

데이터 탐색해 보기(시각화)

Kaggle 대회 (Bike Sharing Demand)

- URL : <https://www.kaggle.com/>
- Competitions 선택하면 다양한 대회 확인 가능.
- 대회 주제 : Bike Sharing Demand
- <https://www.kaggle.com/c/bike-sharing-demand>

데이터 다운로드하기

- 가. <https://www.kaggle.com/c/bike-sharing-demand> 링크를 선택하여 웹 사이트 접속합니다.
- 나. Data를 선택합니다.
- 다. train.csv, test.csv, sampleSubmission.csv를 다운로드 받습니다.
- 라. 다운로드 받은 csv와 주피터 노트북 또는 py 파일은 동일한 폴더에 위치시킵니다.



같은 폴더내에 데이터 파일(csv파일)과 python 노트북 파일을 위치시킵니다.

Data Fields

필드명	설명
datetime	hourly date + timestamp
season	1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday	whether the day is considered a holiday
workingday	whether the day is neither a weekend nor holiday
weather	1: Clear, Few clouds, Partly cloudy, Partly cloudy 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp	temperature in Celsius (온도)
atemp	"feels like" temperature in Celsius (체감온도)
humidity	relative humidity (습도)
windspeed	wind speed (바람속도)
casual	number of non-registered user rentals initiated (비가입자 사용유저)
registered	number of registered user rentals initiated (가입자 사용유저)
count	number of total rentals (전체 렌탈 대수)

1-1 라이브러리 불러오기

```
In [1]: import pandas as pd
import matplotlib.pyplot as plt ## seaborn 보다 고급 시작화 가능. but 코드 복잡
import seaborn as sns ## seaborn은 matplotlib보다 간단하게 사용 가능
```

1-2 데이터 준비 및 데이터 탐색

- train 은 학습을 위한 데이터 셋
- test 은 예측을 위한 데이터 셋
- ../data/bike : 상위폴더의 (data/bike 폴더 경로), 내 컴퓨터의 데이터 경로 지정.
- parse_dates = [컬럼명] : 해당 컬럼을 시간형 자료로 불러옴.

```
In [2]: train = pd.read_csv("../data/bike/train.csv", parse_dates=['datetime'])
test = pd.read_csv("../data/bike/test.csv", parse_dates=['datetime'])
```

1-3 간단한 데이터 탐색

- 데이터 행과 열은? (shape)
- 데이터의 개수와 자료형? (info())
- 데이터의 컬럼명은 무엇일까? (columns)
- 몇행만 데이터를 확인해 보자. (head)

```
In [3]: print(train.shape)
print(test.shape)
```

```
(10886, 12)
(6493, 9)
```

```
In [4]: print(train.info())
print() # 한줄 공백
print(test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 # ... Column ... Non-Null Count Dtype ...
--- ... ...
0 ... datetime ... 10886 non-null datetime64[ns]
1 ... season ... 10886 non-null int64 ...
2 ... holiday ... 10886 non-null int64 ...
3 ... workingday ... 10886 non-null int64 ...
4 ... weather ... 10886 non-null int64 ...
5 ... temp ... 10886 non-null float64 ...
6 ... atemp ... 10886 non-null float64 ...
7 ... humidity ... 10886 non-null int64 ...
8 ... windspeed ... 10886 non-null float64 ...
9 ... casual ... 10886 non-null int64 ...
10 ... registered ... 10886 non-null int64 ...
11 ... count ... 10886 non-null int64 ...
dtypes: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.7 KB
None
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6493 entries, 0 to 6492
Data columns (total 9 columns):
 # ... Column ... Non-Null Count Dtype ...
--- ... ...
0 ... datetime ... 6493 non-null datetime64[ns]
1 ... season ... 6493 non-null int64 ...
2 ... holiday ... 6493 non-null int64 ...
3 ... workingday ... 6493 non-null int64 ...
4 ... weather ... 6493 non-null int64 ...
5 ... temp ... 6493 non-null float64 ...
6 ... atemp ... 6493 non-null float64 ...
7 ... humidity ... 6493 non-null int64 ...
8 ... windspeed ... 6493 non-null float64 ...
dtypes: datetime64[ns](1), float64(3), int64(5)
memory usage: 456.7 KB
None
```

- 기본적으로 데이터 셋은 실수형(float)과 정수형(int)로 이루어져 있다.
- non-null : 결측치가 없음.

```
In [5]: print(train.columns)
print()
print(test.columns)
```

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed'],
      dtype='object')
```

```
In [6]: train.head()
```

Out[6]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0

◀ ▶

In [7]: `train.tail()`

Out[7]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	cas
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	

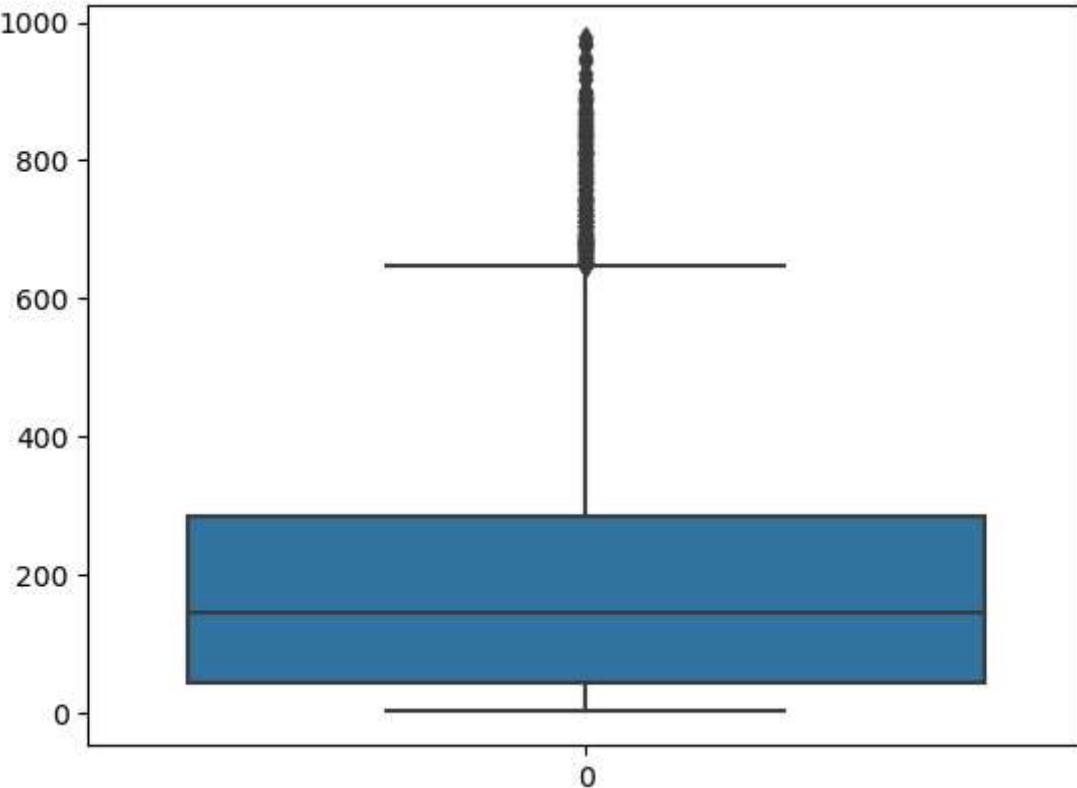
◀ ▶

1-4 데이터 시각화를 통한 데이터 이해

(A) Count- Rental 대수 boxplot(상자그림) 보기

In [8]: `print(train['count'].describe())`
`sns.boxplot(train['count'])`

```
count      10886.000000
mean       191.574132
std        181.144454
min        1.000000
25%        42.000000
50%        145.000000
75%        284.000000
max        977.000000
Name: count, dtype: float64
<Axes: >
```



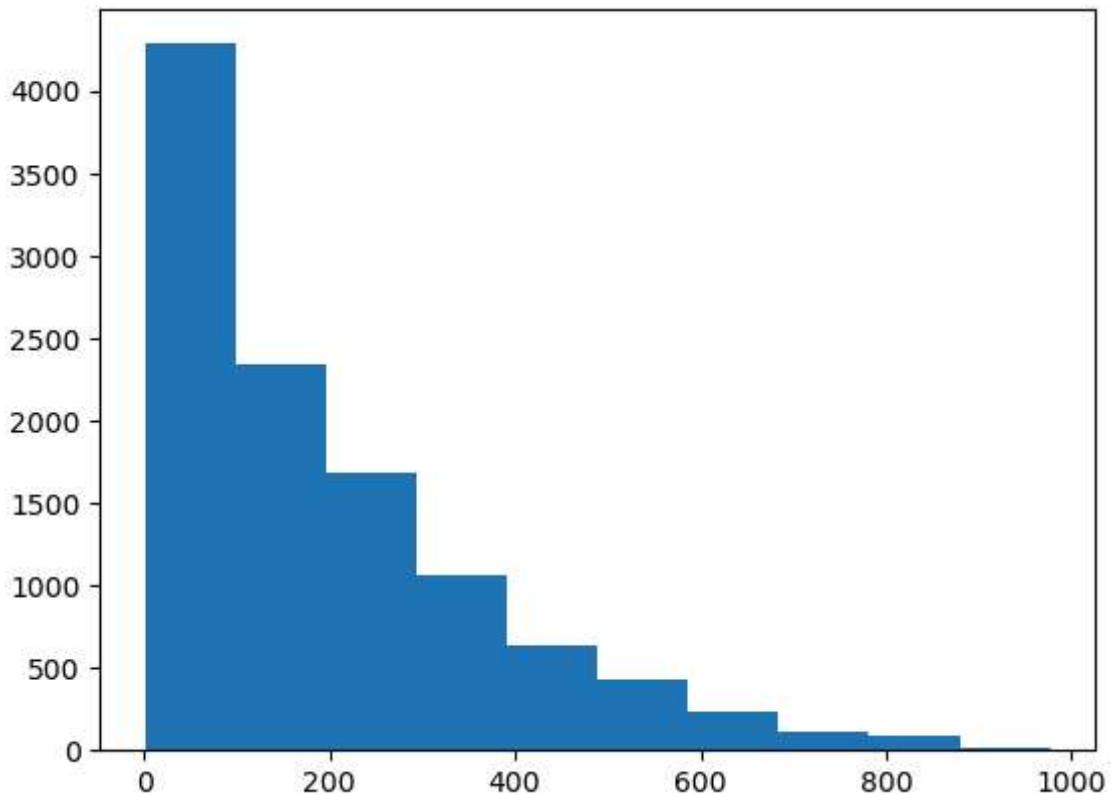
- 데이터(렌탈 대수의 분포) 확인 결과 25% ~ 75% 는 42대~284대로 분포해 있음.
- Boxplot상으로의 아웃라이어로 보여지는 것도 확인된다.

(B) Count- Rental 대수 boxplot 보기

- 연속형 데이터에 대한 히스토그램도 그려보자.
- 이번에는 matplotlib.pyplot 의 hist를 활용해 본다.

```
In [9]: plt.hist(train['count'])
```

```
Out[9]: (array([4284., 2337., 1686., 1067., 633., 426., 233., 116., 85.,
... 19.]),
array([ 1., 98.6, 196.2, 293.8, 391.4, 489., 586.6, 684.2, 781.8,
879.4, 977.]),
<BarContainer object of 10 artists>)
```



데이터를 통해 확인된 범주형 데이터에 대한 막대그래프 확인해 보기

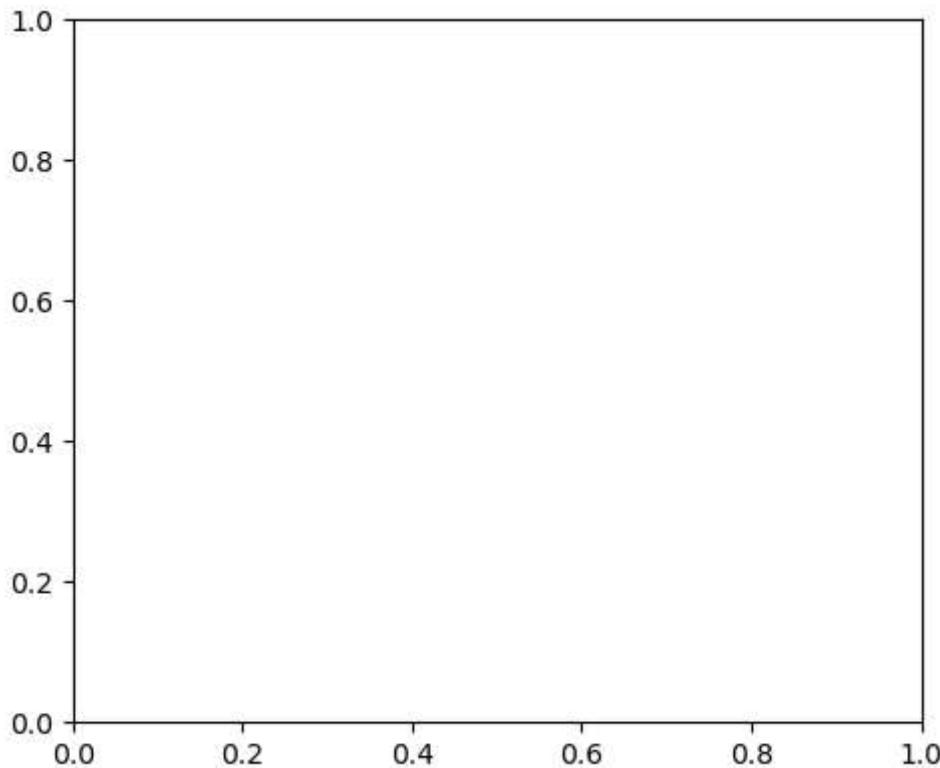
```
In [10]: col_names = [ 'season', 'holiday', 'workingday', 'weather' ]
i = 0
plt.figure(figsize=(12,10)) # 전체 그래프의 크기 지정 (가로, 세로)

for name in col_names:      # 컬럼명을 전달 리스트 수 만큼 반복 -> 4회
    i = i + 1               # 숫자를 1씩 증가.
    plt.subplot(2,2,i)       # 2행 2열에 i번째 그래프 선택
    sns.countplot(name, data=train) # i번째 그래프에 sns.countplot를 그리겠다.

# 주피터에서 아래 행이 필요없지만,
# 다른곳(editor, pycharm)에서는(*.py 파일) plt.show()이걸 실행시켜야 한다.
plt.show()
```

```
-----
-
TypeError Traceback (most recent call last)
Cell In[10], line 8
  6     i = i + 1           # 숫자를 1씩 증가.
  7     plt.subplot(2,2,i)   # 2행 2열에 i번째 그래프 선택
----> 8     sns.countplot(name, data=train) # i번째 그래프에 sns.countplot를 그리겠다.
  9 # 주피터에서 아래 행이 필요없지만,
 10 # 다른곳(editor, pycharm)에서는(*.py 파일) plt.show()이걸 실행시켜야 한다.
 11 plt.show()

TypeError: countplot() got multiple values for argument 'data'
```

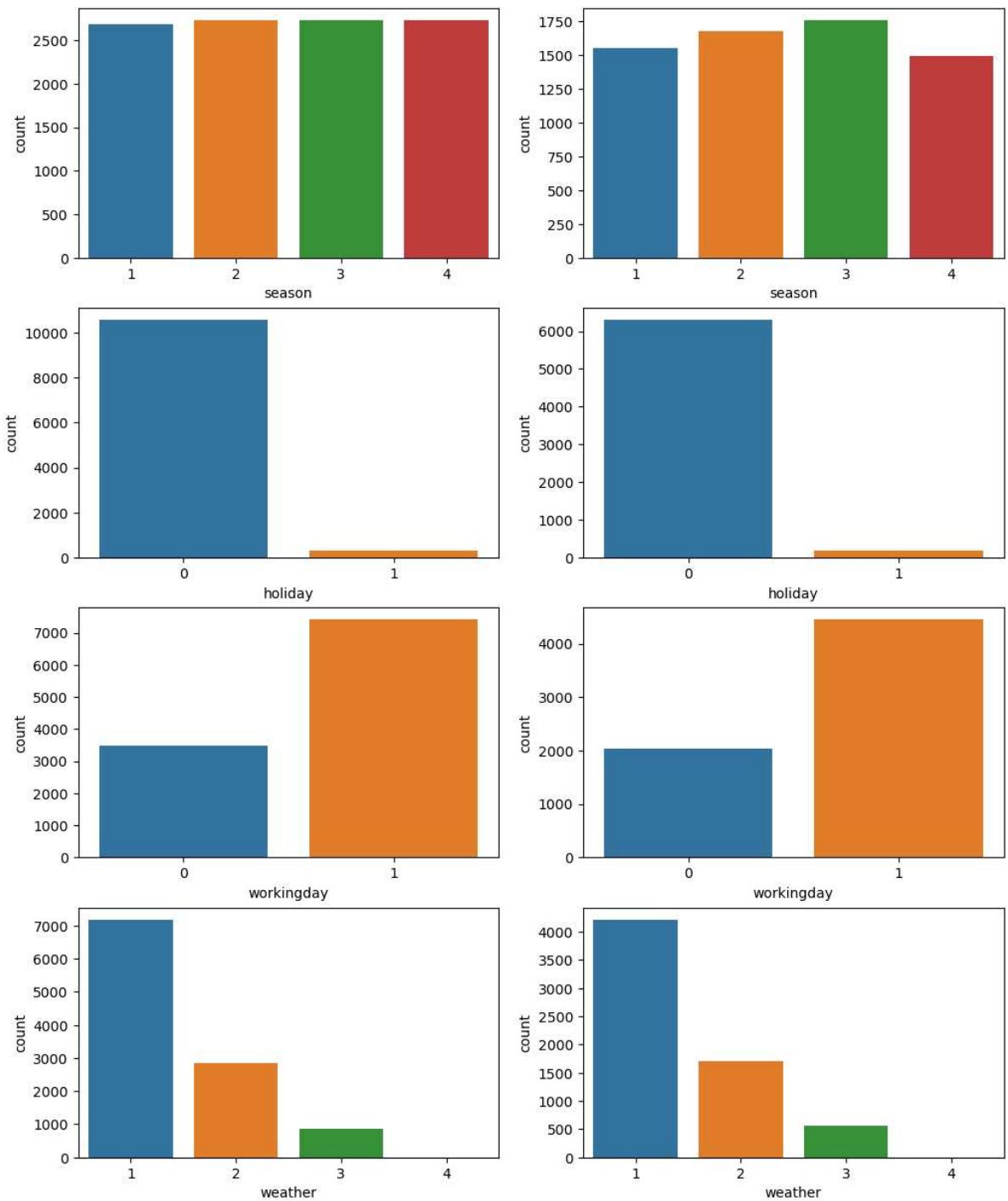


train과 **test**를 비교해서 확인해 보자

```
In [12]: col_names = [ 'season', 'holiday', 'workingday', 'weather' ]
i = 0
plt.figure(figsize=(12,15)) # 전체 그래프의 크기 지정 (가로, 세로)

for name in col_names:      # 컬럼명을 전달 리스트 수 만큼 반복 -> 4회
    i = i + 1               # 숫자를 1씩 증가.
    plt.subplot(4,2,i*2-1)   # 2행 2열에 i번째 그래프 선택
    sns.countplot(x=name, data=train) # i번째 그래프에 sns.countplot를 그리겠다.
    plt.subplot(4,2,i*2)       # 2행 2열에 i번째 그래프 선택
    sns.countplot(x=name, data=test) # i번째 그래프에 sns.countplot를 그리겠다.

# 주피터에서 아래 행이 필요없지만,
# 다른곳(editor, pycharm)에서는 (*.py 파일) plt.show()이걸 실행시켜야 한다.
plt.show()
```



- train과 test의 비교시, test에 season별 렌탈 대수 분포가 다른 것을 볼 수 있다.

데이터를 통해 확인된 범주형 데이터에 대한 상자그림 확인해 보기

```
In [13]: #### temp, atemp, humidity, windspeed
num_names = ['temp', 'atemp', 'humidity', 'windspeed']
train.columns
```

```
Out[13]: Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
       dtype='object')
```

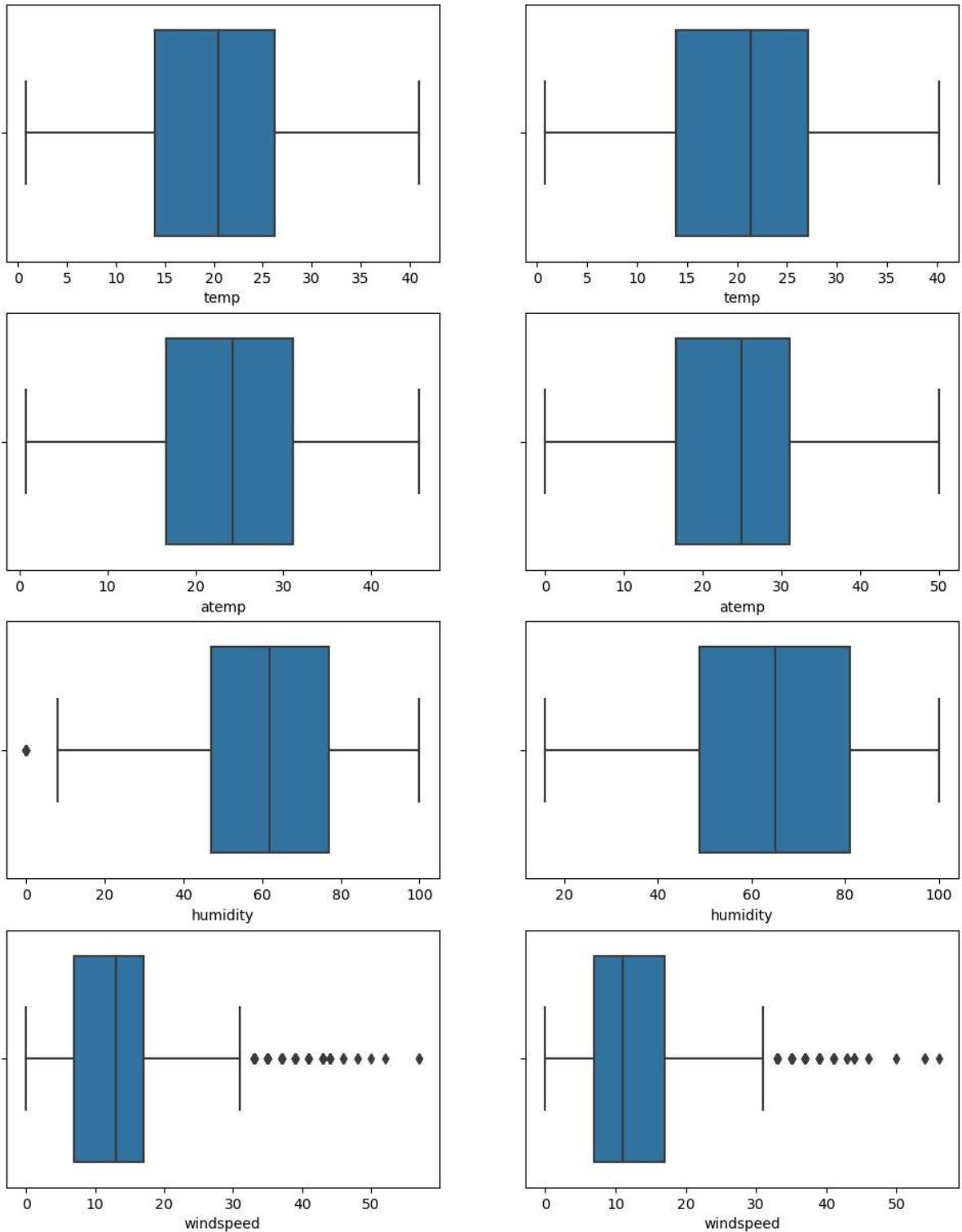
```
In [15]: # par(mfrow=c(2,2)) -> R
i = 0
plt.figure(figsize=(12,15)) # 전체 그래프의 크기 지정 (가로, 세로)
```

```

for name in num_names:          # 컬럼명을 전달 리스트 수 만큼 반복 -> 4회
    i = i + 1                  # 숫자를 1씩 증가.
    plt.subplot(4,2,i*2-1)      # 2행 2열에 i번째 그래프 선택
    sns.boxplot(x=name, data=train) # i번째 그래프에 sns.countplot를 그리겠다.
    plt.subplot(4,2,i*2)        # 2행 2열에 i번째 그래프 선택
    sns.boxplot(x=name, data=test) # i번째 그래프에 sns.countplot를 그리겠다.

plt.show()

```



- 전반적으로 값의 분포가 비슷하며, 간간히 이상치가 보인다.

1-5 데이터 새로운 컬럼을 만들어보기

```
In [16]: new_tr = train # 데이터 백업  
new_test = test  
new_tr.columns
```

```
Out[16]: Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',  
... 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],  
... dtype='object')
```

데이터셋(백업)

```
In [17]: new_tr = train  
new_test = test  
new_tr.columns
```

```
Out[17]: Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',  
... 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],  
... dtype='object')
```

```
In [18]: new_tr.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 10886 entries, 0 to 10885  
Data columns (total 12 columns):  
 # ... Column ... Non-Null Count Dtype ...  
--- ...  
 0 ... datetime ... 10886 non-null datetime64[ns]  
 1 ... season ... 10886 non-null int64 ...  
 2 ... holiday ... 10886 non-null int64 ...  
 3 ... workingday ... 10886 non-null int64 ...  
 4 ... weather ... 10886 non-null int64 ...  
 5 ... temp ... 10886 non-null float64 ...  
 6 ... atemp ... 10886 non-null float64 ...  
 7 ... humidity ... 10886 non-null int64 ...  
 8 ... windspeed ... 10886 non-null float64 ...  
 9 ... casual ... 10886 non-null int64 ...  
 10 ... registered ... 10886 non-null int64 ...  
 11 ... count ... 10886 non-null int64 ...  
dtypes: datetime64[ns](1), float64(3), int64(8)  
memory usage: 1020.7 KB
```

```
In [19]: new_tr['datetime'].dt.day.head()
```

```
Out[19]:  
0 ... 1  
1 ... 1  
2 ... 1  
3 ... 1  
4 ... 1  
Name: datetime, dtype: int32
```

```
In [20]: new_test['datetime'].dt.day.head()
```

```
Out[20]:  
0 ... 20  
1 ... 20  
2 ... 20  
3 ... 20  
4 ... 20  
Name: datetime, dtype: int32
```

날짜에서 년도를 정보를 갖는 컬럼 만들기

```
In [21]: ## 더미변수, 파생변수 생성  
new_tr['year'] = new_tr['datetime'].dt.year  
new_tr.head()
```

Out[21]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0

In [22]:

```
new_tr['month'] = new_tr['datetime'].dt.month
new_tr['day'] = new_tr['datetime'].dt.day
new_tr['hour'] = new_tr['datetime'].dt.hour
new_tr['minute'] = new_tr['datetime'].dt.minute
new_tr['second'] = new_tr['datetime'].dt.second
new_tr['dayofweek'] = new_tr['datetime'].dt.dayofweek
new_tr[ ['datetime', 'year', 'month', 'day', 'hour', 'dayofweek']]
```

Out[22]:

	datetime	year	month	day	hour	dayofweek
0	2011-01-01 00:00:00	2011	1	1	0	5
1	2011-01-01 01:00:00	2011	1	1	1	5
2	2011-01-01 02:00:00	2011	1	1	2	5
3	2011-01-01 03:00:00	2011	1	1	3	5
4	2011-01-01 04:00:00	2011	1	1	4	5
...
10881	2012-12-19 19:00:00	2012	12	19	19	2
10882	2012-12-19 20:00:00	2012	12	19	20	2
10883	2012-12-19 21:00:00	2012	12	19	21	2
10884	2012-12-19 22:00:00	2012	12	19	22	2
10885	2012-12-19 23:00:00	2012	12	19	23	2

10886 rows × 6 columns

생성된 컬럼에 대한 시각화

In [23]:

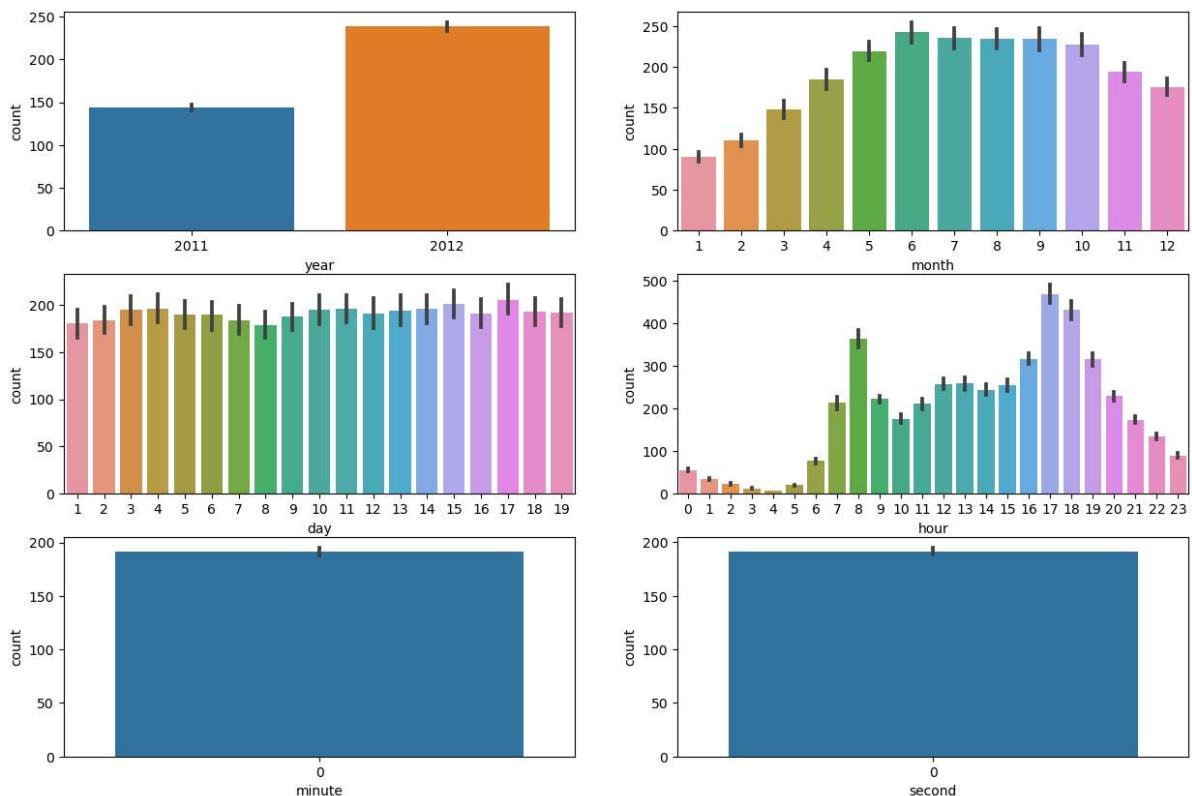
```
train.columns
```

```
Out[23]: Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count',
       'year', 'month', 'day', 'hour', 'minute', 'second', 'dayofweek'],
      dtype='object')
```

```
In [24]: datetime_names = ['year', 'month', 'day', 'hour', 'minute', 'second']
```

```
i=0
plt.figure(figsize=(15,10))
for name in datetime_names:
    i = i + 1
    plt.subplot(3,2,i)
    sns.barplot(x=name, y='count', data=new_tr)

plt.show()
```



확인

- 2011년, 2012년이 더 많다. (성장했는가?)
- 여름이 많다.
- day는 고른 분포를 보인다.
- hour는 8시, 17,18시대에 많다. (새벽 시간대도 있구나... 음.)
- minute, second는 0 데이터 의미가 없음.

실습해 보기

- 테스트 데이터 셋에 대해 확인

```
In [25]: new_test['year'] = new_test['datetime'].dt.year
new_test['month'] = new_test['datetime'].dt.month
new_test['day'] = new_test['datetime'].dt.day
new_test['dayofweek'] = new_test['datetime'].dt.dayofweek
new_test['hour'] = new_test['datetime'].dt.hour
```

```
new_test['minute'] = new_test['datetime'].dt.minute
new_test['second'] = new_test['datetime'].dt.second
```

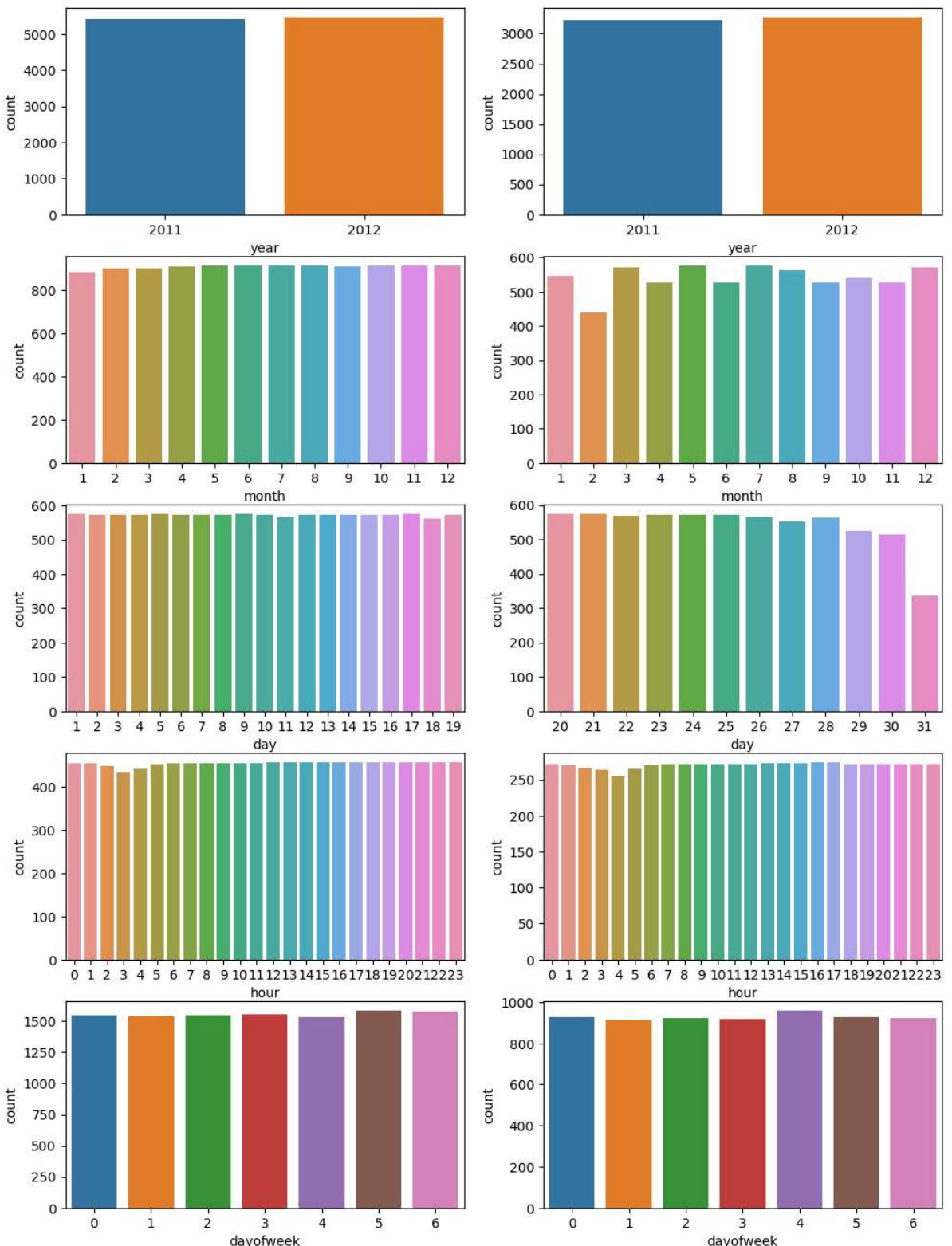
```
In [27]: col_names = ['year', 'month', 'day', 'hour', 'dayofweek']
i = 0

plt.figure(figsize=(12,20)) ##전체 그래프 크기 지정

for name in col_names: ## 컬럼명으로 반복
    i = i+1
    plt.subplot(6,2,i) ##2행2열, i = 1,2,3,4 (왼쪽 상단부터 시계방향으로 순번 지정)
    sns.countplot(x=name, data = new_tr)

    i = i+1
    plt.subplot(6,2,i) ##2행2열, i = 1,2,3,4 (왼쪽 상단부터 시계방향으로 순번 지정)
    sns.countplot(x=name, data = new_test)

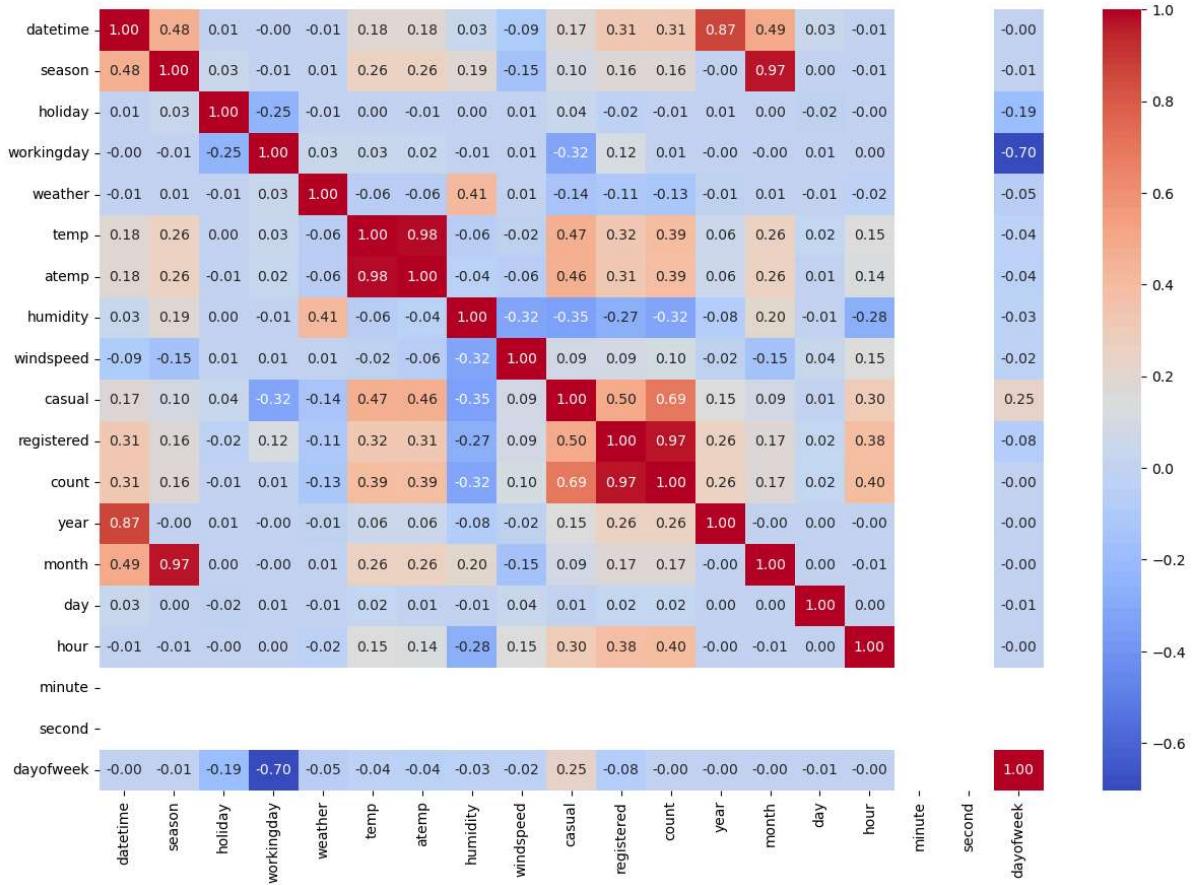
plt.show()
```



- day 1~19 일... 20일이 없네요.(test)
- test의 경우 월별 변동이 있는 편.
- test의 경우, day의 말일에 줄어드는 경향이 있음.

Heatmap를 통한 상관계수 확인

```
In [28]: plt.figure(figsize=(15, 10))
g = sns.heatmap(new_tr.corr(), annot=True, fmt=".2f", cmap="coolwarm")
```



```
In [29]: plt.figure(figsize=(15,10))
g = sns.heatmap(new_test.corr(), annot=True, fmt=".2f", cmap="coolwarm")
```

