# 미션 1

In [1]:
```
data= {
    'weight':[28,2,9,6],
    'movable':['yes','yes','yes','no'],
    'category':['animal','plant','animal','plant'], #학습을 위한 결과값
}
```

In [2]:
```
#트리 만들기1-예제에 나와있는 트리로 구성
target_index=3
target_names=['1번 생물','2번 생물','3번 생물','4번 생물']

print(target_names[target_index], ': 동물/식물을 분류합니다.')

#첫번째 질문-움직이기 가능/불가능
if data['movable'][target_index]=='yes':
    #두번째 질문-무게 확인
    if data['weight'][target_index]<=6:
        #결과값 출력
        print('plant')
    else:
        print('animal')
else:
    print('plant')
```

# 미션 2

In [3]:
```
#예시 트리와 다른 방식으로도 만들어보기

target_index=2
target_names=['1번 생물','2번 생물','3번 생물','4번 생물']

print(target_names[target_index], ': 동물/식물을 분류합니다.')

#첫번째 질문-무게 확인
if data['weight'][target_index]>=6:
    #두번째 질문-움직이기 확인
    if data['movable'][target_index]=='no':
        #결과값 출력
        print('plant')
    else:
        print('animal')
else:
    print('plant')
```

# 자율과제 1: Cabin 특성 살리기

**step 1. Cabin 객실 알파벳의 첫글자만 남기기**

In [15]:
```
import pandas as pd

#파일 불러오기
df=pd.read_csv('data/train.csv')
```

In [16]:
```
#테스트셋 만들기
df_test=pd.read_csv('data/test.csv')
```

In [17]:
```
#Cabin 자료값의 첫번째 글자만 남기기
# ==> 문자열은 리스트 형태로 한글자씩 가져올 수 있다는 특성을 이용함.

df['Cabin']=df['Cabin'].str[0] #df가 가지는 Cabin 열 값들의 첫글자만 따서 다시 할당해줌
```

In [7]:
```
#Cabin열의 빈 공백을 일정한 값으로 채워주는 작업이 필요.
#예시답안의 경우 C로 대체함.

df['Cabin']=df['Cabin'].fillna('C')

#테스트셋에도 동일하게 적용
df_test['Cabin']=df_test['Cabin'].str[0]
df_test['Cabin']=df_test['Cabin'].fillna('C')

#isnull().sum()을 사용해 Cabin열에 공백값이 있는지 확인
df_test.isnull().sum()
```

Out[7]:
```
PassengerId     0
Pclass          0
Name            0
Sex             0
Age            86
SibSp           0
Parch           0
Ticket          0
Fare            1
Cabin           0
Embarked        0
dtype: int64
```

**step 2. Cabin의 문자값 데이터를 숫자로 변환**

In [8]:
```
#value_counts()를 사용해 Cabin열에 어떤 문자값들이 있는지 + 각 문자값들의 개수 확인

df['Cabin'].value_counts()
```

Out[8]:
```
C    746
B     47
D     33
E     32
A     15
F     13
G      4
T      1
Name: Cabin, dtype: int64
```

In [9]:
```
#C,B,D,E,A,F,G,T 순으로 0~7할당

df.loc[df['Cabin']=='C','Cabin']=0
df.loc[df['Cabin']=='B','Cabin']=1
df.loc[df['Cabin']=='D','Cabin']=2
df.loc[df['Cabin']=='E','Cabin']=3
df.loc[df['Cabin']=='A','Cabin']=4
df.loc[df['Cabin']=='F','Cabin']=5
df.loc[df['Cabin']=='G','Cabin']=6
df.loc[df['Cabin']=='T','Cabin']=7
```

In [10]:
```
#테스트셋에도 적용
df_test.loc[df_test['Cabin']=='C','Cabin']=0
df_test.loc[df_test['Cabin']=='B','Cabin']=1
df_test.loc[df_test['Cabin']=='D','Cabin']=2
df_test.loc[df_test['Cabin']=='E','Cabin']=3
df_test.loc[df_test['Cabin']=='A','Cabin']=4
df_test.loc[df_test['Cabin']=='F','Cabin']=5
df_test.loc[df_test['Cabin']=='G','Cabin']=6
df_test.loc[df_test['Cabin']=='T','Cabin']=7

df_test['Cabin'].value_counts()
```

Out[10]:
```
0    362
1     18
2     13
3      9
5      8
4      7
6      1
Name: Cabin, dtype: int64
```

## 자율과제 2: 빈칸(NaN)을 더 현명하게 처리하기

**step 1. Name에서 유용한 정보 남기기**

In [11]:
```
#강의자료에서 이름에 들어가는 Mr,Mrs,Miss에 따라 생존율이 유의미하게 변동함을 발견함.
#이를 사용해서 Name이 가지는 자료값에서 타이틀 정보를 뽑아 수치화하고자함.

df.loc[ df['Name'].str.contains('Mr'), 'Name' ] = 'Mr'
df.loc[ df['Name'].str.contains('Mrs'), 'Name' ] = 'Mrs'
df.loc[ df['Name'].str.contains('Miss'), 'Name' ] = 'Miss'

df.head()
```

**>>str.contains()**

➔ 특정 문자열을 포함하는 요소들을 찾아줌.

예시의 경우,data의 Name열에서 'Mr'을 포함하는 자료값일 경우 그 칸을 전부 'Mr'로 바꾸는 작업을 함

(참고) starswith(), endswith()➔ 특정 문자열로 시작/끝나는 요소

In [12]:
```
#뽑아낸 타이틀 값에 따라 숫자 부여

df['Name'] = df['Name'].map({
    'Mr': 0,
    'Mrs': 1,
    'Miss': 2
})

#df.loc[df['Name']=='Mr₩.  ','Name']=0
#df.loc[df['Name']=='Mrs₩.  ','Name']=1
#df.loc[df['Name']=='Miss₩.  ','Name']=2

df.head(10)
```

Out[12]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 3 | 0.0 | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | 0 | S |
| **1** | 2 | 1 | 1 | 1.0 | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | 0 | C |
| **2** | 3 | 1 | 3 | 2.0 | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | 0 | S |
| **3** | 4 | 1 | 1 | 1.0 | female | 35.0 | 1 | 0 | 113803 | 53.1000 | 0 | S |
| **4** | 5 | 0 | 3 | 0.0 | male | 35.0 | 0 | 0 | 373450 | 8.0500 | 0 | S |
| **5** | 6 | 0 | 3 | 0.0 | male | NaN | 0 | 0 | 330877 | 8.4583 | 0 | Q |
| **6** | 7 | 0 | 1 | 0.0 | male | 54.0 | 0 | 0 | 17463 | 51.8625 | 3 | S |
| **7** | 8 | 0 | 3 | NaN | male | 2.0 | 3 | 1 | 349909 | 21.0750 | 0 | S |
| **8** | 9 | 1 | 3 | 1.0 | female | 27.0 | 0 | 2 | 347742 | 11.1333 | 0 | S |
| **9** | 10 | 1 | 2 | 1.0 | female | 14.0 | 1 | 0 | 237736 | 30.0708 | 0 | C |

**>>map({a:b,c:d})**

**➔ 변환할 값을 dictionary형태로 묶어 반환. 이때, ***dict안에 없는 값이 있는 경우 빈칸(NaN)으로 대체함.*****

기존에는 df.loc을 사용해서 값을 변환했지만, 바꾸고자 하는 값 이외의 자료값들은 그대로 남아있다는 점이 map과 다름.

위의 실습의 경우에는 Mr,Mrs,Miss을 포함하지 않는 다른 값들은 모두 3으로 변환해야 하기 때문에, 빈칸으로 만들고 fillna(3)을 이용해서 수치화함.

**step 2. Name의 빈칸을 3으로 채워넣기**

In [13]:
```
#공백인 경우, Mr,Mrs,Miss이외의 값이라고 판단하여 0~2의 값이 아닌 3으로 채워넣고자함.

df['Name']=df['Name'].fillna(3)
```

**step 3. 수정된 Name을 활용하여 Age값 수정하기**

In [14]:
```
#앞선 강의에서는 Age의 빈칸을 전체 평균으로 대체함.
#여기서는, 이름 타이틀에 따른 유의미한 변화를 반영하기 위해 타이틀별 나이 평균으로
#빈칸을 대체함

df['Age'] = df['Age'].fillna( df.groupby('Name')['Age'].transform('mean') )
df.head(10)
```

Out[14]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 3 | 0.0 | male | 22.00000 | 1 | 0 | A/5 21171 | 7.2500 | 0 | S |
| **1** | 2 | 1 | 1 | 1.0 | female | 38.00000 | 1 | 0 | PC 17599 | 71.2833 | 0 | C |
| **2** | 3 | 1 | 3 | 2.0 | female | 26.00000 | 0 | 0 | STON/O2. 3101282 | 7.9250 | 0 | S |
| **3** | 4 | 1 | 1 | 1.0 | female | 35.00000 | 1 | 0 | 113803 | 53.1000 | 0 | S |
| **4** | 5 | 0 | 3 | 0.0 | male | 35.00000 | 0 | 0 | 373450 | 8.0500 | 0 | S |
| **5** | 6 | 0 | 3 | 0.0 | male | 32.36809 | 0 | 0 | 330877 | 8.4583 | 0 | Q |
| **6** | 7 | 0 | 1 | 0.0 | male | 54.00000 | 0 | 0 | 17463 | 51.8625 | 3 | S |
| **7** | 8 | 0 | 3 | 3.0 | male | 2.00000 | 3 | 1 | 349909 | 21.0750 | 0 | S |
| **8** | 9 | 1 | 3 | 1.0 | female | 27.00000 | 0 | 2 | 347742 | 11.1333 | 0 | S |
| **9** | 10 | 1 | 2 | 1.0 | female | 14.00000 | 1 | 0 | 237736 | 30.0708 | 0 | C |

**>>groupby('Name')['Age'].transform('mean')**

**➔name열에서 age별로 그룹을 만들고, 각 그룹의 평균을 냄.**

**참고:** https://rfriend.tistory.com/403

**<<복사용>>**

```
data= {
    'weight':[28,2,9,6],
    'movable':['yes','yes','yes','no'],
    'category':['animal','plant','animal','plant'], #학습을 위한 결과값
}
#트리 만들기1-예제에 나와있는 트리로 구성
target_index=3
target_names=['1번 생물','2번 생물','3번 생물','4번 생물']

print(target_names[target_index], ': 동물/식물을 분류합니다.')

#첫번째 질문-움직이기 가능/불가능
if data['movable'][target_index]=='yes':
    #두번째 질문-무게 확인
    if data['weight'][target_index]<=6:
        #결과값 출력
        print('plant')
    else:
        print('animal')
else:
    print('plant')

#예시 트리와 다른 방식으로도 만들어보기
```

```
target_index=2
target_names=['1번 생물','2번 생물','3번 생물','4번 생물']

print(target_names[target_index], ': 동물/식물을 분류합니다.')

#첫번째 질문-무게 확인
if data['weight'][target_index]>=6:
    #두번째 질문-움직이기 확인
    if data['movable'][target_index]=='no':
        #결과값 출력
        print('plant')
    else:
        print('animal')
else:
    print('plant')
```

# 자율과제 1: Cabin 특성 살리기

**step 1. Cabin 객실 알파벳의 첫글자만 남기기**

```
import pandas as pd

#파일 불러오기
df=pd.read_csv('data/train.csv')

#테스트셋 만들기
df_test=pd.read_csv('data/test.csv')

#Cabin 자료값의 첫번째 글자만 남기기
```

# ==> 문자열은 리스트 형태로 한글자씩 가져올 수 있다는 특성을 이용함.

df['Cabin']=df['Cabin'].str[0] #df가 가지는 Cabin 열 값들의 첫글자만 따서 다시 할당해줌

#Cabin열의 빈 공백을 일정한 값으로 채워주는 작업이 필요.

#예시답안의 경우 C로 대체함.

df['Cabin']=df['Cabin'].fillna('C')

#테스트셋에도 동일하게 적용

df_test['Cabin']=df_test['Cabin'].str[0]

df_test['Cabin']=df_test['Cabin'].fillna('C')

#isnull().sum()을 사용해 Cabin열에 공백값이 있는지 확인

df_test.isnull().sum()

**step 2. Cabin의 문자값 데이터를 숫자로 변환**

#value_counts()를 사용해 Cabin열에 어떤 문자값들이 있는지 + 각 문자값들의 개수 확인

df['Cabin'].value_counts()

#C,B,D,E,A,F,G,T 순으로 0~7할당

df.loc[df['Cabin']=='C','Cabin']=0

df.loc[df['Cabin']=='B','Cabin']=1

df.loc[df['Cabin']=='D','Cabin']=2

df.loc[df['Cabin']=='E','Cabin']=3

df.loc[df['Cabin']=='A','Cabin']=4

```
df.loc[df['Cabin']=='F','Cabin']=5

df.loc[df['Cabin']=='G','Cabin']=6

df.loc[df['Cabin']=='T','Cabin']=7


#테스트셋에도 적용

df_test.loc[df_test['Cabin']=='C','Cabin']=0

df_test.loc[df_test['Cabin']=='B','Cabin']=1

df_test.loc[df_test['Cabin']=='D','Cabin']=2

df_test.loc[df_test['Cabin']=='E','Cabin']=3

df_test.loc[df_test['Cabin']=='A','Cabin']=4

df_test.loc[df_test['Cabin']=='F','Cabin']=5

df_test.loc[df_test['Cabin']=='G','Cabin']=6

df_test.loc[df_test['Cabin']=='T','Cabin']=7


df_test['Cabin'].value_counts()
```

# 자율과제 2: 빈칸(NaN)을 더 현명하게 처리하기

**step 1. Name에서 유용한 정보 남기기**

```
#강의자료에서 이름에 들어가는 Mr,Mrs,Miss에 따라 생존율이 유의미하게 변동함을 발견함.

#이를 사용해서 Name이 가지는 자료값에서 타이틀 정보를 뽑아 수치화하고자함.


df.loc[ df['Name'].str.contains('Mr'), 'Name' ] = 'Mr'

df.loc[ df['Name'].str.contains('Mrs'), 'Name' ] = 'Mrs'

df.loc[ df['Name'].str.contains('Miss'), 'Name' ] = 'Miss'


df.head()
```

#뽑아낸 타이틀 값에 따라 숫자 부여

```
df['Name'] = df['Name'].map({
    'Mr': 0,
    'Mrs': 1,
    'Miss': 2
})
```

```
#df.loc[df['Name']=='Mr₩. ','Name']=0
#df.loc[df['Name']=='Mrs₩. ','Name']=1
#df.loc[df['Name']=='Miss₩. ','Name']=2
```

```
df.head(10)
```

**step 2. Name의 빈칸을 3으로 채워넣기**

#공백인 경우, Mr,Mrs,Miss이외의 값이라고 판단하여 0~2의 값이 아닌 3으로 채워넣고자함.

```
df['Name']=df['Name'].fillna(3)
```

**step 3. 수정된 Name을 활용하여 Age값 수정하기**

#앞선 강의에서는 Age의 빈칸을 전체 평균으로 대체함.

#여기서는, 이름 타이틀에 따른 유의미한 변화를 반영하기 위해 타이틀별 나이 평균으로

#빈칸을 대체함

```
df['Age'] = df['Age'].fillna( df.groupby('Name')['Age'].transform('mean') )
df.head(10)
```