5장 _ 회귀

5.6 규제 선형 모델 – 릿지, 라쏘, 엘라스틱넷

5.7 로지스틱 회귀

5.8 회귀 트리

5.9 회귀 실습 – 자전거 대여 수요 예측

5.10 회귀 실습 – 캐글 주택 가격 : 고급 회귀 기법

***회귀 분석 순서**

0. 데이터 클렌징 및 가공

1. 회귀 모델 적용! 학습/예측/평가
   + 결과값이 정규분포인지? 피처 인코딩

   -LinearRegression, Ridge,Lasso 이용
   -회귀트리 이용

*캐글 Bike Sharing Demand 예측 경연

Bike_train.csv : 2011년 1월부터 2012년 12월까지 날짜/시간, 기온, 습도, 풍속 등의 정보를 기반으로 1시간 간격 동안의 자전거 대여 횟수 data

	A	B	C	D	E	F	G	H	I	J	K	L
1	datetime	season	holiday	workingda	weather	temp	atemp	humidity	windspeec	casual	registered	count
2	2011-01-01 0:00	1	0	0	1	9.84	14.395	81	0	3	13	16
3	2011-01-01 1:00	1	0	0	1	9.02	13.635	80	0	8	32	40
4	2011-01-01 2:00	1	0	0	1	9.02	13.635	80	0	5	27	32
5	2011-01-01 3:00	1	0	0	1	9.84	14.395	75	0	3	10	13
6	2011-01-01 4:00	1	0	0	1	9.84	14.395	75	0	0	1	1
7	2011-01-01 5:00	1	0	0	2	9.84	12.88	75	6.0032	0	1	1
8	2011-01-01 6:00	1	0	0	1	9.02	13.635	80	0	2	0	2
⋮												
10882	2012-12-19 18:00	4	0	1	1	15.58	19.695	50	23.9994	23	546	569
10883	2012-12-19 19:00	4	0	1	1	15.58	19.695	50	26.0027	7	329	336
10884	2012-12-19 20:00	4	0	1	1	14.76	17.425	57	15.0013	10	231	241
10885	2012-12-19 21:00	4	0	1	1	13.94	15.91	61	15.0013	4	164	168
10886	2012-12-19 22:00	4	0	1	1	13.94	17.425	61	6.0032	12	117	129
10887	2012-12-19 23:00	4	0	1	1	13.12	16.665	66	8.9981	4	84	88

- datetime: hourly date + timestamp
- season: 1 = 봄, 2 = 여름, 3 = 가을, 4 = 겨울
- holiday: 1 = 토, 일요일의 주말을 제외한 국경일 등의 휴일, 0 = 휴일이 아닌 날
- workingday: 1 = 토, 일요일의 주말 및 휴일이 아닌 주중, 0 = 주말 및 휴일
- weather:
  - 1 = 맑음, 약간 구름 낀 흐림
  - 2 = 안개, 안개 + 흐림
  - 3 = 가벼운 눈, 가벼운 비 + 천둥
  - 4 = 심한 눈/비, 천둥/번개
- temp: 온도(섭씨)
- atemp: 체감온도(섭씨)
- humidity: 상대습도
- windspeed: 풍속
- casual: 사전에 등록되지 않는 사용자가 대여한 횟수
- registered: 사전에 등록된 사용자가 대여한 횟수
- count: 대여 횟수

0. 데이터 클렌징 및 가공

In [3]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)

bike_df = pd.read_csv('./bike_train.csv')
print(bike_df.shape)
bike_df.head(3)
```

```
(10886, 12)
```

Out [3]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32

->10886개의 레코드, 12개의 컬럼으로 구성

0. 데이터 클렌징 및 가공

```
In [2]: bike_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 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
casual        10886 non-null int64
registered    10886 non-null int64
count         10886 non-null int64
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.6+ KB
```

->row 데이터 중 Null 데이터는 없다
->datatime컬럼만 object 형
->datatime을 년,월,일,시간 4개의 속성으로 분리 필요!
->문자열을 'datatime'타입으로 변경하는 apply메서드를 이용하여 년,월,일,시간 칼럼을 추출

0. 데이터 클렌징 및 가공

In [4]:
```
# 문자열을 datetime 타입으로 변경.
bike_df['datetime'] = bike_df.datetime.apply(pd.to_datetime)

# datetime 타입에서 년, 월, 일, 시간 추출
bike_df['year'] = bike_df.datetime.apply(lambda x : x.year)
bike_df['month'] = bike_df.datetime.apply(lambda x : x.month)
bike_df['day'] = bike_df.datetime.apply(lambda x : x.day)
bike_df['hour'] = bike_df.datetime.apply(lambda x: x.hour)
bike_df.head(3)
```

Out[4]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	year	month	day	hour
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	2011	1	1	0
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	2011	1	1	1
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	2011	1	1	2

In [5]:
```
drop_columns = ['datetime','casual','registered']
bike_df.drop(drop_columns, axis=1,inplace=True)
```

->datatime, casual, registered 칼럼 제거

1. 회귀 모델 적용! 학습/예측/평가

->RMSLE(Root Mean Square Log Error),RMSE,MSE 한꺼번에 평가하는 함수 evaluate_regr(y,pred) 만들기

In [7]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error

# log 값 변환 시 NaN등의 이슈로 log() 가 아닌 log1p() 를 이용하여 RMSLE 계산
def rmsle(y, pred):
    log_y = np.log1p(y)
    log_pred = np.log1p(pred)
    squared_error = (log_y - log_pred) ** 2
    rmsle = np.sqrt(np.mean(squared_error))
    return rmsle

# 사이킷런의 mean_square_error() 를 이용하여 RMSE 계산
def rmse(y,pred):
    return np.sqrt(mean_squared_error(y,pred))

# MSE, RMSE, RMSLE 를 모두 계산
def evaluate_regr(y,pred):
    rmsle_val = rmsle(y,pred)
    rmse_val = rmse(y,pred)
    # MSE 는 scikit learn의 mean_absolute_error() 로 계산
    mse_val = mean_absolute_error(y,pred)
    print('RMSLE: {0:.3f}, RMSE: {1:.3F}, MSE: {2:.3F}'.format(rmsle_val, rmse_val, mse_val))
```

```
# 다음과 같은 rmsle 구현은 오버플로나 언더플로 오류를 발생하기 쉽습니다.
def rmsle(y, pred):
    msle = mean_squared_log_error(y, pred)
    rmsle = np.sqrt(mse)
    return rmsle
```

->1+log() 값인 log1p()는 데이터 값의 크기에 따라 오버플로/언더플로 오류가 발생하는 문제를 해결!
->log1p()로 변환된 값은 다시 넘파이의 expm1() 함수로 쉽게 원래의 스케일로 복원 가능!

1. 회귀 모델 적용! 학습/예측/평가

In [8]:
```
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.linear_model import LinearRegression , Ridge , Lasso

y_target = bike_df['count']
X_features = bike_df.drop(['count'],axis=1,inplace=False)

X_train, X_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.3, random_state=0)

lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
pred = lr_reg.predict(X_test)

evaluate_regr(y_test ,pred)
```

```
RMSLE: 1.165, RMSE: 140.900, MSE: 105.924
```

->사이킷런의 LinearRegression 객체를 이용해 회귀 예측
->실제 Target 데이터 값인 대여 횟수(Count)를 감안하면 예측 오류로서는 비교적 큰 값

1. 회귀 모델 적용! 학습/예측/평가

In [9]:
```
def get_top_error_data(y_test, pred, n_tops = 5):
    # DataFrame에 컬럼들로 실제 대여횟수(count)와 예측 값을 서로 비교 할 수 있도록 생성.
    result_df = pd.DataFrame(y_test.values, columns=['real_count'])
    result_df['predicted_count']= np.round(pred)
    result_df['diff'] = np.abs(result_df['real_count'] - result_df['predicted_count'])
    # 예측값과 실제값이 가장 큰 데이터 순으로 출력.
    print(result_df.sort_values('diff', ascending=False)[:n_tops])

get_top_error_data(y_test,pred,n_tops=5)
```

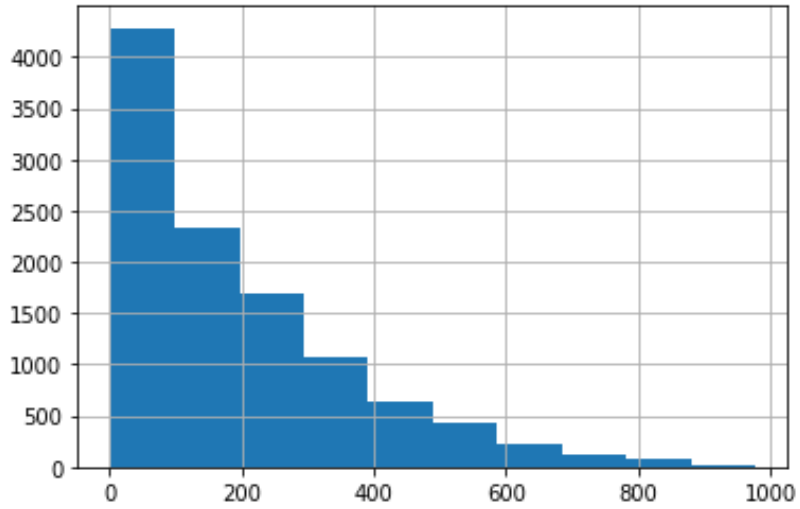
	real_count	predicted_count	diff
1618	890	322.0	568.0
3151	798	241.0	557.0
966	884	327.0	557.0
412	745	194.0	551.0
2817	856	310.0	546.0

->실제값과 예측값이 어느 정도 차이가 나는지 dataframe의 칼럼으로 만들어 확인
->큰 예측 오류가 발생했음을 알 수 있다.
->회귀에서 큰 예측 오류가 발생한 경우, Target 값의 분포가 왜곡된 형태를 이루고 있는지 가장 먼저 확인

***선형 회귀는 피처값과 타깃값의 분포가 정규분포 형태를 매우 선호
왜곡된 형태의 분포도일 경우, 예측 성능에 부정적인 영향을 미친다.

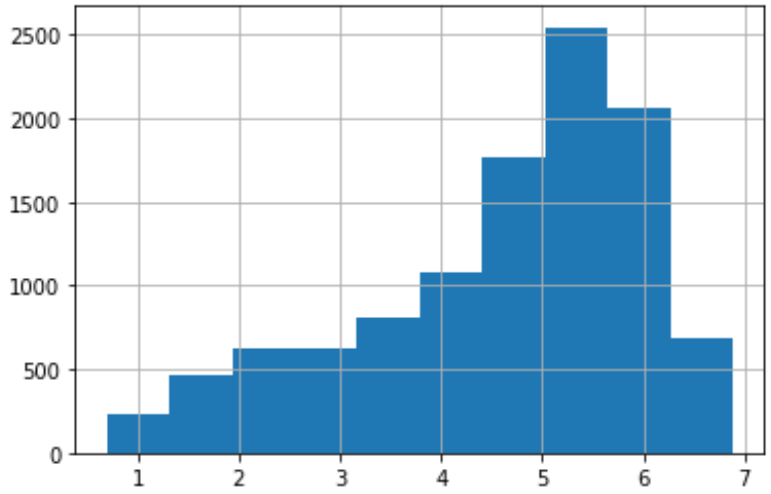
1. 회귀 모델 적용! 학습/예측/평가

```
In [10]: y_target.hist()
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1ced3589f28>
```



->count 컬럼 값이 정규 분포가 아닌
0~200 사이에 왜곡된 모습

```
In [11]: y_log_transform = np.log1p(y_target)
         y_log_transform.hist()
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1ced40bd6d8>
```



->넘파이의 log1p()를 이용한 로그 변환
->왜곡된 정도가 많이 향상된 모습

***왜곡된 값을 정규 분포 형태로 바꾸는 방법
1. 표준정규분포로 변환 : 예측 성능 항상 크게 기대하기 어려운 경우 많다
2. 최대값/최소값 정규화 : 피처의 개수가 많은 경우 과적합의 이슈가 발생할 수 있다
3. 로그변환

1. 회귀 모델 적용! 학습/예측/평가

In [12]:
```
# 타겟 컬럼인 count 값을 log1p 로 Log 변환
y_target_log = np.log1p(y_target)

# 로그 변환된 y_target_log를 반영하여 학습/테스트 데이터 셋 분할
X_train, X_test, y_train, y_test = train_test_split(X_features, y_target_log, test_size=0.3, random_state=0)
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
pred = lr_reg.predict(X_test)

# 테스트 데이터 셋의 Target 값은 Log 변환되었으므로 다시 expm1를 이용하여 원래 scale로 변환
y_test_exp = np.expm1(y_test)

# 예측 값 역시 Log 변환된 타겟 기반으로 학습되어 예측되었으므로 다시 exmp1으로 scale변환
pred_exp = np.expm1(pred)

evaluate_regr(y_test_exp ,pred_exp)
```

```
RMSLE: 1.017, RMSE: 162.594, MSE: 109.286
```

->RMSLE 오류는 줄어들었지만, RMSE는 오히려 늘어남

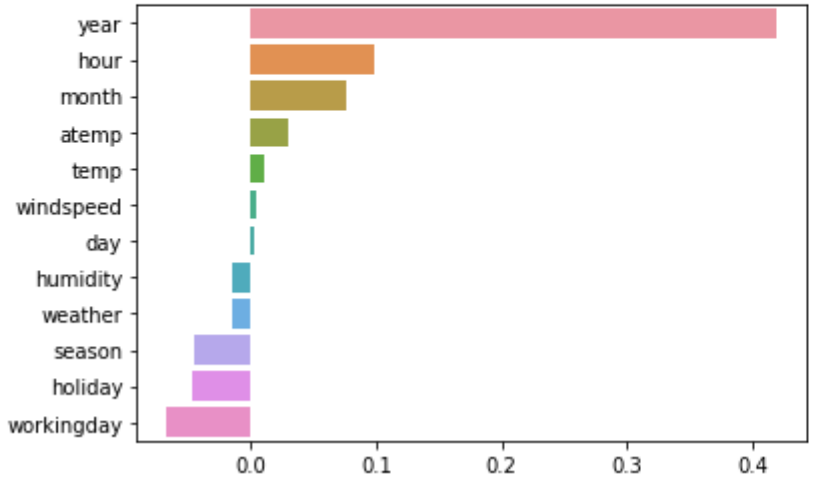
로그 변환 전 예측성능평가 결과

```
RMSLE: 1.165, RMSE: 140.900, MSE: 105.924
```

1. 회귀 모델 적용! 학습/예측/평가

```
In [13]: coef = pd.Series(lr_reg.coef_, index=X_features.columns)
         coef_sort = coef.sort_values(ascending=False)
         sns.barplot(x=coef_sort.values, y=coef_sort.index)
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1ced41b7128>
```



year	month	day	hour
2011	1	1	0
2011	1	1	1
2011	1	1	2

->아까 삭제한 datatime 컬럼만 object형이라 가공해서 문자열을 datatime 타입으로 변경했었음

->각 피처의 회귀 계수 값들을 시각화 한 결과
->Year 피처의 회귀 계수 값이 독보적으로 큰 값을 가지고 있음
->숫자형 카테고리 값을 선형 회귀에 사용할 경우 회귀 계수를 연산할 때 숫자형 값에 크게 영향을 받는 경우 선형 회귀에서는 이러한 피처 인코딩에 원-핫 인코딩을 적용해 변환해야한다

1. 회귀 모델 적용! 학습/예측/평가

->판다스의 get_dummies()를 이용해
원-핫 인코딩한 후 예측 성능 확인

In [14]:
```
# 'year', 'month', 'hour', 'season', 'weather' feature들을 One Hot Encoding
X_features_ohe = pd.get_dummies(X_features, columns=['year','month','hour', 'holiday',
                                                     'workingday','season','weather'])
```

In [15]:
```
# 원-핫 인코딩이 적용된 feature 데이터 세트 기반으로 학습/예측 데이터 분할.
X_train, X_test, y_train, y_test = train_test_split(X_features_ohe, y_target_log,
                                                    test_size=0.3, random_state=0)

# 모델과 학습/테스트 데이터 셋을 입력하면 성능 평가 수치를 반환
def get_model_predict(model, X_train, X_test, y_train, y_test, is_expm1=False):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    if is_expm1 :
        y_test = np.expm1(y_test)
        pred = np.expm1(pred)
    print('###',model.__class__.__name__,'###')
    evaluate_regr(y_test, pred)
# end of function get_model_predict

# model 별로 평가 수행
lr_reg = LinearRegression()
ridge_reg = Ridge(alpha=10)
lasso_reg = Lasso(alpha=0.01)

for model in [lr_reg, ridge_reg, lasso_reg]:
    get_model_predict(model,X_train, X_test, y_train, y_test,is_expm1=True)
```

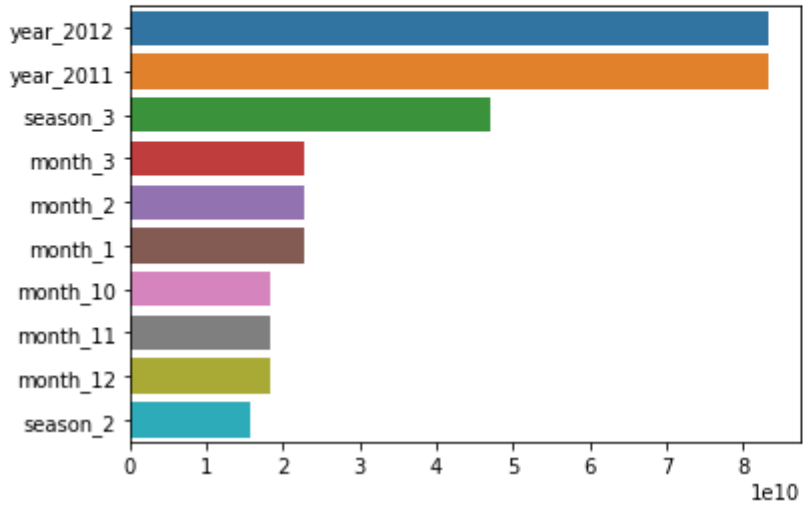
```
### LinearRegression ###
RMSLE: 0.589, RMSE: 97.485, MSE: 63.107
### Ridge ###
RMSLE: 0.589, RMSE: 98.407, MSE: 63.648
### Lasso ###
RMSLE: 0.634, RMSE: 113.031, MSE: 72.658
```

->선형 회귀의 예측 성능이 많이 향상된 결과

1. 회귀 모델 적용! 학습/예측/평가

In [16]:
```
coef = pd.Series(lr_reg.coef_ , index=X_features_ohe.columns)
coef_sort = coef.sort_values(ascending=False)[:10]
sns.barplot(x=coef_sort.values , y=coef_sort.index)
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1ced41b1908>



->원-핫 인코딩된 데이터 세트에서 회귀 계수가 높은 피처를 다시 시각화한 결과
->계절, 월, 날씨 등 자전거를 타는데 필요한 피처의 회귀 계수가 높아짐

1. 회귀 모델 적용! 학습/예측/평가

In [17]:
```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor


# 랜덤 포레스트, GBM, XGBoost, LightGBM model 별로 평가 수행
rf_reg = RandomForestRegressor(n_estimators=500)
gbm_reg = GradientBoostingRegressor(n_estimators=500)
xgb_reg = XGBRegressor(n_estimators=500)
lgbm_reg = LGBMRegressor(n_estimators=500)

for model in [rf_reg, gbm_reg, xgb_reg, lgbm_reg]:
    get_model_predict(model,X_train, X_test, y_train, y_test,is_expm1=True)
```

```
### RandomForestRegressor ###
RMSLE: 0.352, RMSE: 50.634, MSE: 31.309
### GradientBoostingRegressor ###
RMSLE: 0.341, RMSE: 55.749, MSE: 34.320
### XGBRegressor ###
RMSLE: 0.346, RMSE: 56.474, MSE: 34.917
### LGBMRegressor ###
RMSLE: 0.316, RMSE: 46.473, MSE: 28.777
```

->회귀 트리를 이용해 회귀 예측을 수행한 결과
->앞서 적용한 Target 값의 로그 변환된 값과 원-핫 인코딩된 피처 데이터 세트를 그대로 이용
->선형 회귀 모델보다 회귀 예측 성능이 개선된 결과

감사합니다 :)