

Kaggle 입문하기 - 데이터 분석 입문

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)

In [10]:



```
import pandas as pd
import matplotlib.pyplot as plt ## seaborn 보다 고급 시각화 가능. but 코드 복잡
import seaborn as sns          ## seaborn은 matplotlib보다 간단하게 사용 가능
```

In [12]:



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

In [13]:



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

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.

plt.show() # 주피터에서 보여주지만, 다른곳(editor, pycharm)에서는 이걸 실행시켜야 한다.
```

C:\Users\Wtoto\Anaconda3\lib\site-packages\seaborn\decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

warnings.warn(

C:\Users\Wtoto\Anaconda3\lib\site-packages\seaborn\decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

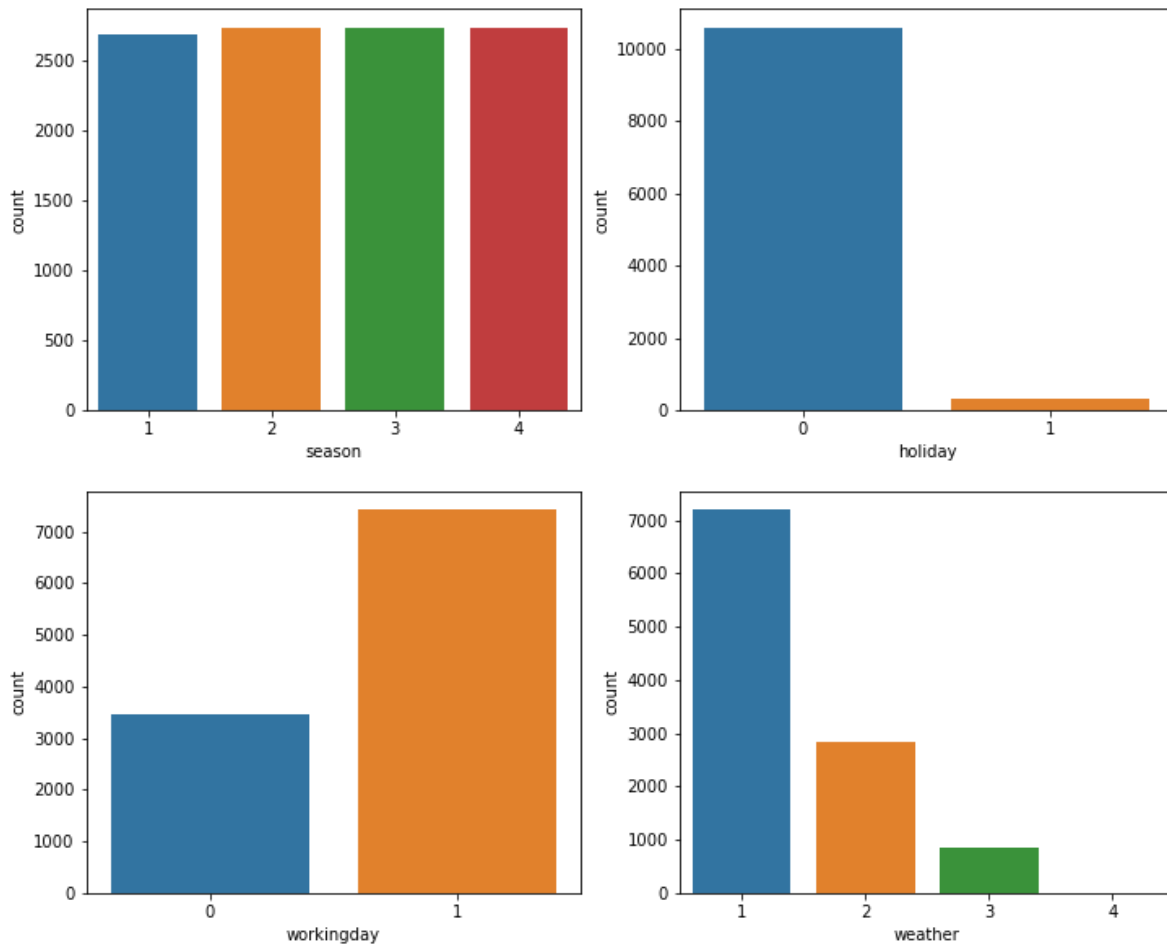
warnings.warn(

C:\Users\Wtoto\Anaconda3\lib\site-packages\seaborn\decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

warnings.warn(

C:\Users\Wtoto\Anaconda3\lib\site-packages\seaborn\decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

warnings.warn(



수치형 데이터 선택

In [14]:

```
### temp, atemp, humidity, windspeed

num_names = ['temp', 'atemp', 'humidity', 'windspeed']
train.columns
```

Out[14]:

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

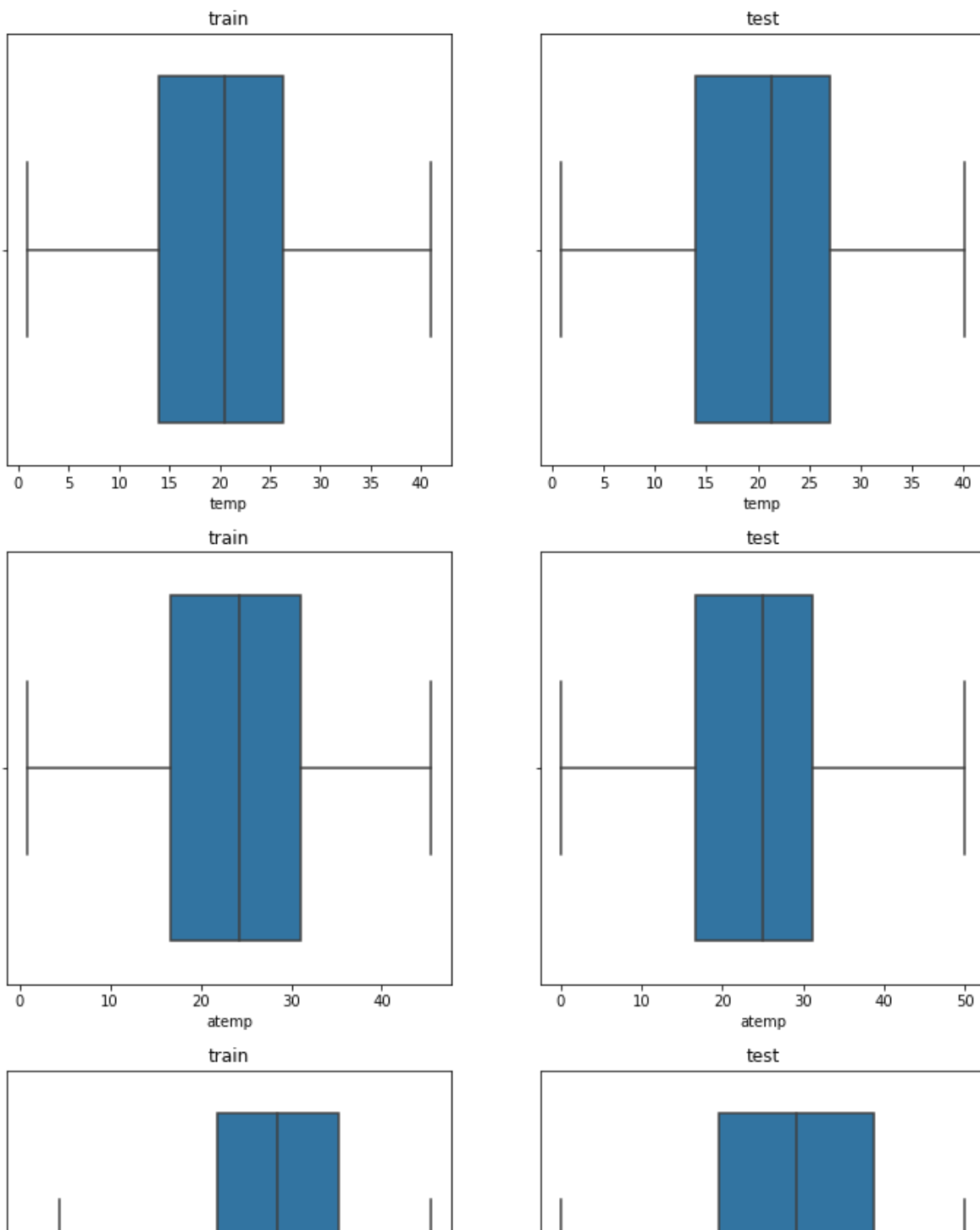
In [16]:

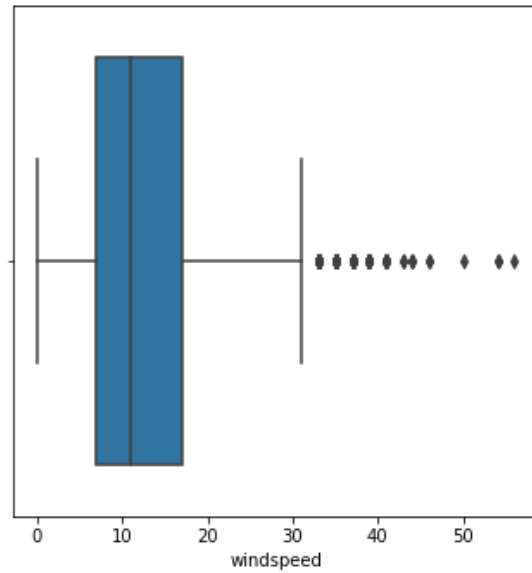
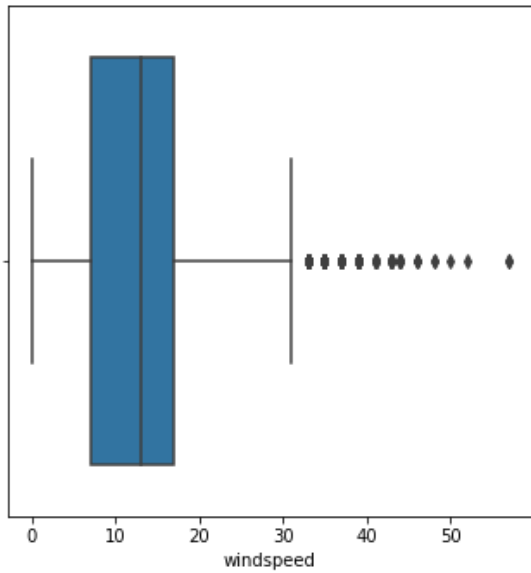
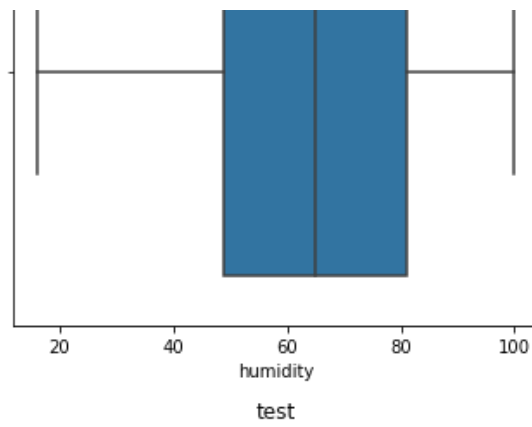
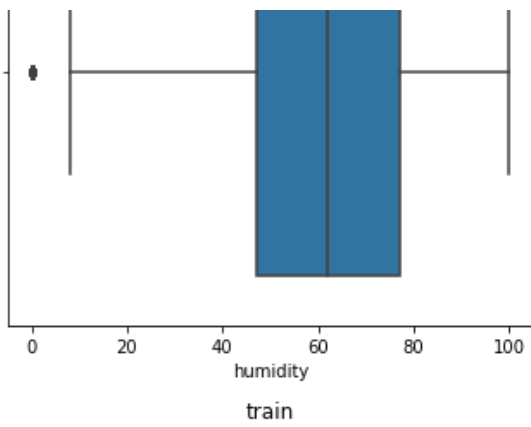


```
# par(mfrow=c(2,2)) -> R

i = 0
plt.figure(figsize=(12,25)) # 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.title("train")
    plt.subplot(4,2,i*2)     # 2행 2열에 i번째 그래프 선택
    sns.boxplot(x=name, data=test) # i번째 그래프에 sns.countplot를 그리겠다.
    plt.title("test")
plt.show()
```





In [17]:



```
new_tr = train.copy() # 데이터 백업
new_test = test.copy()
new_tr.columns
```

Out[17]:

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

파생변수

In [18]:



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

Out[18]:

	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 [19]:



```
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.head()
```

Out[19]:

	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 [20]:



```
train.columns
```

Out[20]:

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

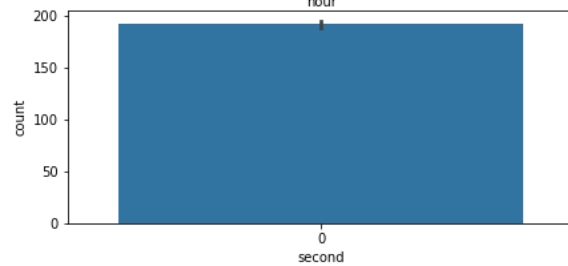
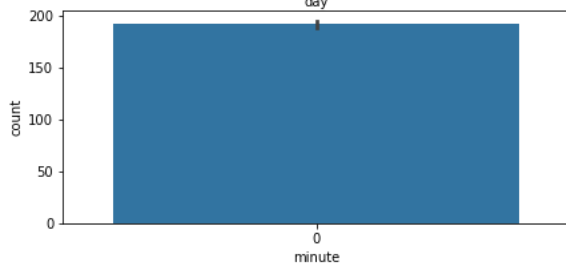
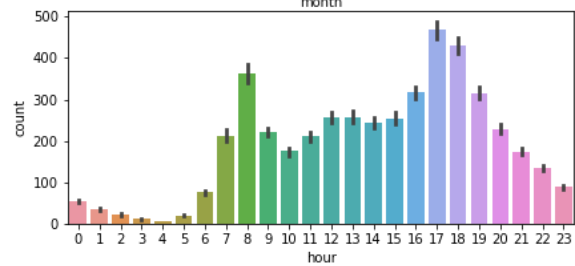
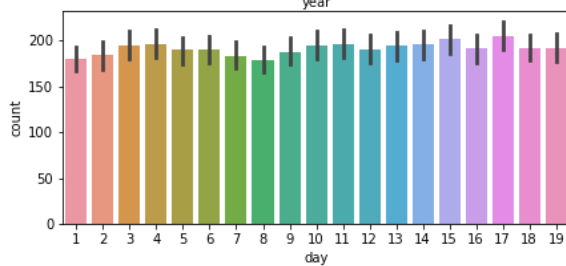
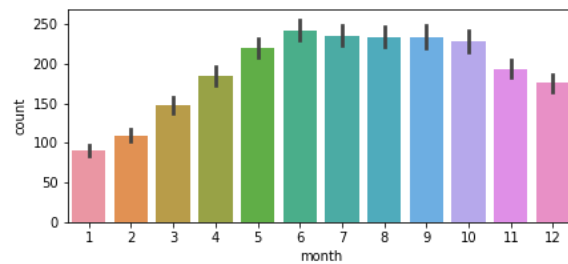
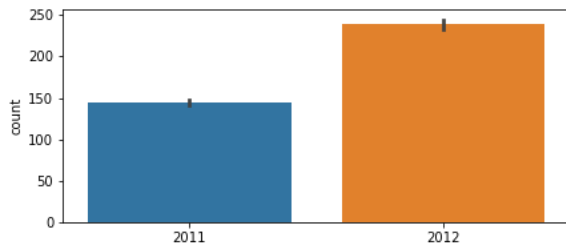
In [21]:



```
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()
```



In [22]:



```
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 [28]:



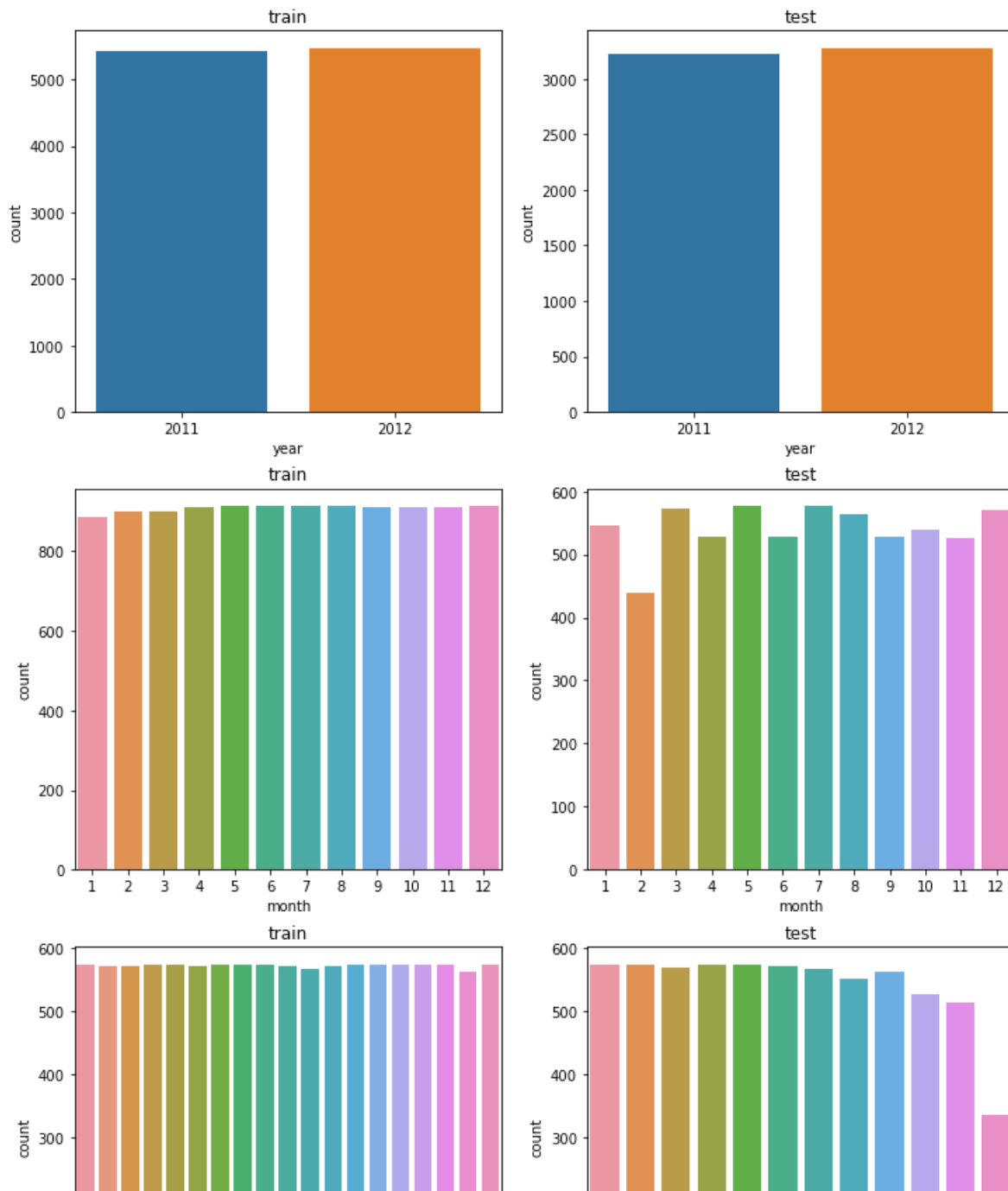
```
col_names = ['year', 'month', 'day', 'hour', 'dayofweek']
i = 0

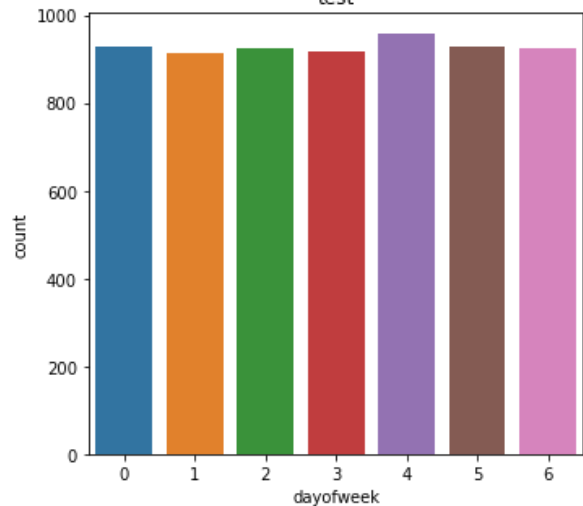
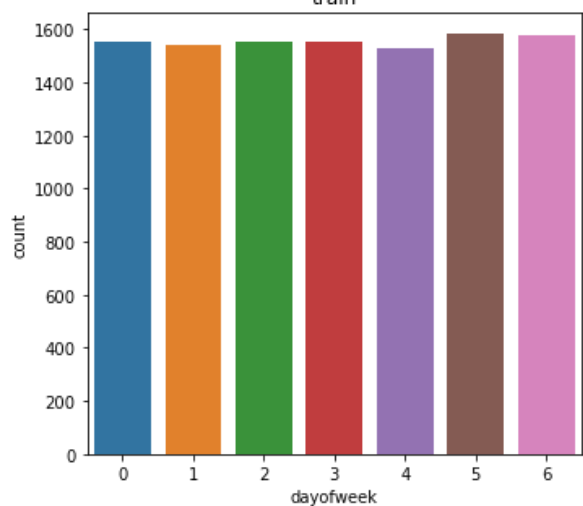
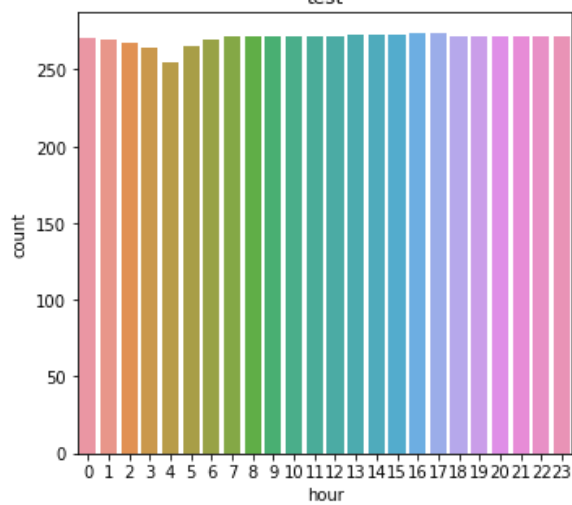
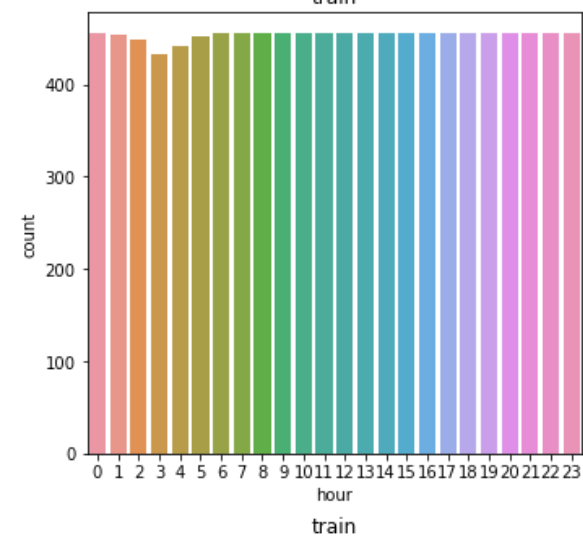
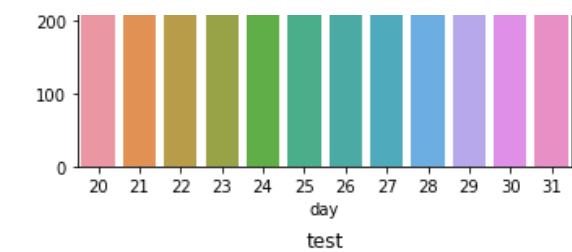
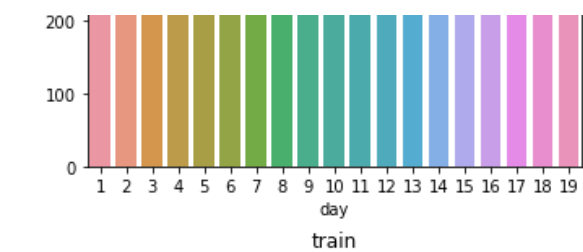
plt.figure(figsize=(12,35)) ##전체 그래프 크기 지정

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)
    plt.title('train')

    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.title('test')

plt.show()
```





In [29]:

```
new_tr['dayofweek'] = new_tr['datetime'].dt.dayofweek # Monday=0, Sunday=6
```



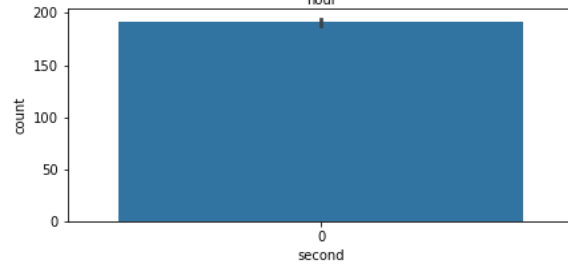
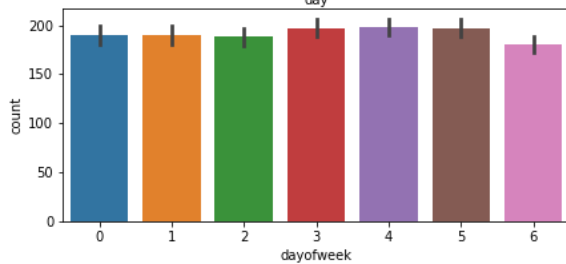
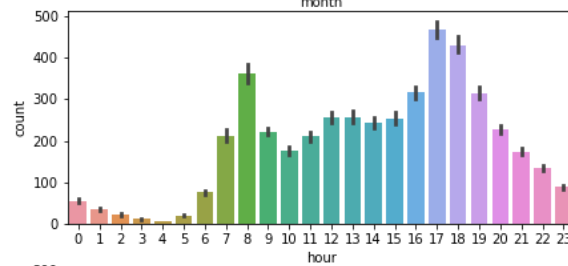
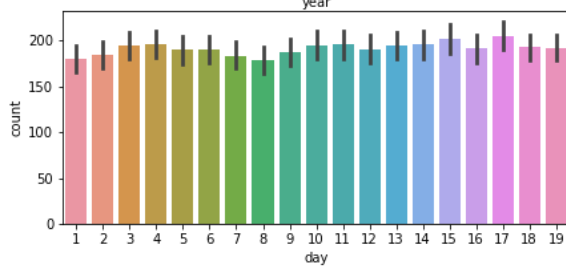
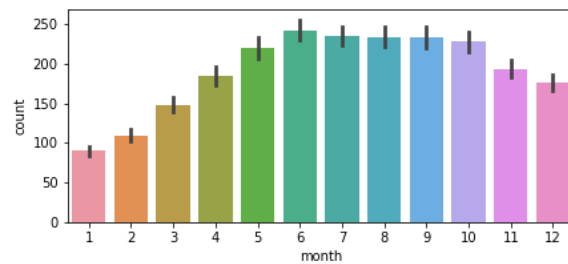
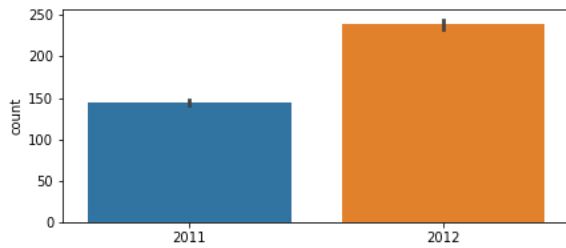
In [30]:



```
datetime_names = ['year', 'month', 'day', 'hour', 'dayofweek', '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()
```



In [31]:

```
print(new_test.shape)
new_test[["datetime", "year", "month", "day", "hour", "minute", "second", "dayofweek"]].head()
```

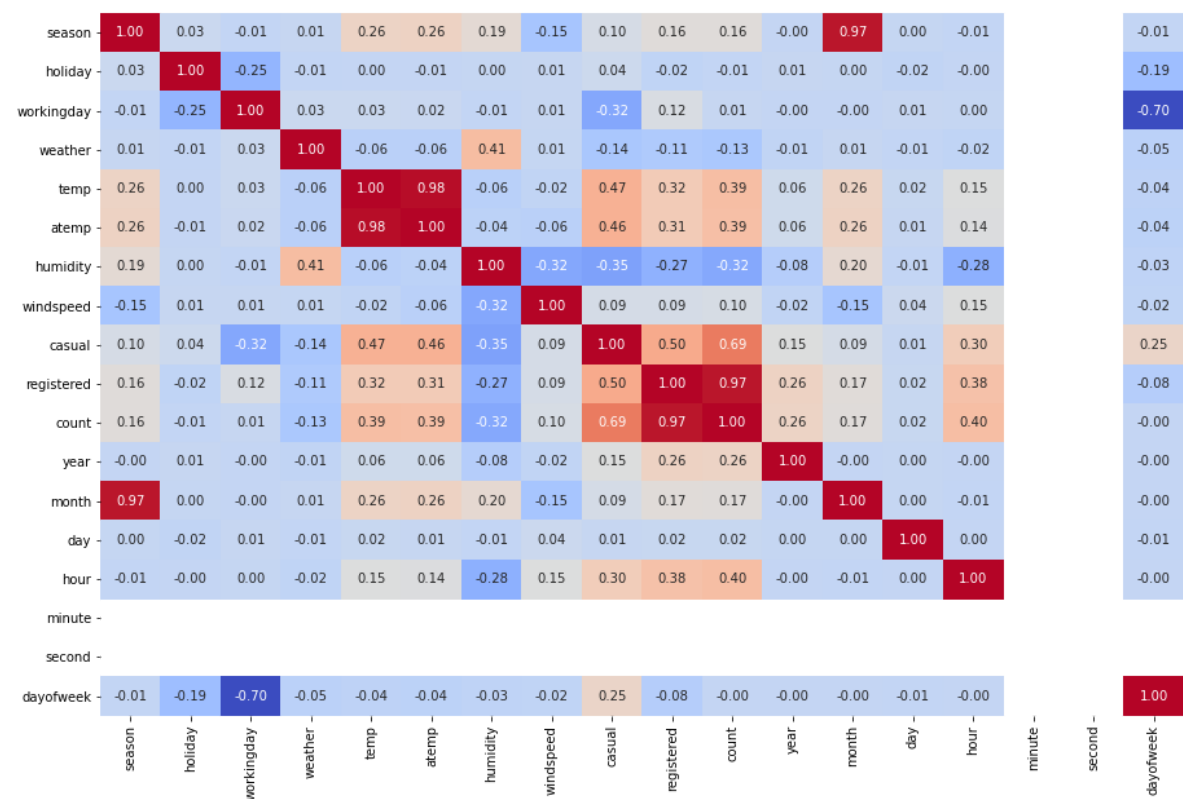
(6493, 16)

Out[31]:

	datetime	year	month	day	hour	minute	second	dayofweek
0	2011-01-20 00:00:00	2011	1	20	0	0	0	3
1	2011-01-20 01:00:00	2011	1	20	1	0	0	3
2	2011-01-20 02:00:00	2011	1	20	2	0	0	3
3	2011-01-20 03:00:00	2011	1	20	3	0	0	3
4	2011-01-20 04:00:00	2011	1	20	4	0	0	3

In [32]:

```
plt.figure(figsize=(15,10))
g = sns.heatmap(new_tr.corr(), annot=True, fmt=".2f", cmap="coolwarm", cbar=False)
```



In [36]:



```
feature_names = [ 'season', 'holiday', 'workingday', 'weather',  
                  'temp', 'atemp', 'humidity', 'windspeed',  
                  "year", "hour", "dayofweek"] # 공통 변수  
  
X_tr_all = new_tr[feature_names] # 학습용 데이터 변수 선택  
X_test_all = new_test[feature_names] # 테스트 데이터의 변수 선택  
print(X_tr_all.head())
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	W
0	1	0	0	1	9.84	14.395	81	0.0	
1	1	0	0	1	9.02	13.635	80	0.0	
2	1	0	0	1	9.02	13.635	80	0.0	
3	1	0	0	1	9.84	14.395	75	0.0	
4	1	0	0	1	9.84	14.395	75	0.0	

	year	hour	dayofweek
0	2011	0	5
1	2011	1	5
2	2011	2	5
3	2011	3	5
4	2011	4	5

In [37]:



```
label_name = 'count' # 렌탈 대수 (종속변수)  
y_tr_all = new_tr[label_name] # 렌탈 대수 변수 값 선택
```

In [38]:



```
from sklearn.model_selection import train_test_split
```

In [39]:



```
X_train, X_test, y_train, y_test = train_test_split(X_tr_all,  
                                                    y_tr_all,  
                                                    test_size=0.3,  
                                                    random_state=77)
```

In [40]:



```
from sklearn.linear_model import LinearRegression  
  
model = LinearRegression() # 모델 객체 생성.  
model.fit(X_train, y_train)  
  
# 정확도 확인  
print("학습용 세트 정확도: {:.3f}".format(model.score(X_train, y_train)))  
print("테스트 세트 정확도: {:.3f}".format(model.score(X_test, y_test)))
```

학습용 세트 정확도: 0.391
테스트 세트 정확도: 0.377

In [41]:



```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor() # 모델 객체 생성.
model.fit(X_train, y_train)

# 정확도 확인
print("학습용 세트 정확도: {:.3f}".format(model.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(model.score(X_test, y_test)))
```

학습용 세트 정확도: 1.000
테스트 세트 정확도: 0.891

In [42]:



```
from sklearn.ensemble import RandomForestRegressor # 앙상블(의사결정트리 확장판)

seed = 37
model = RandomForestRegressor(n_jobs=-1, random_state=seed) # 모델 객체 생성.
model.fit(X_train, y_train) # 모델 학습(공부가 되었다.)

# 정확도 확인
print("학습용 세트 정확도: {:.3f}".format(model.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(model.score(X_test, y_test)))
```

학습용 세트 정확도: 0.992
테스트 세트 정확도: 0.946

In [43]:



```
from sklearn.ensemble import GradientBoostingRegressor

seed = 37
model = GradientBoostingRegressor(random_state=seed) # 모델 객체 생성.
model.fit(X_train, y_train) # 모델 학습(공부가 되었다.)

# 정확도 확인
print("학습용 세트 정확도: {:.3f}".format(model.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(model.score(X_test, y_test)))
```

학습용 세트 정확도: 0.860
테스트 세트 정확도: 0.849

In [45]:

```
sub = pd.read_csv("../bike/sampleSubmission.csv")
sub.head()
```

Out[45]:

	datetime	count
0	2011-01-20 00:00:00	0
1	2011-01-20 01:00:00	0
2	2011-01-20 02:00:00	0
3	2011-01-20 03:00:00	0
4	2011-01-20 04:00:00	0

In [46]:

```
from sklearn.ensemble import RandomForestRegressor # 앙상블(의사결정트리 확장판)

seed = 37
model = RandomForestRegressor(n_jobs=-1, random_state=seed) # 모델 객체 생성.
model.fit(X_train, y_train) # 모델 학습(공부가 되었다.)
```

Out[46]:

RandomForestRegressor(n_jobs=-1, random_state=37)

In [47]:

```
pred = model.predict(X_test_all) # 예측
sub['count'] = pred
sub.loc[sub['count'] < 0, 'count'] = 0
sub.head(3)
```

Out[47]:

	datetime	count
0	2011-01-20 00:00:00	11.75
1	2011-01-20 01:00:00	4.20
2	2011-01-20 02:00:00	4.78

In [48]:

```
# 처음 만는 제출용 csv 파일, 행번호를 없애기
sub.to_csv("third_sub.csv", index=False)
```

score : 0.43006

실습 : 특성을 확대하면 어떨까?

[illegible]

In [53]:



```
from sklearn.ensemble import RandomForestRegressor # 앙상블(의사결정트리 확장판)

seed = 37
model = RandomForestRegressor(n_jobs=-1, random_state=seed) # 모델 객체 생성.
model.fit(X_train, y_train) # 모델 학습(공부가 되었다.)

# 정확도 확인
print("학습용 세트 정확도: {:.3f}".format(model.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(model.score(X_test, y_test)))
```

학습용 세트 정확도: 0.992

테스트 세트 정확도: 0.950

In [54]:



```
nor_X_test_all = scaler.transform(X_test_all)
ex_X_test = PolynomialFeatures(degree=2,
                               include_bias=False).fit_transform(nor_X_test_all)
```

In [55]:



```
pred = model.predict(ex_X_test) # 예측
sub['count'] = pred
sub.loc[sub['count'] < 0, 'count'] = 0
sub.head(3)
```

Out[55]:

	datetime	count
0	2011-01-20 00:00:00	13.66
1	2011-01-20 01:00:00	5.08
2	2011-01-20 02:00:00	3.53

In [56]:



```
# 처음 만는 제출용 csv 파일, 행번호를 없애기
sub.to_csv("four_sub.csv", index=False)
```

score : 0.41328

대표적인 모델 xgboost 사용해 보기

In [57]:



```
import xgboost as xgb
```

xgb.DMatrix

- dense matrix, sparse matrix, local file로부터 DMatrix object 객체를 만든다.

In [58]:

```
# data_dmatrix = xgb.DMatrix(data=ex_X_tr, label=y_tr_all)
```

In [59]:

```
X_train, X_test, y_train, y_test = train_test_split(ex_X_tr,
                                                    y_tr_all,
                                                    test_size=0.2,
                                                    random_state=42)
```

In [60]:

```
# 기본 옵션 확인
xg_reg = xgb.XGBRegressor()
xg_reg
```

Out [60]:

```
XGBRegressor(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
```

- learning_rate: 0~1사이의 값. 과적합을 방지하기 위한 단계 크기
- max_depth: 각각의 나무 **모델의 최대 깊이**
- subsample: **각 나무마다 사용하는 샘플 퍼센트**, 낮은 값은 underfitting(과소적합)을 야기할 수 있음.
- colsample_bytree: 각 나무마다 사용하는 **feature 퍼센트**. High value can lead to overfitting.
- n_estimators: 트리의 수(우리가 모델을 생성할)
- loss function(손실함수)결정.
- objective(목적함수)
 - reg:linear for regression problems(회귀 문제),
 - reg:logistic for classification problems with only decision(분류 문제),
 - binary:logistic for classification problems with probability.
- alpha : L1 규제에 대한 항

In [65]:



```
xg_reg = xgb.XGBRegressor(objective='reg:linear',
                           colsample_bytree = 0.3, # 각나무마다 사용하는 feature 비율
                           learning_rate = 0.1,
                           max_depth = 4,
                           alpha = 0.1,
                           n_estimators = 100) # n_estimators=100

xg_reg
```

Out[65]:

```
XGBRegressor(alpha=0.1, base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=0.3, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints=None,
              learning_rate=0.1, max_delta_step=None, max_depth=4,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              objective='reg:linear', random_state=None, reg_alpha=None,
              reg_lambda=None, scale_pos_weight=None, subsample=None,
              tree_method=None, validate_parameters=None, verbosity=None)
```

학습

In [66]:



```
xg_reg.fit(X_train, y_train) # 모델 학습(공부가 되었다.)
```

[08:20:47] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarererror.

Out[66]:

```
XGBRegressor(alpha=0.1, base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=nan, monotone_constraints=(),
              n_estimators=100, n_jobs=8, num_parallel_tree=1,
              objective='reg:linear', random_state=0, reg_alpha=0.100000001,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [67]:



```
# 정확도 확인
print("학습용 세트 정확도: {:.3f}".format(xg_reg.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(xg_reg.score(X_test, y_test)))
```

학습용 세트 정확도: 0.945
테스트 세트 정확도: 0.933

실습

- 나무의 개수를 조정해 보면서 확인해 보자.

In [68]:



```
%%time

num_list = [100,200,300,500,1000, 1500]

for num in num_list:
    xg_reg = xgb.XGBRegressor(objective='reg:linear',
                              colsample_bytree = 0.3, # 각나무마다 사용하는 feature 비율
                              learning_rate = 0.1,
                              max_depth = 3,
                              alpha = 0.1,
                              n_estimators = num)
    xg_reg.fit(X_train, y_train)
    print("학습용 세트 정확도: {:.3f}".format(xg_reg.score(X_train, y_train)))
    print("테스트 세트 정확도: {:.3f}".format(xg_reg.score(X_test, y_test)))
```

[08:20:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarrederror.

학습용 세트 정확도: 0.912

테스트 세트 정확도: 0.901

[08:20:51] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarrederror.

학습용 세트 정확도: 0.940

테스트 세트 정확도: 0.928

[08:20:52] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarrederror.

학습용 세트 정확도: 0.950

테스트 세트 정확도: 0.936

[08:20:53] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarrederror.

학습용 세트 정확도: 0.961

테스트 세트 정확도: 0.943

[08:20:55] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarrederror.

학습용 세트 정확도: 0.973

테스트 세트 정확도: 0.950

[08:20:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarrederror.

학습용 세트 정확도: 0.980

테스트 세트 정확도: 0.952

Wall time: 15.2 s

최종 모델

In [69]:

```
xg_reg = xgb.XGBRegressor(objective='reg:linear',
                           colsample_bytree = 0.3, # 각나무마다 사용하는 feature 비율
                           learning_rate = 0.05,
                           max_depth = 5,
                           alpha = 0.1,
                           n_estimators = 500)

xg_reg.fit(X_train, y_train)
print("학습용 세트 정확도: {:.3f}".format(xg_reg.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(xg_reg.score(X_test, y_test)))
```

[08:21:06] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarederror.

학습용 세트 정확도: 0.979

테스트 세트 정확도: 0.956

In [70]:

```
nor_X_test_all = scaler.transform(X_test_all)
ex_X_test = PolynomialFeatures(degree=2,
                               include_bias=False).fit_transform(nor_X_test_all)
```

In [71]:

```
pred = xg_reg.predict(ex_X_test) # 예측
sub['count'] = pred
sub.loc[sub['count'] < 0, 'count'] = 0
sub.head(3)
```

Out[71]:

	datetime	count
0	2011-01-20 00:00:00	12.755984
1	2011-01-20 01:00:00	2.368304
2	2011-01-20 02:00:00	1.013963

In [72]:

```
# 처음 만는 제출용 csv 파일, 행번호를 없애기
sub.to_csv("five_xgb_sub.csv", index=False)
```

(실습) lr_rate와 max_depth를 조정해 가면서 확인해 보자.

score : 0.65533

- lr_rate=0.1, max_depth = 4, 1000개 트리

score : 0.70240

- lr_rate=0.1, max_depth = 3, 1500개 트리

score : 0.75037

- lr_rate = 0.001, max_depth = 3, 1000개 트리

score : 0.57720

- lr_rate = 0.05, max_depth = 5, 500개 트리

In []:

