Bike Sharing Demand

- 참고 : 캐글 사이트

1. Loading Library

```
import pandas as pd
import numpy as np
from scipy import stats
```

```
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.style.use('ggplot')
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

2. Loading Data

```
train = pd.read_csv('C:\\ai\\workspace\\data\\bike\\train.csv')
train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1

변수설명

- 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
- atemp - "feels like" temperature in Celsius
- humidity - relative humidity
- windspeed - wind speed

- casual - number of non-registered user rentals initiated
- registered - number of registered user rentals initiated
- count - number of total rentals (Dependent Variable)

```
test = pd.read_csv('C:\\ai\\workspace\\data\\bike\\test.csv')
test.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.0014
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.0014

```
train = train.drop(['casual', 'registered'], axis=1)
train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	1

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 10 columns):
datetime      10886 non-null object
season        10886 non-null int64
holiday       10886 non-null int64
workingday    10886 non-null int64
weather       10886 non-null int64
temp          10886 non-null float64
atemp         10886 non-null float64
humidity      10886 non-null int64
windspeed     10886 non-null float64
count         10886 non-null int64
dtypes: float64(3), int64(6), object(1)
memory usage: 850.6+ KB
```

```
train['datetime'] = pd.to_datetime(train['datetime'])
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
```

```
Data columns (total 10 columns):
datetime      10886 non-null datetime64[ns]
season        10886 non-null int64
holiday       10886 non-null int64
workingday    10886 non-null int64
weather       10886 non-null int64
temp          10886 non-null float64
atemp         10886 non-null float64
humidity      10886 non-null int64
windspeed     10886 non-null float64
count         10886 non-null int64
dtypes: datetime64[ns](1), float64(3), int64(6)
memory usage: 850.6 KB
```

```
datetime = train['datetime']
datetime.head()
```

```
0   2011-01-01 00:00:00
1   2011-01-01 01:00:00
2   2011-01-01 02:00:00
3   2011-01-01 03:00:00
4   2011-01-01 04:00:00
Name: datetime, dtype: datetime64[ns]
```

```
train['year'] = datetime.dt.year
train['year'].head()
```

```
0    2011
1    2011
2    2011
3    2011
4    2011
Name: year, dtype: int64
```

```
train['month'] = datetime.dt.month
train['month'].head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: month, dtype: int64
```

```
train['day'] = datetime.dt.day
train['day'].head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: day, dtype: int64
```

```
train['hour'] = datetime.dt.hour
train['hour'].head()
```

```
0    0
1    1
2    2
3    3
4    4
Name: hour, dtype: int64
```

```
train['minute'] = datetime.dt.minute
train['minute'].head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: minute, dtype: int64
```

```
train['second'] = datetime.dt.second
train['second'].head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: second, dtype: int64
```

```
train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	year	mor
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	16	2011	1
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	40	2011	1
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	32	2011	1
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	13	2011	1
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	1	2011	1

```
train['workingday'].value_counts()
```

```
1    7412
0    3474
Name: workingday, dtype: int64
```

```
train['minute'].value_counts()
```

```
0    10886
Name: minute, dtype: int64
```

```
train['second'].value_counts()
```

```
0    10886
Name: second, dtype: int64
```

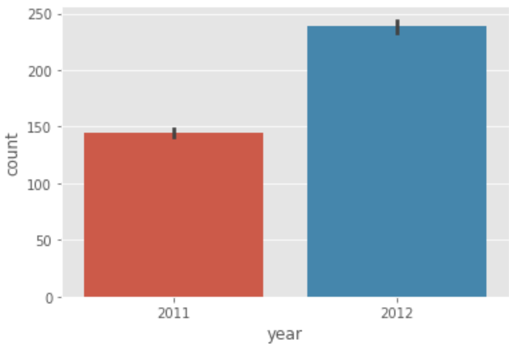
```
train = train.drop(['minute', 'second'], axis=1)
train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	year	mor
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	16	2011	1
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	40	2011	1
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	32	2011	1
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	13	2011	1
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	1	2011	1

3. EDA

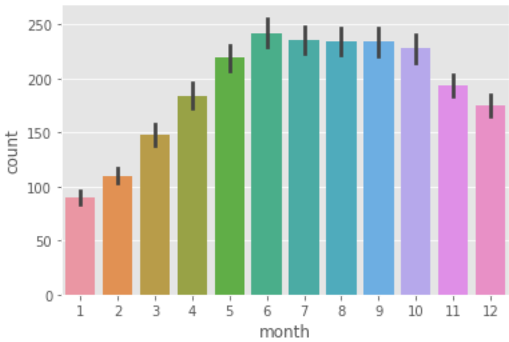
```
# 연도
sns.barplot(data=train, x='year', y='count')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1ba8d7c48>
```




```
# 월
sns.barplot(data=train, x='month', y='count')
```

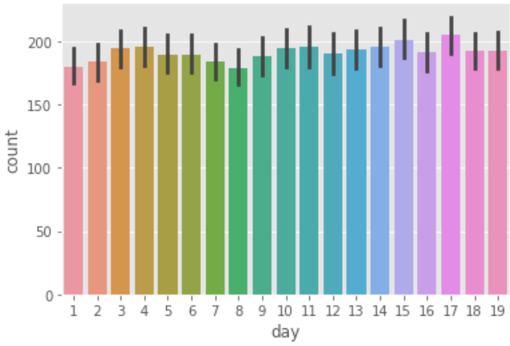
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1b9e8a908>
```



- 5월부터 10월까지 높은 이용률을 보이고, 날씨가 상대적으로 추운 11월~3월 사이에는 낮은 이용률을 보인다.

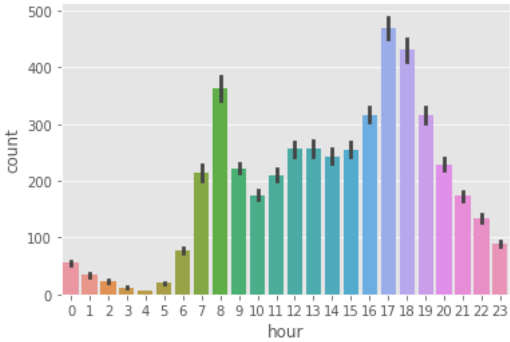
```
# 일
sns.barplot(data=train, x='day', y='count')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1ba8a90c8>
```




```python
# 시간
sns.barplot(data=train, x='hour', y='count')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1b1e70c48>
```



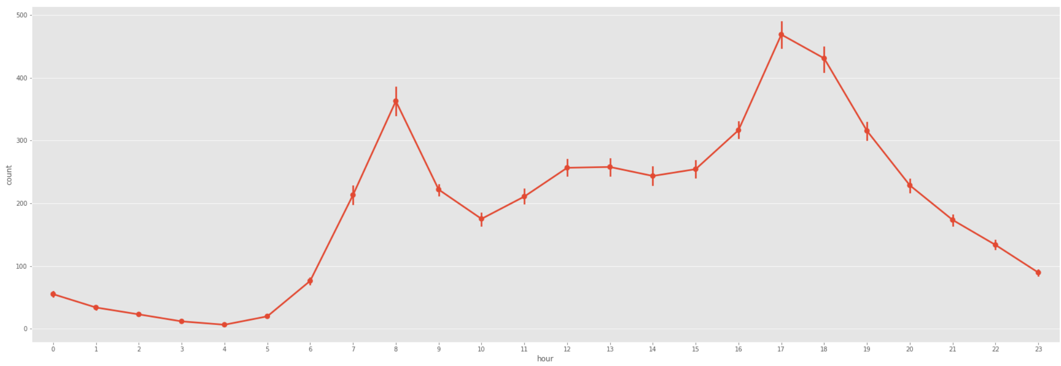
- 출근시간과 퇴근시간에 이용 횟수가 높은 것을 확인할 수 있다.

```python
train['dayofweek'] = datetime.dt.dayofweek
train['dayofweek'].head()
```

```
0    5
1    5
2    5
3    5
4    5
Name: dayofweek, dtype: int64
```

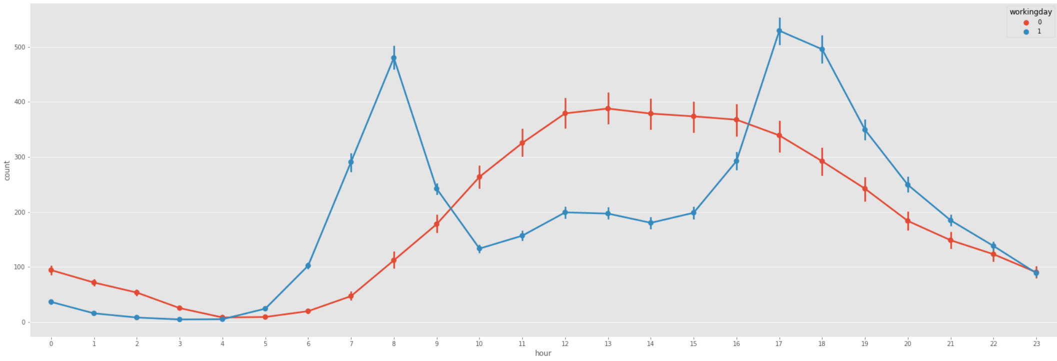
```python
# 시간
plt.figure(figsize=(30,10))
sns.pointplot(data=train, x='hour', y='count')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1c3ddb888>
```



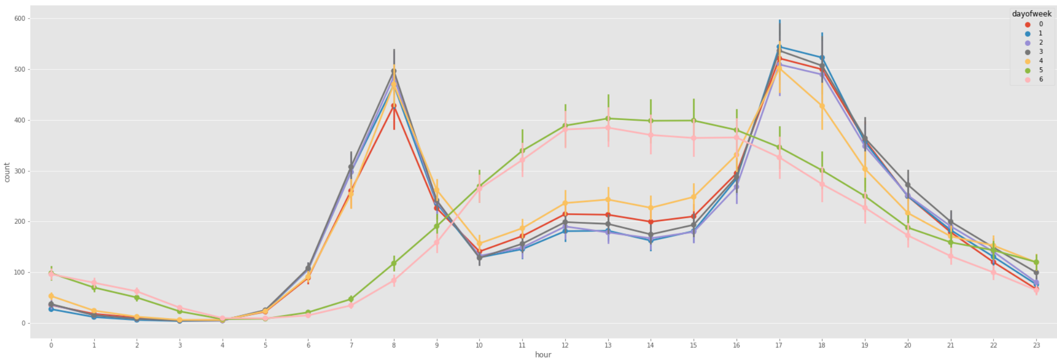

```python
# 시간 & 주말여부
plt.figure(figsize=(30,10))
sns.pointplot(data=train, x='hour', y='count', hue='workingday')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1c3d7db48>
```



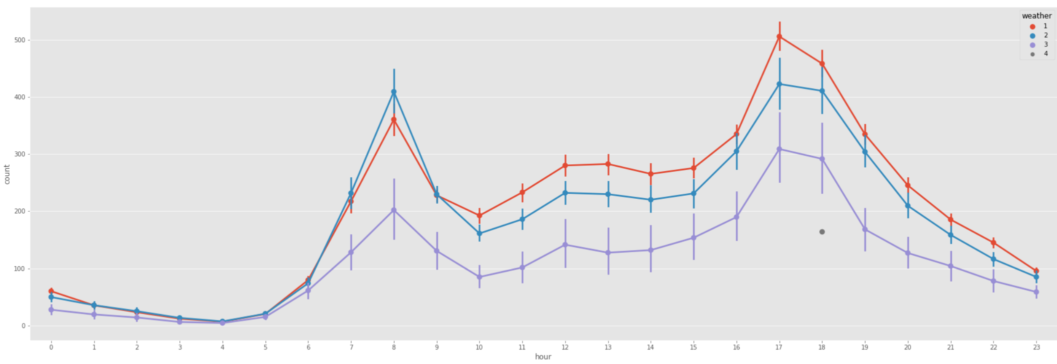

```python
# 시간 & 요일
plt.figure(figsize=(30,10))
sns.pointplot(data=train, x='hour', y='count', hue = 'dayofweek')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1c3e9ae08>
```




```python
# 시간 & 날씨
plt.figure(figsize=(30,10))
sns.pointplot(data=train, x='hour', y='count', hue = 'weather')
```

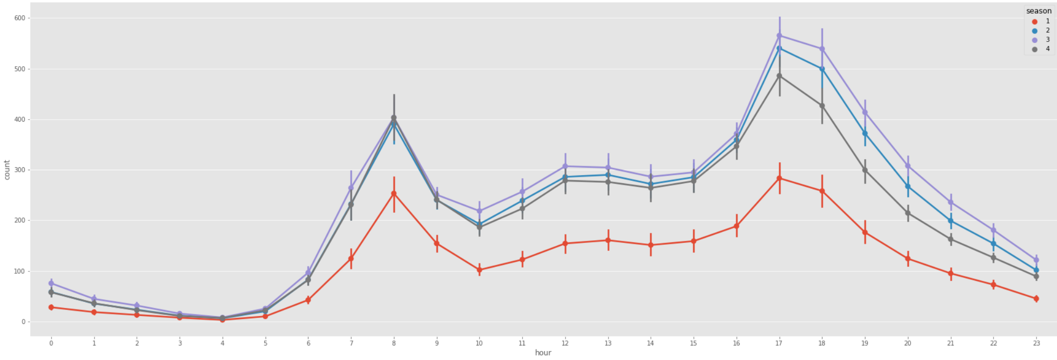
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1bce16a08>
```



- 4번 날씨(기상 악화)일 때 이용이 거의 저조한 것을 확인할 수 있다.

```python
# 시간 & 계절
plt.figure(figsize=(30,10))
sns.pointplot(data=train, x='hour', y='count', hue = 'season')
```

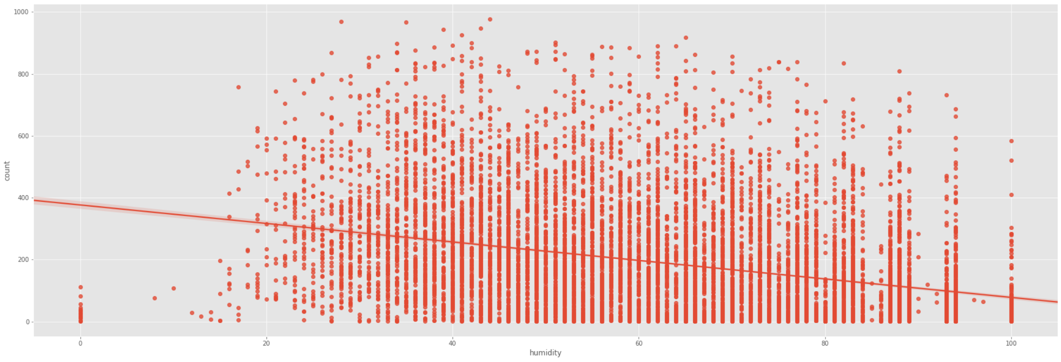
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1b2e37c88>
```



- 상대적으로 봄에 이용이 적은 것을 확인할 수 있다.

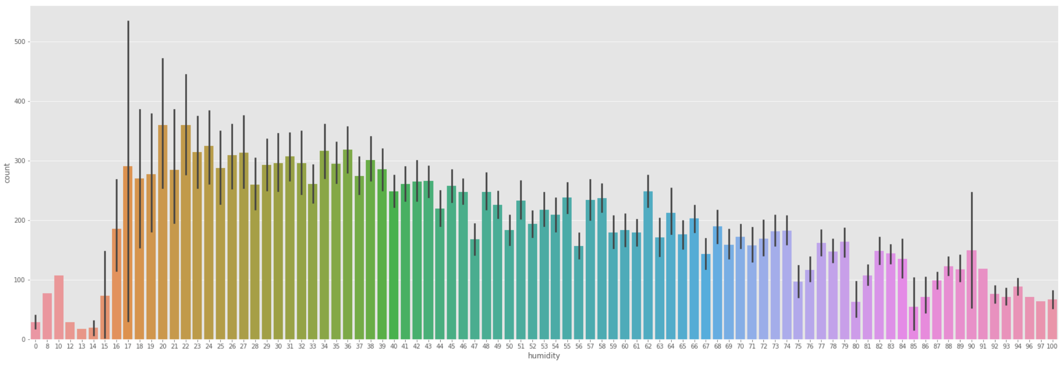
```
# 습도
plt.figure(figsize=(30,10))
sns.regplot(data=train, x='humidity', y='count')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1c35b06c8>
```



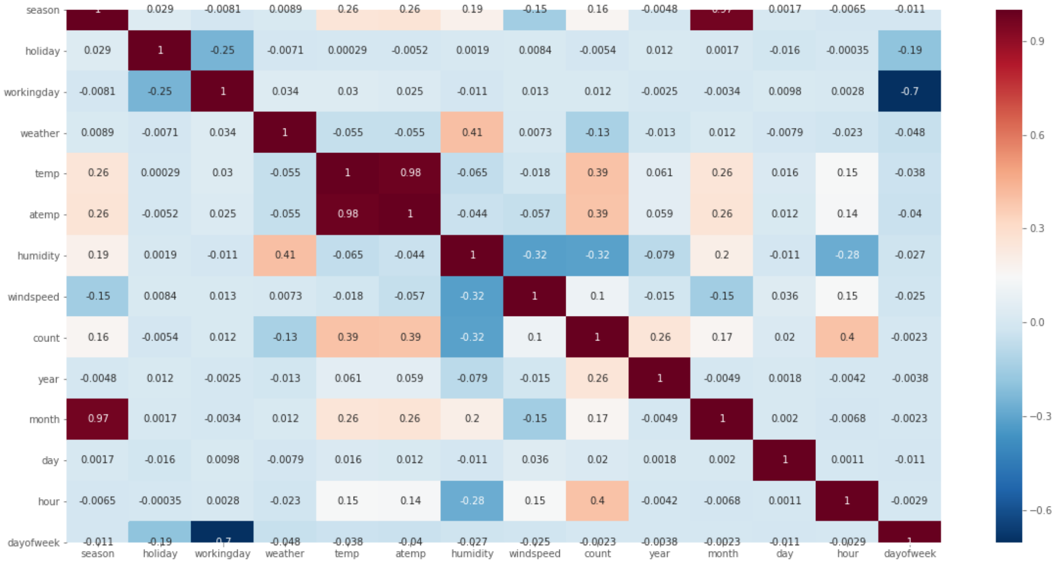

```
# 습도
plt.figure(figsize=(30,10))
sns.barplot(data=train, x='humidity', y='count')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1c2a786c8>
```




```
# 변수간 상관관계
fig, ax = plt.subplots()
fig.set_size_inches(20,10)
sns.heatmap(train.corr(), annot = True, cmap='RdBu_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1cafd6248>
```




```
train.corr()['count'].apply(lambda x : np.abs(x)).sort_values(ascending=False)
```

```
count         1.000000
hour          0.400601
temp          0.394454
atemp         0.389784
humidity      0.317371
year          0.260403
month         0.166862
season        0.163439
weather       0.128655
windspeed     0.101369
day           0.019826
workingday    0.011594
holiday       0.005393
dayofweek     0.002283
Name: count, dtype: float64
```

- hour, temp, atemp, humidity, year, month, season, weather 순서로 count와 상관관계가 높음.

4. Data Preprocessing

```
# 결측치
train.isna().sum()
```

```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
count         0
year          0
month         0
day           0
hour          0
dayofweek     0
dtype: int64
```

```
train[train['windspeed']==0]
```

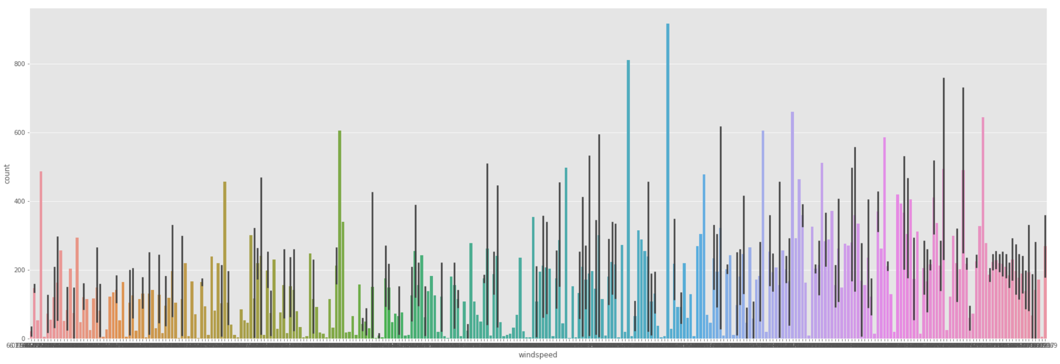
	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	year
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	16	2011
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	40	2011
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	32	2011
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	13	2011
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	1	2011
...	...	...	...	...	...	...	...	...	...	...	...
10826	2012-12-17 12:00:00	4	0	1	2	16.40	20.455	87	0.0	232	2012
10829	2012-12-17 15:00:00	4	0	1	2	17.22	21.210	88	0.0	211	2012
10846	2012-12-18 08:00:00	4	0	1	1	15.58	19.695	94	0.0	662	2012
10860	2012-12-18 22:00:00	4	0	1	1	13.94	16.665	49	0.0	132	2012
10862	2012-12-19 00:00:00	4	0	1	1	12.30	15.910	61	0.0	41	2012

1313 rows × 15 columns

```
# windspeed 0 결측치로 만들어 채워넣기
train.set_index('datetime', inplace=True)
train.loc[train['windspeed'] == 0, "windspeed"] = np.nan
train.interpolate(method='time', inplace=True)
train.fillna(0, inplace=True)
train.reset_index(inplace=True)
```

```
# 바람
plt.figure(figsize=(30,10))
sns.barplot(data=train, x='windspeed', y='count')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1ca67c6c8>
```



```
# 이상치 제거
def del_outlier(data, feature):
    q1 = np.percentile(feature, 25)
    q3 = np.percentile(feature, 75)
    IQR = q3-q1
    new_data = data[(feature>=q1-(1.5*IQR)) & (feature<=q3+(1.5*IQR))]
    return new_data
```

```
print(train.shape)
print(del_outlier(train, train['count']).shape)
```

```
(10886, 15)
(10586, 15)
```

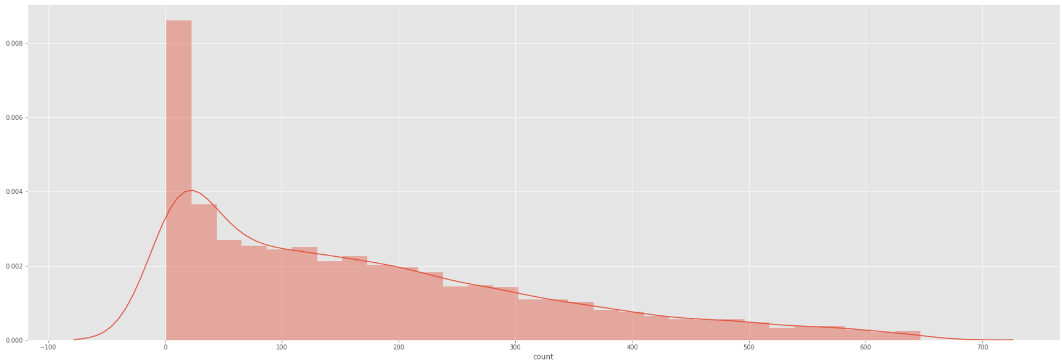
```
# 전체 변수의 이상치 제거
for col in train.columns:
    if col != 'datetime' :
        train = del_outlier(train, train[col])
```

```
print(train.shape)
```

```
(9857, 15)
```

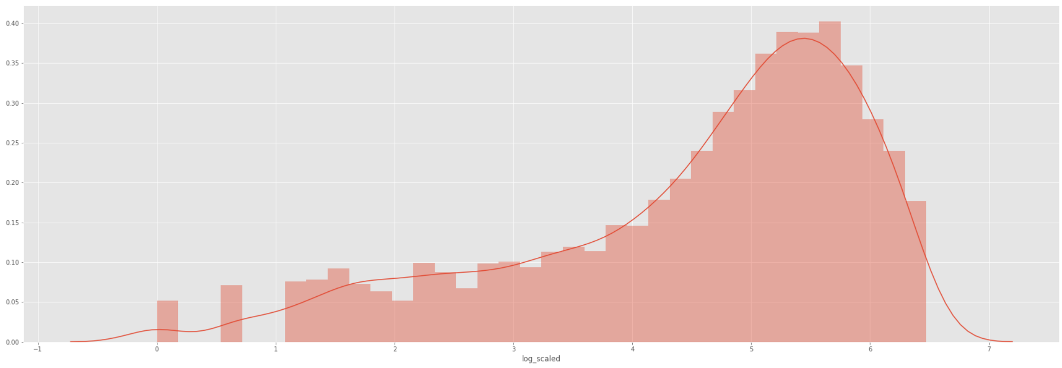
```
# 분포
plt.figure(figsize = (30,10))
sns.distplot(train['count'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1d0a31ac8>
```



```
# log scale 후 데이터 분포 보기
train['log_scaled'] = np.log(train['count'])
plt.figure(figsize = (30,10))
sns.distplot(train['log_scaled'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1d0e89148>
```



```
train.head().transpose()
```

	0	1	2	3	4
datetime	2011-01-01 00:00:00	2011-01-01 01:00:00	2011-01-01 02:00:00	2011-01-01 03:00:00	2011-01-01 04:00:00
season	1	1	1	1	1
holiday	0	0	0	0	0
workingday	0	0	0	0	0
weather	1	1	1	1	1
temp	9.84	9.02	9.02	9.84	9.84
atemp	14.395	13.635	13.635	14.395	14.395
humidity	81	80	80	75	75
windspeed	0	0	0	0	0
count	16	40	32	13	1
year	2011	2011	2011	2011	2011
month	1	1	1	1	1
day	1	1	1	1	1
hour	0	1	2	3	4
dayofweek	5	5	5	5	5
log_scaled	2.77259	3.68888	3.46574	2.56495	0

```
# workingday 기준으로 데이터를 split 하여 학습한다.
nowork_train = train[train['workingday']==0]
work_train = train[train['workingday']==1]
```

```
print(nowork_train.shape)
print(work_train.shape)
```

```
(3005, 16)
(6852, 16)
```

5. Modeling

```
def rmsle(y, y_,convertExp=True):
    if convertExp:
        y = np.exp(y),
        y_ = np.exp(y_)
    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))
```

```
def std_scale(data):
    scaler = StandardScaler()
    scaler.fit(data)
    scaled_data = pd.DataFrame(scaler.transform(data), columns = data.columns)
    return scaled_data
```

```
def fit_eval(X, y):

    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

    yLog_train = np.log1p(y_train)
    yLog_test = np.log1p(y_test)

    lr = LinearRegression()
    lr.fit(X_train, yLog_train)
    pred_test = lr.predict(X_test)

    MSE = mean_squared_error(pred_test, yLog_test)
    RMSLE = rmsle(np.exp(yLog_test),np.exp(pred_test),False)
    score = lr.score(X_test, yLog_test)

    print('MSE : ', MSE)
```

```python
    print ("RMSLE Value For Linear Regression: ",RMSLE)
    print('score : ',score)

    return pred_test, MSE, RMSLE, score
```

```python
def ensemble_model(data1, data2) :

    y1 = data1['count']
    X1 = data1.drop(['datetime', 'count', 'log_scaled'], axis=1)

    print('data1')
    pred_test1, MSE1, RMSLE1, score1 = fit_eval(X1, y1)
    print("="*50)

    y2 = data2['count']
    X2 = data2.drop(['datetime', 'count', 'log_scaled'], axis=1)

    print('data2')
    pred_test2, MSE2, RMSLE2, score2 = fit_eval(X2, y2)
    print("="*50)

    avg_MSE = (MSE1 + MSE2) / 2
    avg_score = (score1 + score2) / 2

    print('avg_MSE : ', avg_MSE)
    print('avg_score : ', avg_score)
    pred_test = np.concatenate((pred_test1, pred_test2), axis=0)

    return pred_test
```

```python
def scaled_ensemble_model(data1, data2) :

    y1 = data1['count']
    X1 = data1.drop(['datetime', 'count', 'log_scaled'], axis=1)
    scaled_X1 = std_scale(X1)

    print('data1')
    pred_test1, MSE1, RMSLE1, score1 = fit_eval(scaled_X1, y1)
    print("="*50)

    y2 = data2['count']
    X2 = data2.drop(['datetime', 'count', 'log_scaled'], axis=1)
    scaled_X2 = std_scale(X2)

    print('data2')
    pred_test2, MSE2, RMSLE2, score2 = fit_eval(scaled_X2, y2)
    print("="*50)

    avg_MSE = (MSE1 + MSE2) / 2
    avg_score = (score1 + score2) / 2

    print('avg_MSE : ', avg_MSE)
    print('avg_score : ', avg_score)
    pred_test = np.concatenate((pred_test1, pred_test2), axis=0)

    return pred_test
```

```python
pred_test = ensemble_model(nowork_train, work_train)
```

```
data1
MSE :  0.7759768709188656
RMSLE Value For Linear Regression:  0.8537543543562682
score :  0.5159560248838708
==================================================
data2
MSE :  1.0048970453778279
RMSLE Value For Linear Regression:  0.9641469772563248
score :  0.4935592780885689
==================================================
avg_MSE :  0.8904369581483467
avg_score :  0.5047576514862199
```

```python
scaled_pred_test = scaled_ensemble_model(nowork_train, work_train)
```

```
data1
MSE :  1.0327083375606902
RMSLE Value For Linear Regression:  0.974984542201302
score :  0.45378933744027705
==================================================
data2
MSE :  1.0350660547398114
RMSLE Value For Linear Regression:  0.9758414950221532
score :  0.4960917055425429
==================================================
avg_MSE :  1.0338871961502507
avg_score :  0.47494052149141
```

- train_test_split이 랜덤하게 적용되기 때문에 실행할 때마다 score값이 변하게 된다.

test데이터 적용

```
# 모델 학습
def fitting_lr(train1, train2):

    y1 = train1['count']
    X1 = train1.drop(['datetime', 'count', 'log_scaled'], axis=1)
    scaled_X1 = std_scale(X1)
    yLog_train1 = np.log1p(y1)

    model1 = LinearRegression()
    model1.fit(scaled_X1, yLog_train1)

    y2 = train2['count']
    X2 = train2.drop(['datetime', 'count', 'log_scaled'], axis=1)
    scaled_X2 = std_scale(X2)
    yLog_train2 = np.log1p(y2)

    model2 = LinearRegression()
    model2.fit(scaled_X2, yLog_train2)

    return model1, model2
```

```
nowork_model, work_model = fitting_lr(nowork_train, work_train)
```

```
# 테스트 데이터 전처리
def test_preprocess(test) :
    test['datetime'] = pd.to_datetime(test['datetime'])
    datetime = test['datetime']
    test['year'] = datetime.dt.year
    test['month'] = datetime.dt.month
    test['day'] = datetime.dt.day
    test['hour'] = datetime.dt.hour
    test['dayofweek'] = datetime.dt.dayofweek

    test.set_index('datetime', inplace=True)
    test.loc[test['windspeed'] == 0, "windspeed"] = np.nan
    test.interpolate(method='time', inplace=True)
    test.fillna(0, inplace=True)
    test.reset_index(inplace=True)

    print(test.shape)

    return test
```

```
test = pd.read_csv('C:\\ai\\workspace\\data\\bike\\test.csv')
test.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.0014
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.0014

```
test = test_preprocess(test)
```

```
(6493, 14)
```

```
test.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	year	month	day
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.002700	2011	1	20
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	21.002267	2011	1	20
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	16.001833	2011	1	20
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.001400	2011	1	20
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.001400	2011	1	20

```
nowork_test = test[test['workingday']==0]
work_test = test[test['workingday']==1]
```

```
datetimecol1 = nowork_test["datetime"]
datetimecol2 = work_test["datetime"]
```

```
X1 = nowork_test.drop(['datetime'], axis=1)
X2 = work_test.drop(['datetime'], axis=1)
scaled_X1 = std_scale(X1)
scaled_X2 = std_scale(X2)
```

```
nowork_predict = nowork_model.predict(scaled_X1)
```

```
work_predict = work_model.predict(scaled_X2)
```

```
submission1 = pd.DataFrame({
        "datetime": datetimecol1,
        "count": [max(0, x) for x in np.exp(nowork_predict)]
    })
```

```
submission2 = pd.DataFrame({
        "datetime": datetimecol2,
        "count": [max(0, x) for x in np.exp(work_predict)]
    })
```

```
submission = pd.concat([submission1, submission2])
submission = submission.sort_values(by='datetime')

submission.to_csv('bike_predictions.csv', index=False)
```

```
submission.shape
```

```
(6493, 2)
```

결과

Name	Submitted	Wait time	Execution time	Score
bike_predictions.csv	a minute ago	0 seconds	0 seconds	1.06903

Complete

강사님 분석

```
# 여기부터 강사님코드
```

```
y = train['count']
y.head()
```

```
0    16
1    40
2    32
3    13
4     1
Name: count, dtype: int64
```

```
X = train.drop(['datetime', 'count', 'log_scaled'], axis=1)
X.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	year	month	day	hour
0	1	0	0	1	9.84	14.395	81	0.0	2011	1	1	0
1	1	0	0	1	9.02	13.635	80	0.0	2011	1	1	1
2	1	0	0	1	9.02	13.635	80	0.0	2011	1	1	2
3	1	0	0	1	9.84	14.395	75	0.0	2011	1	1	3
4	1	0	0	1	9.84	14.395	75	0.0	2011	1	1	4

```
len(X)
```

```
9857
```

```
len(y)
```

```
9857
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

```
yLog_train = np.log1p(y_train)
yLog_test = np.log1p(y_test)
```

```
lr = LinearRegression()
lr.fit(X_train, yLog_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
# Train 데이터
```

```
pred_train = lr.predict(X_train)
print('MSE : ', mean_squared_error(pred_train, yLog_train))
print ("RMSLE Value For Linear Regression: ",rmsle(np.exp(yLog_train),np.exp(pred_train),False))
print('score : ', lr.score(X_train, yLog_train))
```

```
MSE :  1.010815392961708
RMSLE Value For Linear Regression:  0.9672261266550728
score :  0.481356525605516
```

```
# Test 데이터
```

```
pred_test = lr.predict(X_test)
print('MSE : ', mean_squared_error(pred_test, yLog_test))
print ("RMSLE Value For Linear Regression: ",rmsle(np.exp(yLog_test),np.exp(pred_test),False))
print('score : ', lr.score(X_test, yLog_test))
```

```
MSE :  1.0347322023713155
RMSLE Value For Linear Regression:  0.9749764750955473
score :  0.48125230809464903
```

```

```

```
# 추가
```

```
# Standard Scale
scaler = StandardScaler()
scaler.fit(X)
scaled_X = pd.DataFrame(scaler.transform(X), columns = X.columns)
scaled_X.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	year	month	day	
0	-1.364818	0.0	-1.510033	-0.664819	-1.329030	-1.098823	0.962179	-2.243459	-0.97723	-1.622182	-1.638389	-1.62
1	-1.364818	0.0	-1.510033	-0.664819	-1.434835	-1.189506	0.909101	-2.243459	-0.97723	-1.622182	-1.638389	-1.48
2	-1.364818	0.0	-1.510033	-0.664819	-1.434835	-1.189506	0.909101	-2.243459	-0.97723	-1.622182	-1.638389	-1.33
3	-1.364818	0.0	-1.510033	-0.664819	-1.329030	-1.098823	0.643714	-2.243459	-0.97723	-1.622182	-1.638389	-1.19
4	-1.364818	0.0	-1.510033	-0.664819	-1.329030	-1.098823	0.643714	-2.243459	-0.97723	-1.622182	-1.638389	-1.05

```
X_train, X_test, y_train, y_test = train_test_split(scaled_X,y,test_size=0.2)
```

```
yLog_train = np.log1p(y_train)
yLog_test = np.log1p(y_test)
```

```
lr = LinearRegression()
lr.fit(X_train, yLog_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
pred_test = lr.predict(X_test)
print('MSE : ', mean_squared_error(pred_test, yLog_test))
print ("RMSLE Value For Linear Regression: ",rmsle(np.exp(yLog_test),np.exp(pred_test),False))
print('score : ', lr.score(X_test, yLog_test))
```

```
MSE :  1.0748154194572215
RMSLE Value For Linear Regression:  0.9945454748631787
score :  0.47263176769660586
```