

Kaggle 대회

In [1]:

```
from IPython.display import display, Image
```



Data Fields

- datetime - hourly date + timestamp (시간 데이터 + timestamp)
- season - 1 = spring, 2 = summer, 3 = fall, 4 = winter (봄[1], 여름[2], 가을[3], 겨울[4])
- holiday - whether the day is considered a holiday (휴일인지 아닌지)
- workingday - whether the day is neither a weekend nor holiday(일하는 날인지 아닌지)
- weather - 날씨에 따른 값
 - 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 Pellets + Thunderstorm + Mist, Snow + Fog
- temp - temperature in Celsius (온도)
- atemp - "feels like" temperature in Celsius (체감온도)
- humidity - relative humidity (습도)
- windspeed - wind speed (바람 속도)
- casual - number of non-registered user rentals initiated (등록되지 않은 사용자의 빌린 자전거 대수)
- registered - number of registered user rentals initiated (등록된 사용자의 빌린 자전거 대수)
- count - number of total rentals (자전거 빌린 총 대수 - 시간별)

In [2]:

```
import pandas as pd
```



1-1 데이터 준비하기

In [3]:

```
## train 데이터 셋 , test 데이터 셋  
## train 은 학습을 위한 데이터 셋  
## test 은 예측을 위한 데이터 셋(평가)  
## parse_dates : datetime 컬럼을 시간형으로 불러올 수 있음  
train = pd.read_csv("bike/train.csv", parse_dates=['datetime'])  
test = pd.read_csv("bike/test.csv", parse_dates=['datetime'])
```



In [4]:

```
print(train.shape) # : 행과 열 갯수 확인  
print(test.shape)
```



(10886, 12)

(6493, 9)

In [5]:



```
train.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 [6]:



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

1-2 데이터 탐색해 보기 - 시각화

- 범주형 데이터 : 'season', 'holiday', 'workingday', 'weather'

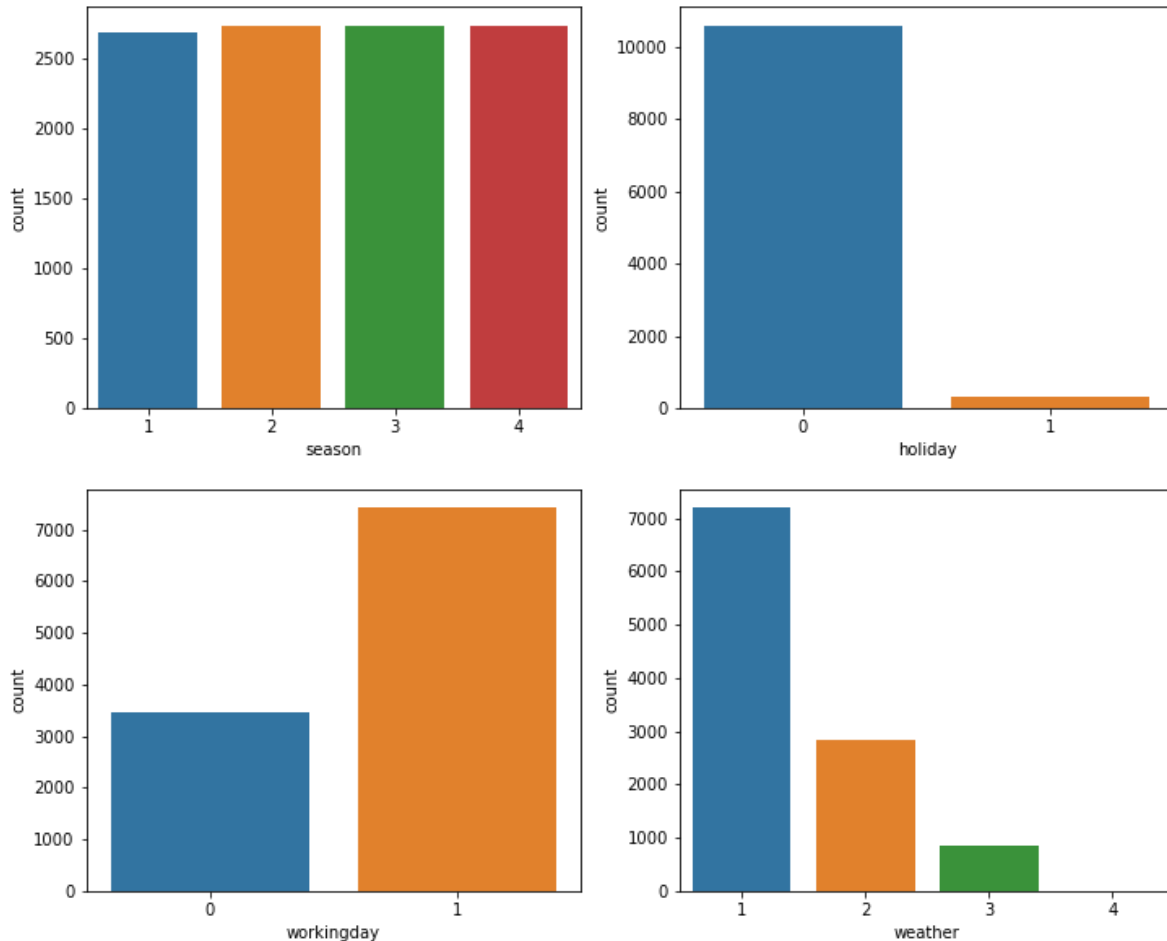
In [7]:



```
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.
```



수치형 데이터 선택

In [8]:



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

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

Out[8]:

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

In [9]:

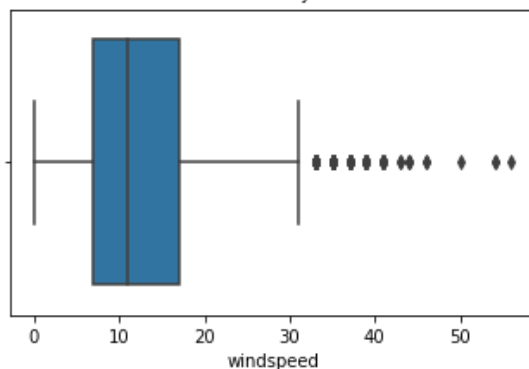
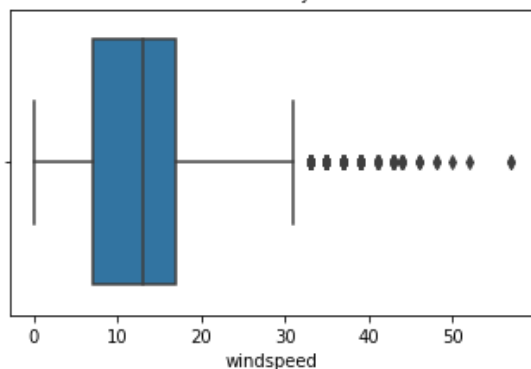
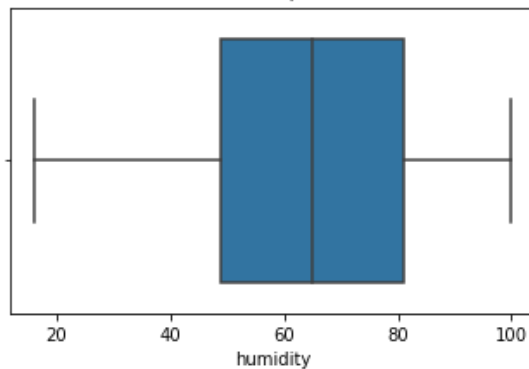
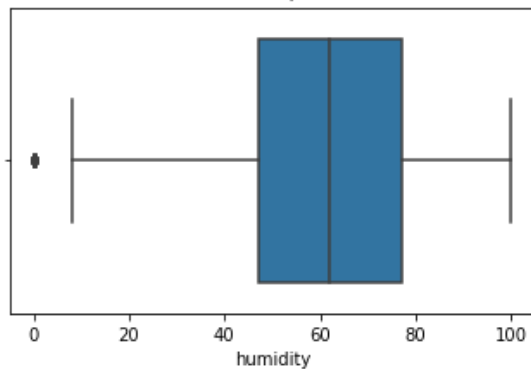
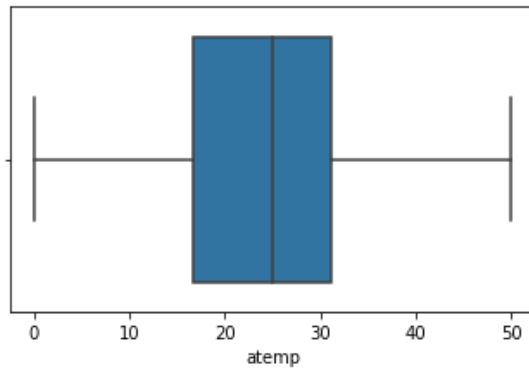
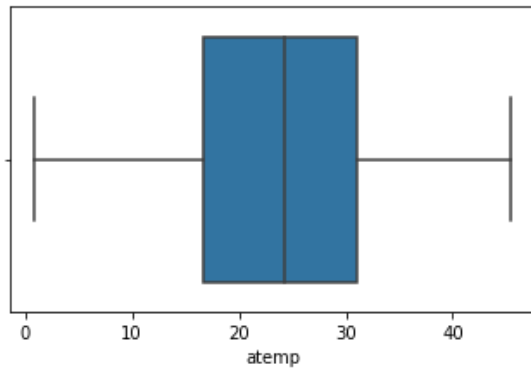
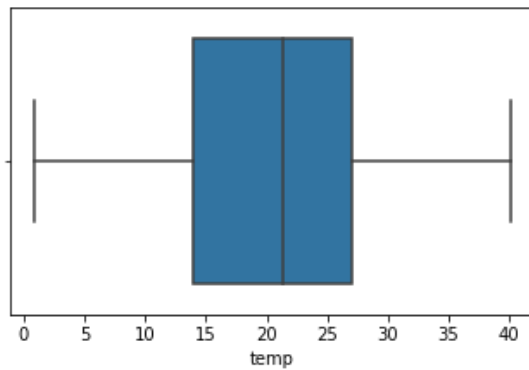
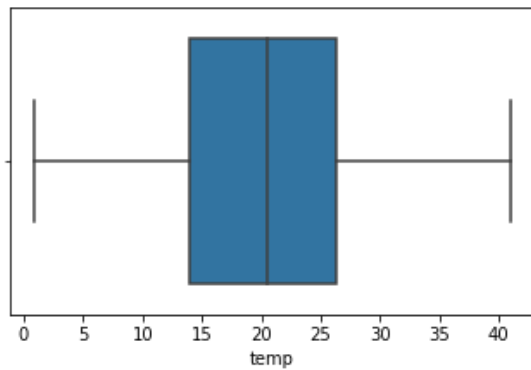


```
# par(mfrow=c(2,2)) -> R

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(name, data=train) # i번째 그래프에 sns.countplot를 그리겠다.
    plt.subplot(4,2,i*2)     # 2행 2열에 i번째 그래프 선택
    sns.boxplot(name, data=test) # i번째 그래프에 sns.countplot를 그리겠다.

plt.show()
```



In [10]:



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

Out[10]:

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

1-3 파생변수(더미변수) 생성

In [11]:



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

Out[11]:

	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 [12]:



```
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[12]:

	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 [13]:

```
train.columns
```

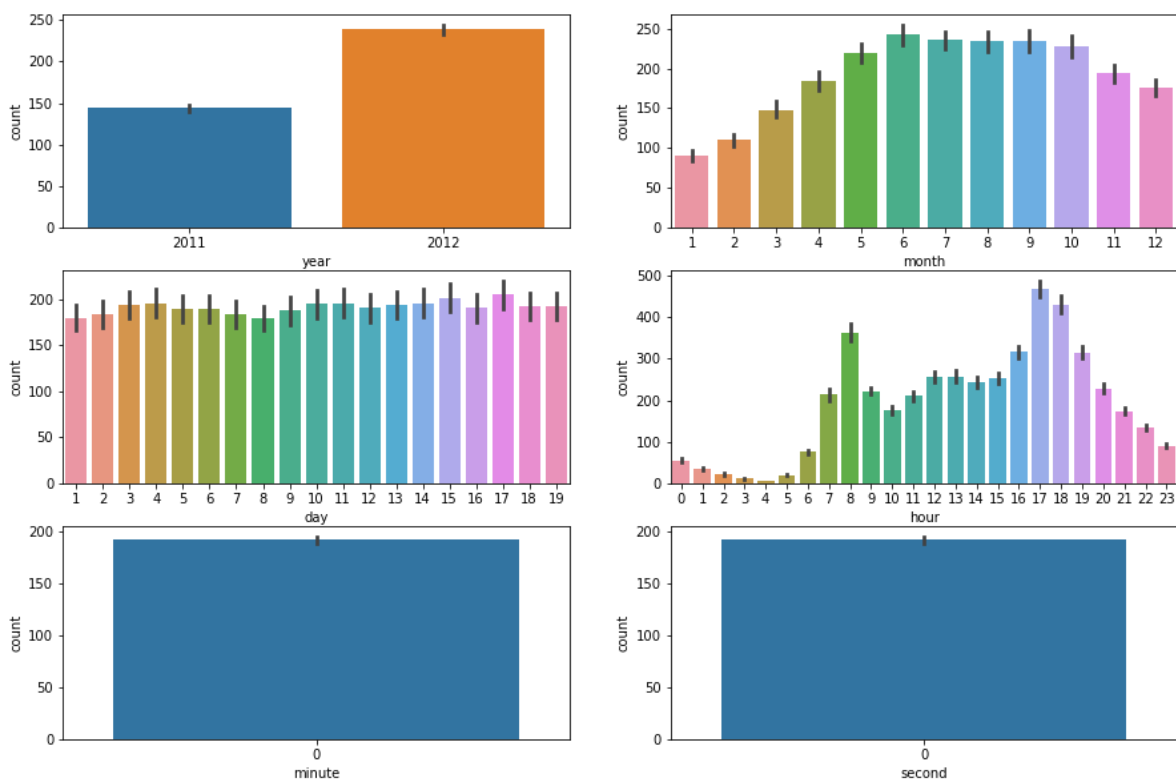
Out[13]:

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

In [14]:

```
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.
- day 1~19 일... 20일이 없네요.(test)

In [15]:



```
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 [16]:



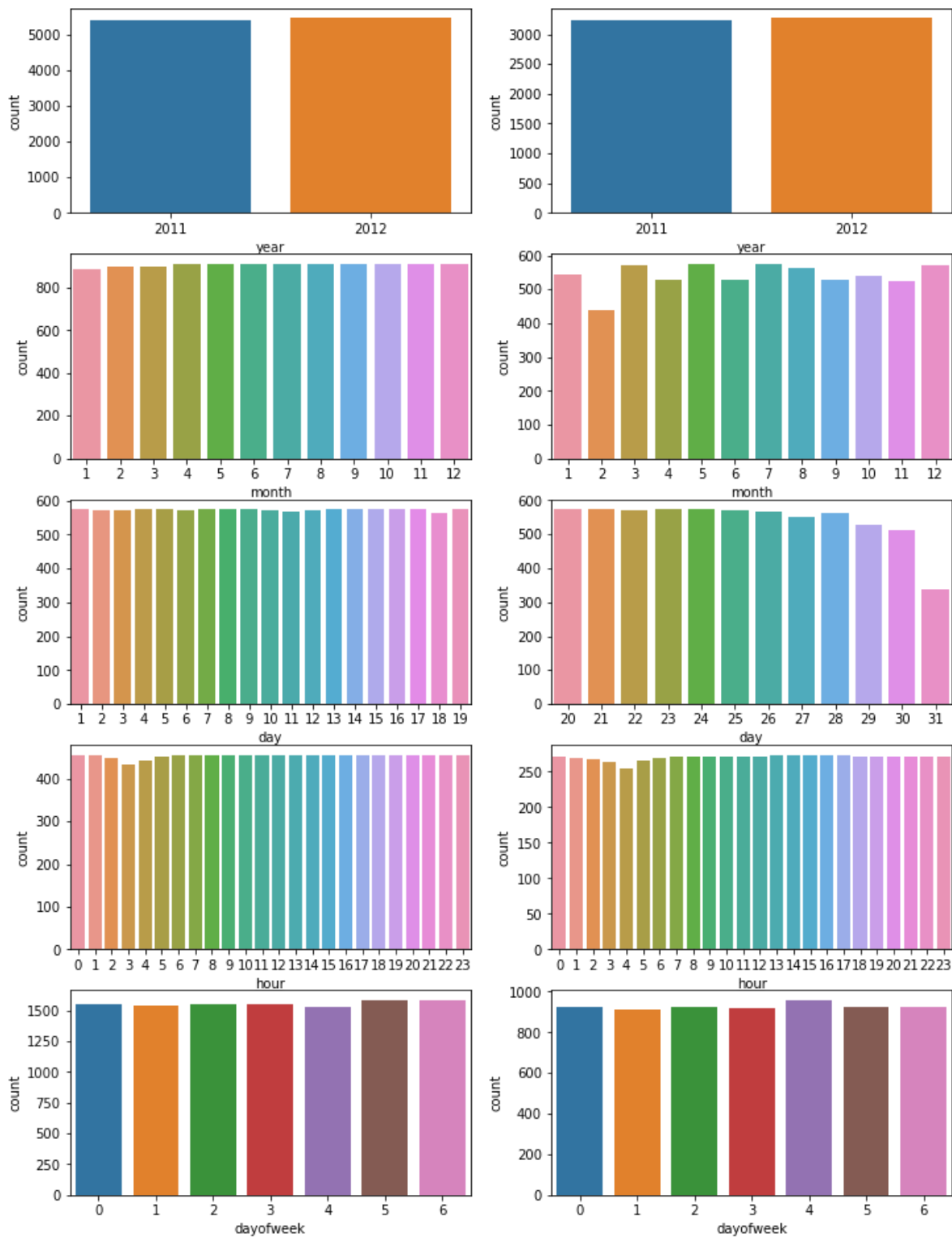
```
col_names = ['year', 'month', 'day', 'hour', 'dayofweek']
i = 0

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(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(name, data = new_test)

plt.show()
```



변수 생성

In [17]:

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

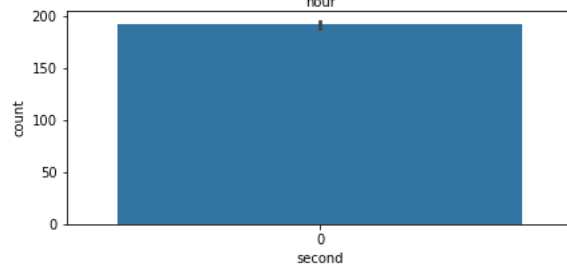
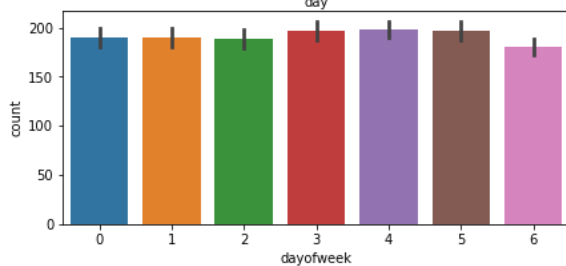
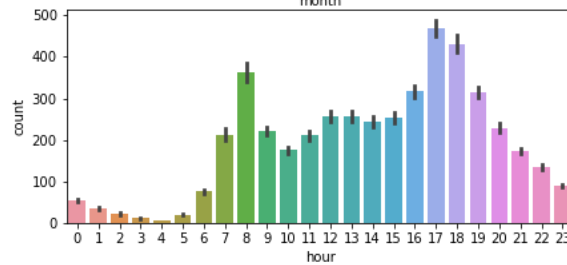
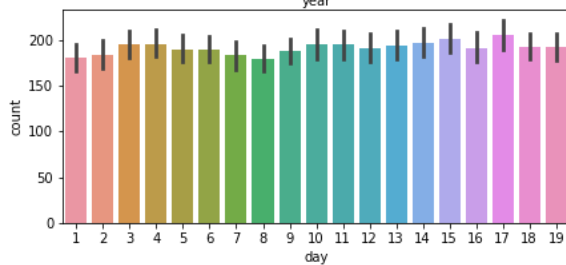
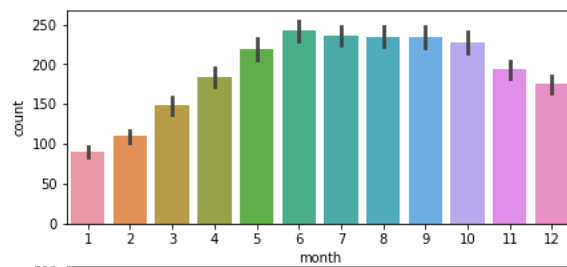
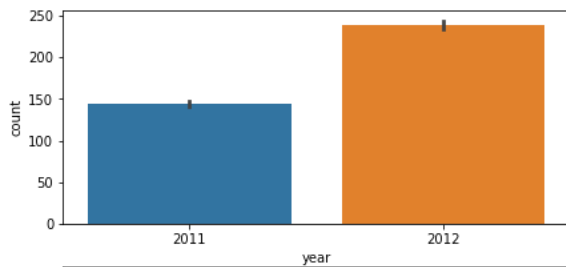
In [18]:



```
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 [19]:

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

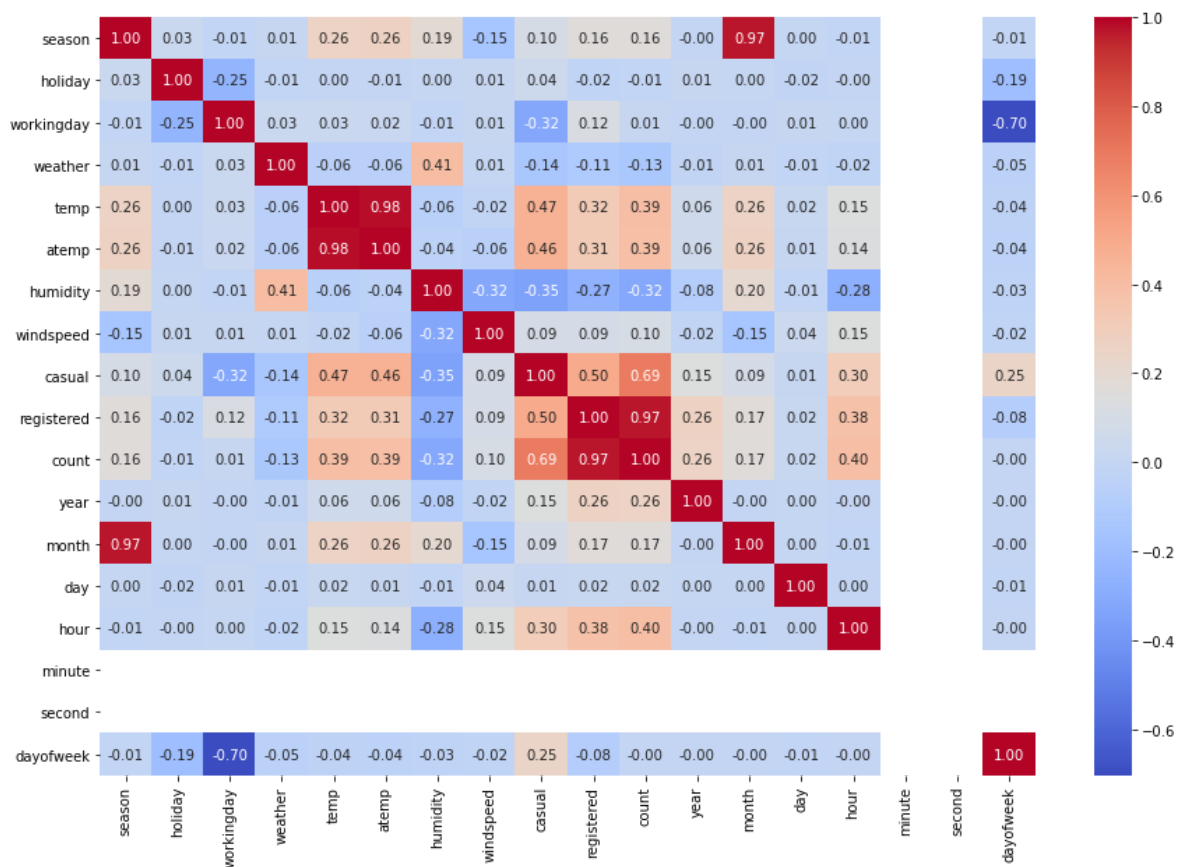
(6493, 16)

Out[19]:

	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 [20]:

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



In [21]:

```
print(new_tr.columns)
print(new_test.columns)
```

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

In [22]:

```
feature_names = [ 'season', 'holiday', 'workingday', 'weather',
                  'temp', 'atemp', 'humidity', 'windspeed',
                  "year", "hour", "dayofweek"] # 공통 변수

X_train = new_tr[feature_names] # 학습용 데이터 변수 선택
X_test = new_test[feature_names] # 테스트 데이터의 변수 선택
print(X_train.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 [23]:

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

1-4 모델 만들기 및 제출

모델 만들기 및 예측 순서

- 모델을 생성한다. model = 모델명()
- 모델을 학습한다. model.fit(입력값, 출력값)
- 모델을 이용하여 예측 model.predict(입력값)

In [24]:



```
from sklearn.linear_model import LinearRegression # 선형회귀
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test) # 예측(새로운 데이터로)
predictions
```

Out[24]:

```
array([-23.27179232, -20.84936197, -13.04580719, ..., 209.84495832,
       227.95174821, 217.86201958])
```

In [25]:



```
sub = pd.read_csv("bike/sampleSubmission.csv")
sub['count'] = predictions
sub.head()
```

Out[25]:

	datetime	count
0	2011-01-20 00:00:00	-23.271792
1	2011-01-20 01:00:00	-20.849362
2	2011-01-20 02:00:00	-13.045807
3	2011-01-20 03:00:00	-1.986454
4	2011-01-20 04:00:00	5.817101

음수에 대한 값처리

- count가 0이하의 경우에 대해서 'count'를 0으로 한다.

In [26]:



```
sub.loc[ sub['count'] < 0, 'count' ] = 0
sub.loc[ sub['count'] < 0, : ]
```

Out[26]:

	datetime	count
--	----------	-------

In [27]:



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

의사 결정 트리 모델 만들기

In [28]:

```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor() # 모델 객체 생성.
model.fit(X_train, y_train)
pred_2 = model.predict(X_test) # 예측(새로운 데이터로)
pred_2
```

Out[28]:

```
array([12.,  4.,  3., ..., 78., 78., 41.])
```

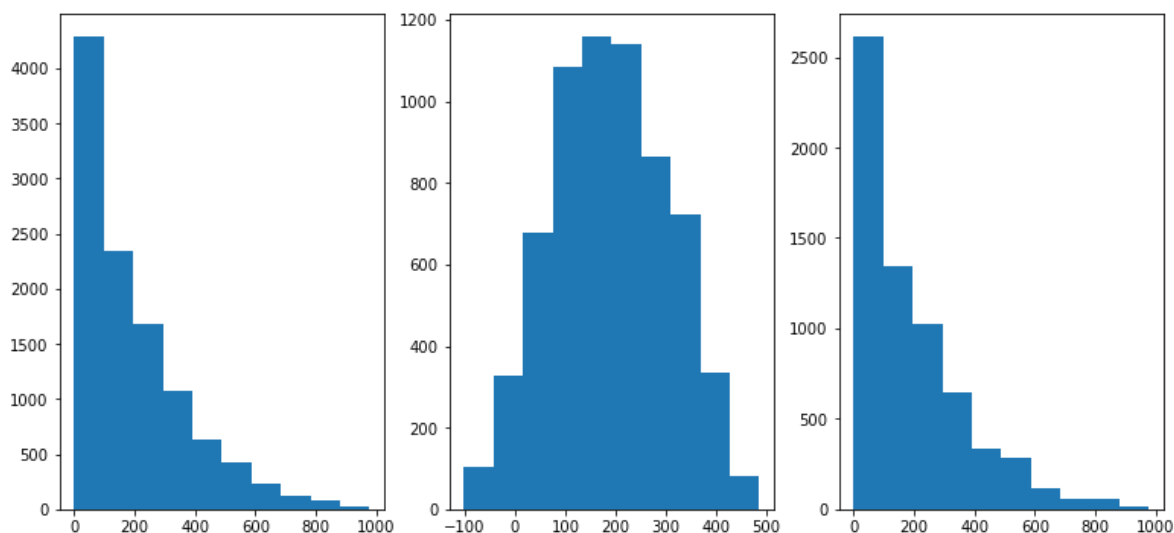
In []:

In [29]:

```
plt.figure(figsize=(13,6))
plt.subplot(1,3,1)
plt.hist(new_tr['count']) # 학습용 데이터 자전거 렌탈 대수
plt.subplot(1,3,2)
plt.hist(predictions) # 선형 회귀로 테스트 데이터 이용 예측한 대수
plt.subplot(1,3,3)
plt.hist(pred_2) # 의사결정트리로 테스트 데이터 이용 예측한 대수
```

Out[29]:

```
(array([2613., 1342., 1021.,  643.,  337.,  287.,  116.,  61.,  55.,
        18.]),
 array([ 1.,  98.6, 196.2, 293.8, 391.4, 489.,  586.6, 684.2, 781.8,
        879.4, 977. ]),
 <a list of 10 Patch objects>)
```



In [30]:

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


In [31]:



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

Out[31]:

```
array([ 11.52      ,  4.58      ,  3.76      , ..., 102.1      ,
        99.91333333, 46.9      ])
```

In [32]:



```
sub = pd.read_csv("bike/sampleSubmission.csv")
sub['count'] = predictions
```

In [33]:



```
sub.head()
```

Out[33]:

	datetime	count
0	2011-01-20 00:00:00	11.52
1	2011-01-20 01:00:00	4.58
2	2011-01-20 02:00:00	3.76
3	2011-01-20 03:00:00	3.61
4	2011-01-20 04:00:00	2.97

In [34]:



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

그렇다면 어떤 모델이 나은지 어떻게 판단할 수 있는가?

1-5 모델 평가 및 제출

- 데이터 나누는 방법으로 기본으로 `train_test_split` 함수가 있음.
- 교차검증 반복 함수 `cross_val_score`
 - `cross-validation`에 의해 점수를 평가한다.
- `cross_val_score(model, X, y, scoring=None, cv=None)`

model : 회귀 분석 모형

X : 독립 변수 데이터

y : 종속 변수 데이터

scoring : 성능 검증에 사용할 함수 이름

cv : 교차검증 생성기 객체 또는 숫자.

None이면 `KFold(3)`, 숫자 `k`이면 `KFold(k)`

In [35]:

```
from sklearn.model_selection import cross_val_score
```

In [36]:

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
```

In [37]:

```
import numpy as np
```

In [38]:

```
model_list = ["LinearRegression", "DecisionTreeRegressor", "KNeighborsRegressor", "RandomForestRegressor"]
model_score = []
```

선형회귀 01

In [39]:

```
model = LinearRegression()
model.fit(X_train, y_train)
score = cross_val_score(model, X_train, y_train,
                        cv=5, scoring="neg_mean_squared_error")

print(score)
print("MSE 평균 :", score.mean())
m_score = np.abs(score.mean()) # 절대값
model_score.append(m_score)
```

```
[-10001.73269892 -14719.1205855 -13684.42876335 -33057.3894553
 -33971.64722717]
MSE 평균 : -21086.863746046256
```

의사결정트리 decision tree, knn

In [40]:



```
model = DecisionTreeRegressor()
model.fit(X_train, y_train)
score = cross_val_score(model, X_train, y_train,
                        cv=5, scoring="neg_mean_squared_error")
print(score)
print("MSE 평균 :", score.mean())
m_score = np.abs(score.mean()) # 절대값
print(m_score)
model_score.append(m_score)
```

```
[-8410.10743802 -5497.64354616 -8485.19074414 -8419.71061093
 -8498.58187873]
MSE 평균 : -7862.2468435977935
7862.2468435977935
```

In [41]:



```
model = KNeighborsRegressor()
model.fit(X_train, y_train)
score = cross_val_score(model, X_train, y_train,
                        cv=5, scoring="neg_mean_squared_error")
print(score)
print("MSE 평균 :", score.mean())
m_score = np.abs(score.mean()) # 절대값
model_score.append(m_score)
```

```
[-15489.78091827 -19509.55790537 -12562.31961415 -24299.72957281
 -28798.93245751]
MSE 평균 : -20132.064093622575
```

앙상블 RandomForest, Ada

In [42]:



```
model = RandomForestRegressor()
model.fit(X_train, y_train)
score = cross_val_score(model, X_train, y_train,
                        cv=5, scoring="neg_mean_squared_error")
print(score)
print("MSE 평균 :", score.mean())
m_score = np.abs(score.mean()) # 절대값
model_score.append(m_score)
```

```
[-7009.03576673 -2904.03346244 -6353.86903426 -4518.07150772
 -5776.63486115]
MSE 평균 : -5312.328926461666
```

In [43]:

```
model = AdaBoostRegressor()
model.fit(X_train, y_train)
score = cross_val_score(model, X_train, y_train,
                        cv=5, scoring="neg_mean_squared_error")
print(score)
print("MSE 평균 :", score.mean())
m_score = np.abs(score.mean()) # 절대값
model_score.append(m_score)
```

```
[-21254.40209591 -10436.22572022 -16275.49097409 -13543.60351299
 -16619.86773596]
MSE 평균 : -15625.91800783496
```

In [44]:

```
import pandas as pd
dat = pd.DataFrame( {'model_name':model_list, 'score': model_score })
dat
```

Out[44]:

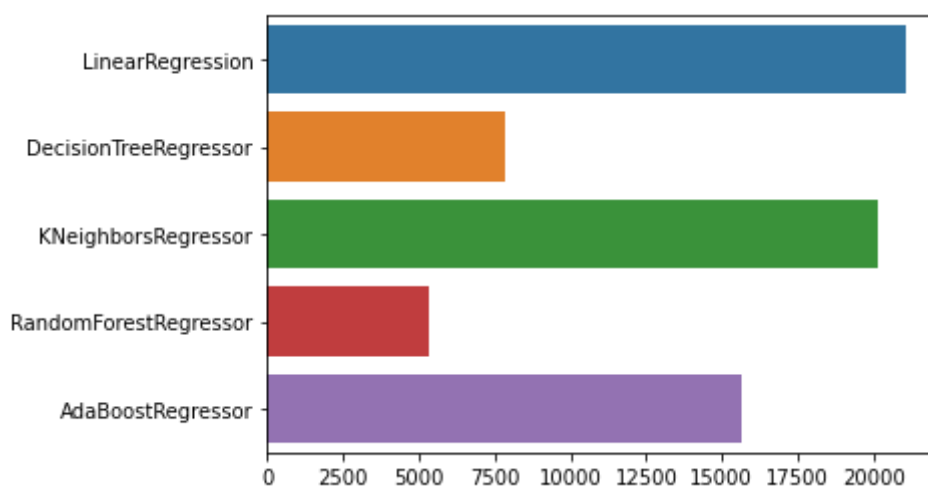
	model_name	score
0	LinearRegression	21086.863746
1	DecisionTreeRegressor	7862.246844
2	KNeighborsRegressor	20132.064094
3	RandomForestRegressor	5312.328926
4	AdaBoostRegressor	15625.918008

In [45]:

```
# 모델과 스코어 확인하기
sns.barplot(x=model_score , y=model_list, data=dat)
```

Out[45]:

<matplotlib.axes._subplots.AxesSubplot at 0x1f772127280>



가장 오차가 적은 RandomForestRegressor 모델로 예측해 보기

In [46]:



```
# 최종 모델 선택 및 제출
model_RF = RandomForestRegressor()
model_RF.fit(X_train, y_train)
pred = model_RF.predict(X_test)
pred[0:10]
```

Out[46]:

```
array([ 11.62,   4.78,   3.89,   3.59,   2.83,   6.53,  37.72, 105.86,
        242.14, 138.46])
```

In [47]:



```
pd.DataFrame( {"pred":pred})
```

Out[47]:

	pred
0	11.620000
1	4.780000
2	3.890000
3	3.590000
4	2.830000
...	...
6488	201.770000
6489	162.640000
6490	100.350000
6491	97.981667
6492	46.270000

6493 rows × 1 columns

머신러닝 대표적 앙상블 중의 하나 XGBOOSTING 기법 사용해보기

In [1]:



```
import xgboost as xgb
```

xgb.DMatrix

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

In []:



```
data_dmatrix = xgb.DMatrix(data=X_train,label=y_train)
```

In []:



```
# 기본 옵션 확인
```

```
xg_reg = xgb.XGBRegressor()  
xg_reg
```

- 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 []:



```
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 = 100) # n_estimators=100  
xg_reg
```

학습

In []:



```
xg_reg.fit(X_train,y_train)
```

In []:



```
score = cross_val_score(xg_reg, X_train, y_train,  
                        cv=5, scoring="neg_mean_squared_error")  
print(score)  
print("MSE 평균 :", score.mean())
```

In []:



```
# 최종 모델 선택 및 제출  
model_RF = RandomForestRegressor()  
model_RF.fit(X_train, y_train)  
predictions = model_RF.predict(X_test)  
sub = pd.read_csv("bike/sampleSubmission.csv")  
sub['count'] = predictions  
sub.to_csv("submission1907.csv", index=False)
```

REF

- cross_val_score:

- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)
- 모델 평가 scoring : https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html)
- XGBOOST Documentation : <https://xgboost.readthedocs.io/en/latest/index.html>
(<https://xgboost.readthedocs.io/en/latest/index.html>)

History

- 2020-06-08 update
- 2020-09-05 update v12