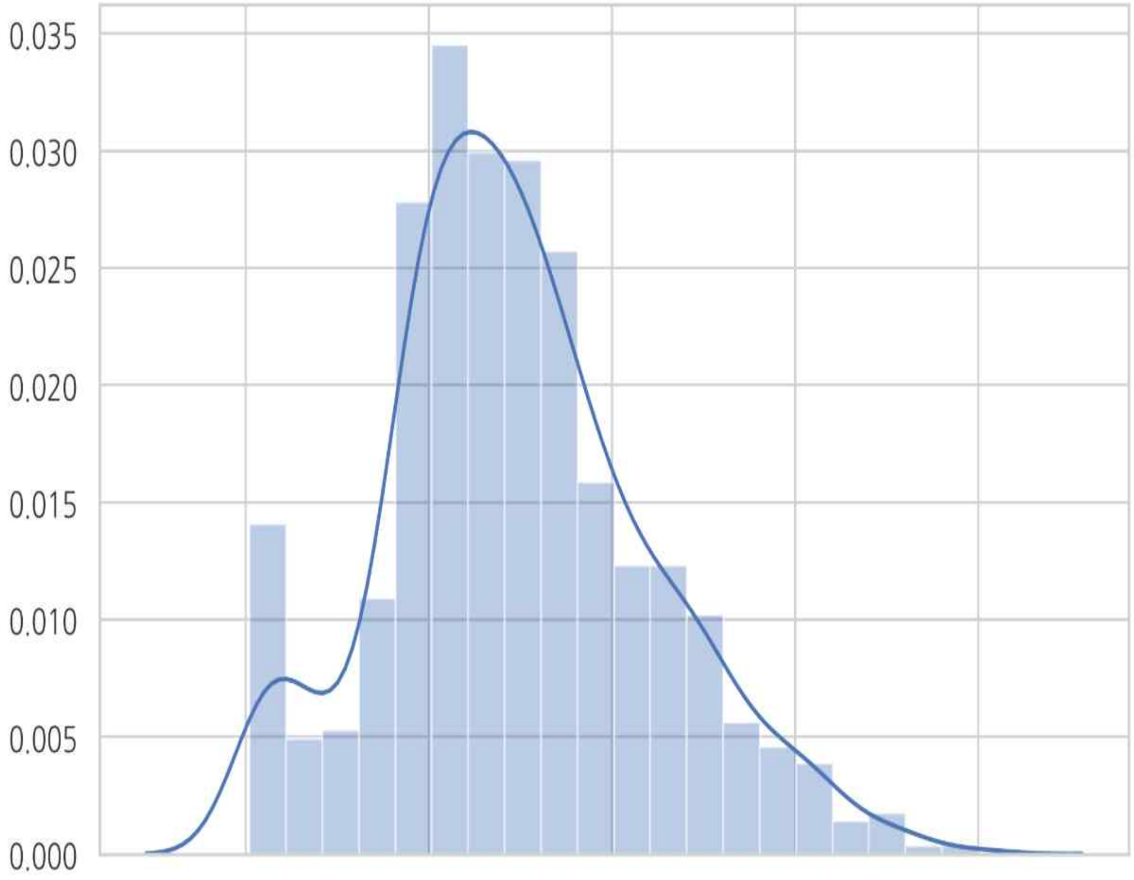
```
df.dropna(thresh=5,axis=0)
```

	x1	x2	x3	x4	x5
0	1	0.1	1.0	2019-01-01	A
4	5	0.5	5.0	2019-01-05	B
7	8	0.8	8.0	2019-01-08	B

```
df.dropna(thresh=7, axis=1)
```

	x1	x4	x5
0	1	2019-01-01	A
1	2	2019-01-02	B
2	3	2019-01-03	C
3	NaN	2019-01-04	A
4	5	2019-01-05	B
5	NaN	2019-01-06	C
6	7	NaT	A
7	8	2019-01-08	B
8	9	2019-01-09	C

```
sns.distplot(df.age.dropna())
plt.show()
```



Replace null data

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
df_copy1 = df.copy()
df_copy1["age"] = imputer.fit_transform(df.age.values.reshape(-1,1))

msno.bar(df_copy1)
plt.show()
```

Statsmodel package

- Function in R->perform with Python

- Patsy package(help data encoding/transformation)
- What is encoding?:

Encoding is the process of converting data into a format required for a number of information processing needs, including:

Program compiling and execution

Data transmission, storage and compression/decompression

Application data processing, such as file conversion

- R style formula operator
- Stateful formation

Patsy package:for linear regression preprocessing

```
df = pd.DataFrame(demo_data("x1", "x2", "y"))
df
```

	x1	x2	y
0	1.764052	-0.977278	0.144044
1	0.400157	0.950088	1.454274
2	0.978738	-0.151357	0.761038
3	2.240893	-0.103219	0.121675
4	1.867558	0.410599	0.443863

```
dmatrix("x1", df)
```

```
DesignMatrix with shape (5, 2)
  Intercept       x1
          1  1.76405
          1  0.40016
          1  0.97874
          1  2.24089
          1  1.86756
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
```

Bias augmentation

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \rightarrow X_a = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1D} \\ 1 & x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}$$

$$f(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D = \begin{bmatrix} 1 & x_1 & x_2 & \cdots & x_D \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{bmatrix}$$

$$= x_a^T w_a = w_a^T x_a$$

R style formula operator

기호	설명
1, 0	바이어스(bias, intercept) 추가 및 제거
+	설명 변수 추가
-	설명 변수 제거
:	상호작용(interaction)
*	a*b = a + b + a:b
/	a/b = a + a:b

```
dmatrix("x1 - 1", df)
```

```
DesignMatrix with shape (5, 1)
      x1
  1.76405
  0.40016
  0.97874
  2.24089
  1.86756
  Terms:
    'x1' (column 0)
```

```
dmatrix("x1 + x2", df)
```

```
DesignMatrix with shape (5, 3)
  Intercept       x1        x2
          1  1.76405  -0.97728
          1  0.40016   0.95009
          1  0.97874  -0.15136
          1  2.24089  -0.10322
          1  1.86756   0.41060
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'x2' (column 2)
```

```
dmatrix("x1 + x2 + x1:x2", df)
```

```
DesignMatrix with shape (5, 4)
  Intercept       x1        x2     x1:x2
          1  1.76405  -0.97728  -1.72397
          1  0.40016   0.95009   0.38018
          1  0.97874  -0.15136  -0.14814
          1  2.24089  -0.10322  -0.23130
          1  1.86756   0.41060   0.76682
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'x2' (column 2)
    'x1:x2' (column 3)
```

```
dmatrix("x1 / x2", df)
```

```
DesignMatrix with shape (5, 3)
  Intercept       x1     x1:x2
          1  1.76405  -1.72397
          1  0.40016   0.38018
          1  0.97874  -0.14814
          1  2.24089  -0.23130
          1  1.86756   0.76682
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'x1:x2' (column 2)
```

Transform/stateful transform

```
dmatrix("x1 + np.log(np.abs(x2))", df)
```

```
DesignMatrix with shape (5, 3)
  Intercept       x1  np.log(np.abs(x2))
          1  1.76405            -0.02298
          1  0.40016            -0.05120
          1  0.97874            -1.88811
          1  2.24089            -2.27090
          1  1.86756            -0.89014
  Terms:
    'Intercept' (column 0)
    'x1' (column 1)
    'np.log(np.abs(x2))' (column 2)
```

```
def doubleit(x):
    return 2 * x

dmatrix("doubleit(x1)", df)
```

```
DesignMatrix with shape (5, 2)
  Intercept  doubleit(x1)
          1       3.52810
          1       0.80031
          1       1.95748
          1       4.48179
          1       3.73512
  Terms:
    'Intercept' (column 0)
    'doubleit(x1)' (column 1)
```

Scikit-learn package

- Scaling
- Pipeline
- Transform

Scaling

- a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization

StandardScaler(X) : 평균이 0과 표준편차가 1이 되도록 변환.
RobustScaler(X) : 중앙값(median)이 0, IQR(interquartile range)이 1이 되도록 변환.
MinMaxScaler(X) : 최대값이 각각 1, 최소값이 0이 되도록 변환
MaxAbsScaler(X) : 0을 기준으로 절대값이 가장 큰 수가 1또는 -1이 되도록 변환

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
np.mean(X_scaled), np.std(X_scaled)
```

```
(0.0, 1.0)
```

```
from sklearn.preprocessing import RobustScaler

robust_scaler = RobustScaler()
robust_scaler.fit(X)
X_robust_scaled = robust_scaler.transform(X)
np.mean(X_robust_scaled), np.std(X_robust_scaled)
```

```
(2.0888888888888889, 6.622408647636923)
```

Pipeline

- *Sequentially apply a list of transforms and a final estimator. Intermediate steps of pipeline must implement fit and transform methods and the final estimator only needs to implement fit.*
- *1.* train and test data loss can be avoided.
- *2.* Easily create cross-validation and other model selection types
- *3.* increased reproducibility

```python
from sklearn.datasets import make_regression,make_classification

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

X, y = make_classification(n_samples=100,n_features=10,n_informative=2)

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.33, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape

# it takes a list of tuples as parameter
pipeline = Pipeline([
    ('scaler',StandardScaler()),
    ('clf', LogisticRegression())
        ])

# use the pipeline object as you would
# a regular classifier
pipeline.fit(X_train,y_train)


y_preds = pipeline.predict(X_test)

accuracy_score(y_test,y_preds)
```

Transform

$$x \rightarrow [f_1(x), f_2(x), f_3(x), \cdots]$$

```
from sklearn.preprocessing import FunctionTransformer


def kernel(X):
    x0 = X[:, :1]
    x1 = X[:, 1:2]
    x2 = X[:, 2:3]
    X_new = np.hstack([x0, 2 * x1, x2 ** 2, np.log(x1)])
    return X_new
```

```
X = np.arange(12).reshape(4, 3)
X
```

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

```
kernel(X)
```

```
array([[  0.        ,   2.        ,   4.        ,   0.        ],
       [  3.        ,   8.        ,  25.        ,   1.38629436],
       [  6.        ,  14.        ,  64.        ,   1.94591015],
       [  9.        ,  20.        , 121.        ,   2.30258509]])
```

```
FunctionTransformer(kernel).fit_transform(X)
```

```
array([[  0.        ,   2.        ,   4.        ,   0.        ],
       [  3.        ,   8.        ,  25.        ,   1.38629436],
       [  6.        ,  14.        ,  64.        ,   1.94591015],
       [  9.        ,  20.        , 121.        ,   2.30258509]])
```

Scaling - conditional number

- **conditional number**

df) conditional number = max eigenvalue / min eigenvalue

If conditional number 大 , error 大

In [14]:
```python
import numpy as np
import scipy.linalg as linalg
import scipy as sp
```

In [15]:
```python
A = np.eye(4)
```

In [16]:
```python
b = np.ones(4)
sp.linalg.solve(A, b)
```

Out[16]:
```
array([1., 1., 1., 1.])
```

In [17]:
```python
sp.linalg.solve(A + 0.0001 * np.eye(4), b)
```

Out[17]:
```
array([0.99990001, 0.99990001, 0.99990001, 0.99990001])
```

In [18]:
```python
A = sp.linalg.hilbert(4)
A
```

Out[18]:
```
array([[1.        , 0.5       , 0.33333333, 0.25      ],
       [0.5       , 0.33333333, 0.25      , 0.2       ],
       [0.33333333, 0.25      , 0.2       , 0.16666667],
       [0.25      , 0.2       , 0.16666667, 0.14285714]])
```

In [19]:
```python
np.linalg.cond(A)
```

Out[19]:
```
15513.738738929038
```

In [20]:
```python
sp.linalg.solve(A, b)
```

Out[20]:
```
array([  -4.,   60., -180.,  140.])
```

In [21]:
```python
sp.linalg.solve(A + 0.0001 * np.eye(4), b)
```

Out[21]:
```
array([ -0.58897672,  21.1225671 , -85.75912499,  78.45650825])
```

Linear regression & Scaling

1. When the scale of the number varies greatly due to the **unit differences of the variables.** In this case, it is solved by scaling.
2. If there are independent **variables that are highly correlated**, that is, select variables or reduce dimensions using PCA.

* In StatsModels, we can scale using the scale command in the model designation string. Scaling in this way is convenient because it stores the mean and standard deviation used for scaling and then uses the same scale when using the predict command later. Note that the dummy variable CHAS does not scale.

In [8]:

```
from sklearn.datasets import load_boston

boston = load_boston()

dfX = pd.DataFrame(boston.data, columns=boston.feature_names)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])
df = pd.concat([dfX, dfy], axis=1)

model1 = sm.OLS.from_formula("MEDV ~ " + "+".join(boston.feature_names), data=df)
result1 = model1.fit()
print(result1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   MEDV   R-squared:                       0.741
Model:                            OLS   Adj. R-squared:                  0.734
Method:                 Least Squares   F-statistic:                     108.1
Date:                Mon, 17 Jun 2019   Prob (F-statistic):          6.72e-135
Time:                        19:17:18   Log-Likelihood:                -1498.8
No. Observations:                 506   AIC:                             3026.
Df Residuals:                     492   BIC:                             3085.
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     36.4595      5.103      7.144      0.000      26.432      46.487
CRIM          -0.1080      0.033     -3.287      0.001      -0.173      -0.043
ZN             0.0464      0.014      3.382      0.001       0.019       0.073
INDUS          0.0206      0.061      0.334      0.738      -0.100       0.141
CHAS           2.6867      0.862      3.118      0.002       0.994       4.380
NOX          -17.7666      3.820     -4.651      0.000     -25.272     -10.262
RM             3.8099      0.418      9.116      0.000       2.989       4.631
AGE            0.0007      0.013      0.052      0.958      -0.025       0.027
DIS           -1.4756      0.199     -7.398      0.000      -1.867      -1.084
RAD            0.3060      0.066      4.613      0.000       0.176       0.436
TAX           -0.0123      0.004     -3.280      0.001      -0.020      -0.005
PTRATIO       -0.9527      0.131     -7.283      0.000      -1.210      -0.696
B              0.0093      0.003      3.467      0.001       0.004       0.015
LSTAT         -0.5248      0.051    -10.347      0.000      -0.624      -0.425
==============================================================================
Omnibus:                      178.041   Durbin-Watson:                   1.078
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              783.126
Skew:                           1.521   Prob(JB):                    8.84e-171
Kurtosis:                       8.281   Cond. No.                     1.51e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.51e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [11]:

```
feature_names = list(boston.feature_names)
feature_names.remove("CHAS")
feature_names = ["scale({})".format(name) for name in feature_names] + ["CHAS"]
model3 = sm.OLS.from_formula("MEDV ~ " + "+".join(feature_names), data=df2)
result3 = model3.fit()
print(result3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   MEDV   R-squared:                       0.741
Model:                            OLS   Adj. R-squared:                  0.734
Method:                 Least Squares   F-statistic:                     108.1
Date:                Mon, 17 Jun 2019   Prob (F-statistic):          6.72e-135
Time:                        19:17:19   Log-Likelihood:                -1498.8
No. Observations:                 506   AIC:                             3026.
Df Residuals:                     492   BIC:                             3085.
Df Model:                          13
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept         22.3470      0.219    101.943      0.000      21.916      22.778
scale(CRIM)       -0.9281      0.282     -3.287      0.001      -1.483      -0.373
scale(ZN)          1.0816      0.320      3.382      0.001       0.453       1.710
scale(INDUS)       0.1409      0.421      0.334      0.738      -0.687       0.969
scale(NOX)        -2.0567      0.442     -4.651      0.000      -2.926      -1.188
scale(RM)          2.6742      0.293      9.116      0.000       2.098       3.251
scale(AGE)         0.0195      0.371      0.052      0.958      -0.710       0.749
scale(DIS)        -3.1040      0.420     -7.398      0.000      -3.928      -2.280
scale(RAD)         2.6622      0.577      4.613      0.000       1.528       3.796
scale(TAX)        -2.0768      0.633     -3.280      0.001      -3.321      -0.833
scale(PTRATIO)    -2.0606      0.283     -7.283      0.000      -2.617      -1.505
scale(B)           0.8493      0.245      3.467      0.001       0.368       1.331
scale(LSTAT)      -3.7436      0.362    -10.347      0.000      -4.454      -3.033
CHAS               2.6867      0.862      3.118      0.002       0.994       4.380
==============================================================================
Omnibus:                      178.041   Durbin-Watson:                   1.078
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              783.126
Skew:                           1.521   Prob(JB):                    8.84e-171
Kurtosis:                       8.281   Cond. No.                         10.6
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [4]:

```
df2 = pd.DataFrame([1, 2, 3, 4], columns=["x1"])
df2
```

	x1
0	1
1	2
2	3
3	4

In [5]:

```
dmatrix("C(x1) - 1", df2)
```

```
DesignMatrix with shape (4, 4)
  C(x1)[1]  C(x1)[2]  C(x1)[3]  C(x1)[4]
         1         0         0         0
         0         1         0         0
         0         0         1         0
         0         0         0         1
  Terms:
    'C(x1)' (columns 0:4)
```

Dummy variables

A **dummy variable** is an independent variable that indicates which features exist or do not exist, with values expressed in 0 or 10,000.
It is also called:

- **Boolean indicator**
- **binary variable**
- **indicator variable**
- **design variable**
- **treatment**

* The dmatrix command in the patsy package and the from_formula method in the OLS class provide the ability to encode values of categorical variables as dummy variables using Formula strings.

In [2]:

```
df1 = pd.DataFrame(["A", "A", "B", "B"], columns=["x1"])
df1
```

	x1
0	A
1	A
2	B
3	B

In [3]:

```
dmatrix("x1 + 0", df1)
```

```
DesignMatrix with shape (4, 2)
  x1[A]  x1[B]
      1      0
      1      0
      0      1
      0      1
  Terms:
    'x1' (columns 0:2)
```

Reducec-rank

In [6]:

```
df3 = pd.DataFrame(["A", "B", "C"], columns=["x1"])
df3
```

	x1
0	A
1	B
2	C

In [7]:

```
dmatrix("x1", df3)
```

```
DesignMatrix with shape (3, 3)
  Intercept  x1[T.B]  x1[T.C]
          1        0        0
          1        1        0
          1        0        1
  Terms:
    'Intercept' (column 0)
    'x1' (columns 1:3)
```

Interaction between categorical and real independent variables

- Interaction between categorical and real independent variables

-> If we want a model that does not only change the constant term when the values of categorical variables change, but also the effects of other independent variables change, you can use interaction.

https://datascienceschool.net/view-notebook/7dda1bc9ad1c435fb309ea88f672eff9/

In [19]:

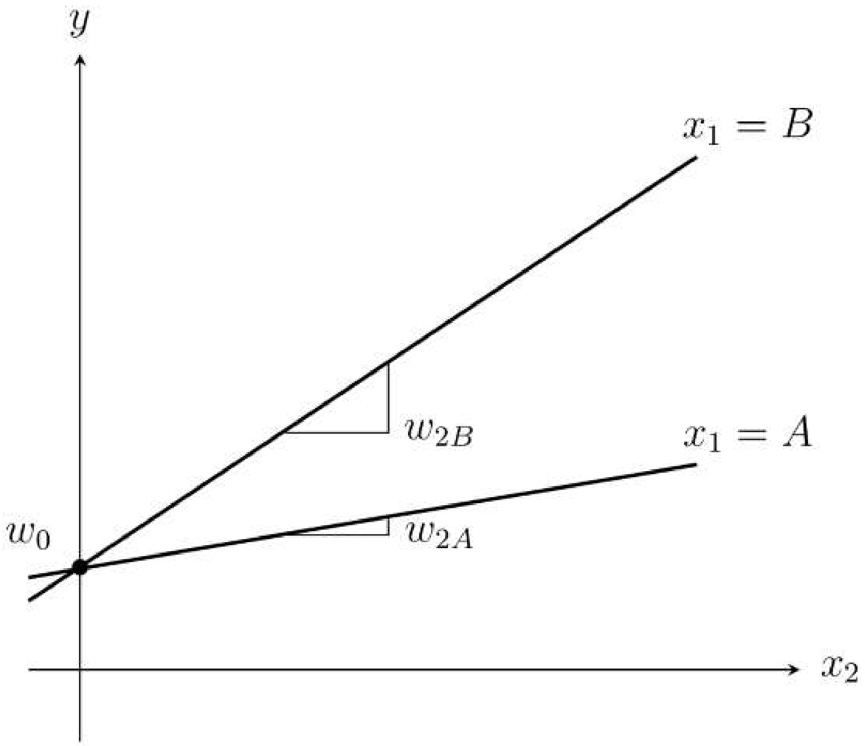
```
df6 = pd.DataFrame([["A", 1], ["B", 2], ["A", 4], ["B", 5]], columns=["x1", "x2"])
df6
```

	x1	x2
0	A	1
1	B	2
2	A	4
3	B	5

In [20]:

```
dmatrix("C(x1):x2", df6)
```

```
DesignMatrix with shape (4, 3)
  Intercept  C(x1)[A]:x2  C(x1)[B]:x2
          1            1            0
          1            0            2
          1            4            0
          1            0            5
  Terms:
    'Intercept' (column 0)
    'C(x1):x2' (columns 1:3)
```



In [21]:

```
dmatrix("C(x1) + C(x1):x2", df6)
```

```
DesignMatrix with shape (4, 4)
  Intercept  C(x1)[T.B]  C(x1)[A]:x2  C(x1)[B]:x2
          1           0            1            0
          1           1            0            2
          1           0            4            0
          1           1            0            5
  Terms:
    'Intercept' (column 0)
    'C(x1)' (column 1)
    'C(x1):x2' (columns 2:4)
```