1 기계학습

회귀

라이브러리 호출

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
pd.options.display.max_columns = None
```

그래프 한글 깨짐 방지

In [2]:
```
from matplotlib import font_manager, rc
path='malgun.ttf'
font_name = font_manager.FontProperties(fname=path).get_name()
rc('font', family=font_name)
```

데이터 로딩

In [3]:
```
df = pd.read_csv('./data/bikeshare.csv')
```

데이터 구조 확인

In [4]:
```
df.head()
```

Out[4]:

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

In [5]:
```
df.tail()
```

Out[5]:

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

In [6]:
```
df.shape
```

Out[6]:
```
(10886, 12)
```

In [7]:
```
df.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  object 
 1   season      10886 non-null  object 
 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: float64(3), int64(7), object(2)
memory usage: 1020.7+ KB
```

데이터 타입 맞춰주기

In [8]:
```
df.columns
```

Out[8]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

In [9]:
```
col_id = []
col_dt = ['datetime']
col_cat = ['season']
col_int = ['weather', 'humidity','windspeed','casual','registered','count']
col_float = ['temp','atemp']
col_bool = ['holiday', 'workingday']
col_num = col_int + col_float
```

In [10]:
```
df['datetime'] = pd.to_datetime(df['datetime'])
df[col_cat] = df[col_cat].astype('str')
df[col_int] = df[col_int].astype('int', errors = 'ignore')
df[col_float] = df[col_float].astype('float')
```

DQ Check(빈도분석, 분포분석)

연속형 변수

In [11]:
```
def DA(data):
    da = data.describe(percentiles=[0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95])
    da = da.T
    df1 = data.isna().sum()
    df1.name = 'missing'
    df2 = data.median()
    df2.name = 'median'
    df3 = np.var(data)
    df3.name = 'variance'
    df4 = data.skew()
    df4.name = 'skewness'
    df5 = data.kurtosis()
    df5.name = 'kurtosis'

    da = pd.concat([da, df1, df2, df3, df4, df5], axis =1)
    da['total'] = da['count'] + da['missing']

    col_nm = da.columns.tolist()
    order = ['total','count','missing','mean','median','std','variance','skewn
ess','kurtosis','min',
             '5%','10%','25%','50%','75%','90%','95%','max']
    col_nm_new=[]
    for i in order:
        col_nm_new.append(i)
    da = da[col_nm_new]
    da = da.round(2)
    return da
```

In [12]:
```
DA1 = DA(df[col_num])
DA1
```

Out[12]:

	total	count	missing	mean	median	std	variance	skewness	kurtosis	m
weather	10886.0	10886.0	0	1.42	1.00	0.63	0.40	1.24	0.40	1.
humidity	10886.0	10886.0	0	61.89	62.00	19.25	370.34	-0.09	-0.76	0.
windspeed	10886.0	10886.0	0	12.43	12.00	8.05	64.73	0.65	0.72	0.
casual	10886.0	10886.0	0	36.02	17.00	49.96	2495.82	2.50	7.55	0.
registered	10886.0	10886.0	0	155.55	118.00	151.04	22810.69	1.52	2.63	0.
count	10886.0	10886.0	0	191.57	145.00	181.14	32810.30	1.24	1.30	1.
temp	10886.0	10886.0	0	20.23	20.50	7.79	60.70	0.00	-0.91	0.
atemp	10886.0	10886.0	0	23.66	24.24	8.47	71.81	-0.10	-0.85	0.

범주형 변수

In [13]:
```
def DA_cat(data, col_cat):
    DA_cat = pd.DataFrame()

    for i in col_cat:
        a = data[i].value_counts(dropna=False).to_frame().sort_index().rename(
columns={i:'count'}).reset_index()
        a['col_nm'] = i
        a = a.rename(columns = {'index':'class'})
        a = a[['col_nm','class','count']]
        b=data[i].value_counts(dropna = False, normalize = True).to_frame().so
rt_index().rename(
        columns = {i:'ratio'}).reset_index()
        b = b['ratio'].to_frame()
        b['ratio'] = b['ratio'].round(2)
        c = pd.concat([a,b], axis = 1)
        DA_cat = pd.concat([DA_cat, c], axis=0)
    DA_cat = DA_cat.reset_index(drop=True)
    return DA_cat
```

In [14]:
```
DA2 = DA_cat(df,col_cat+col_bool)
DA2
```

Out[14]:

	col_nm	class	count	ratio
0	season	A	2686	0.25
1	season	B	2733	0.25
2	season	C	2733	0.25
3	season	D	2734	0.25
4	holiday	0	10575	0.97
5	holiday	1	311	0.03
6	workingday	0	3474	0.32
7	workingday	1	7412	0.68

전처리(중복값, 결측치, 이상치 처리)

중복값

In [15]:
```
df[df.duplicated(keep=False)]
```

Out[15]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	r

In [ ]:
```
df.drop_duplicates()
df.drop_duplicates(['col1'], keep='last')
```

결측치

In [16]:
```
df.isna().sum()
```

Out[16]:
```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

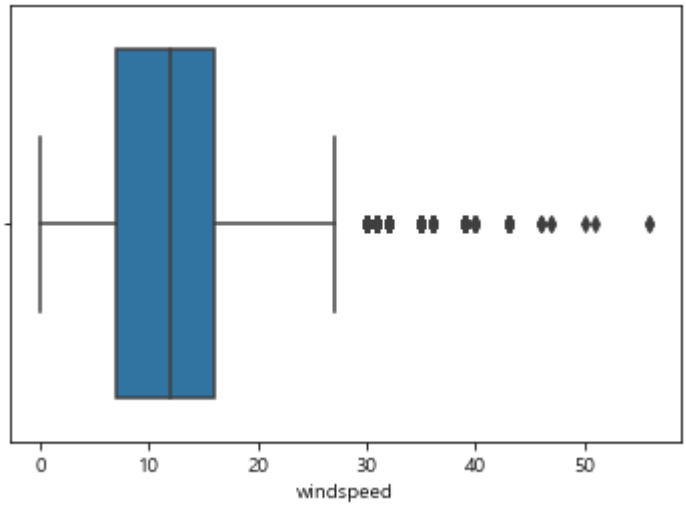
In [ ]:
```
# na 처리 : dropna(), fillna()
df.dropna() # nan이 하나라도 들어간 행은 삭제
df.dropna(how = 'all') # 데이터가 모두 nan인 행만 삭제 / 초기값:'any'
## Parameters
# axis = 'index' / 'columns'
# subset = ['col1', 'col2', ...] # 적용 대상 컬럼 특정

df.fillna(0) # na를 0으로 채우기

new_data = {'a':0, 'b':1, 'c':-999}
df.fillna(new_data) # na 발생 시 a 열에는 0, b 열에는 1, c 열에는 -999로 채움
df.fillna(new_data, limit = 2) # 각 열별로 2개의 nan까지 대체
df.fillna(method = 'ffill') # 열 별로 바로 앞의 데이터로 채움
df.fillna(method = 'bfill') # 열 별로 바로 뒤의 데이터로 채움
# ffill의 경우 첫 행이거나, 앞의 데이터가 nan일 경우 nan유지. bfill도 반대로 동일

# 평균값, 중앙값으로 대치
df.loc[19,'Leaflets'] = df['Leaflets'].mean() # 평균값으로
df.loc[19,'Leaflets'] = df['Leaflets'].median # 중앙값으로
```

이상치

In [17]:
```
tmp = 'windspeed'
```

In [18]:
```
sns.boxplot(y = tmp, data = df, orient = 'h')
```

Out[18]: <AxesSubplot:xlabel='windspeed'>



In [19]:
```
# IQR 활용
q1 = df[tmp].quantile(.25)
q3 = df[tmp].quantile(.75)
iqr = q3-q1
min_iqr = q1 - 1.5 * iqr
max_iqr = q3 + 1.5 * iqr
min_from_all = df[tmp].min()
max_from_all = df[tmp].max()
if (min_iqr < min_from_all) :
    min_iqr = min_from_all
if (max_iqr > max_from_all) :
    max_iqr = max_from_all

outlier = df[(df[tmp] < min_iqr ) | (df[tmp] > max_iqr)] # 이상치 조회
outlier_index = outlier.index
print(outlier.shape)
outlier
```

```
(427, 12)
```

Out[19]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	cas
42	2011-01-02 19:00:00	A	0	0	1	13.12	14.395	42	30	
175	2011-01-08 14:00:00	A	0	0	1	8.20	8.335	32	32	
176	2011-01-08 15:00:00	A	0	0	1	8.20	8.335	32	30	
177	2011-01-08 16:00:00	A	0	0	1	7.38	6.820	29	30	
178	2011-01-08 17:00:00	A	0	0	1	6.56	6.060	37	36	
...	...	...	...	...	...	...	...	...	...	...
10682	2012-12-11 12:00:00	D	0	1	2	14.76	15.910	53	30	
10852	2012-12-18 14:00:00	D	0	1	1	18.86	22.725	47	30	
10853	2012-12-18 15:00:00	D	0	1	1	18.86	22.725	44	32	
10854	2012-12-18 16:00:00	D	0	1	1	18.04	21.970	41	31	
10855	2012-12-18 17:00:00	D	0	1	1	16.40	20.455	47	30	

427 rows × 12 columns

*min/max*값으로 보정

In [20]:
```
df.loc[(df[tmp] < min_iqr ),tmp] = min_iqr # 이상치 보정 - 하한치로 보정
df.loc[(df[tmp] > max_iqr ),tmp] = max_iqr # 이상치 보정 - 상한치로 보정
```

이상치 제거

In [21]:
```
df = df.drop(outlier_index, axis=0)
df.shape
```

Out[21]: (10459, 12)

요약데이터로 변환

In [22]:
```
df.groupby('season').aggregate({'datetime':'count','temp': 'min', 'windspeed':
np.mean, 'count': np.sum})
```

Out[22]:

season	datetime	temp	windspeed	count
A	2480	0.82	12.610484	285485
B	2618	9.84	12.132544	559432
C	2686	15.58	10.737528	625673
D	2675	5.74	10.885607	528522

파생변수 생성 (파머 책 p.452 참고)

In [24]:
```
# Recency
today = pd.to_datetime('2020-12-13') # 아니면 그냥 각 ID별로 최대값을 today에서 빼기
= ID별 Recency가 같음
cond1 = (today-df['datetime']) >= pd.Timedelta('3000 days')
cond2 = ((today-df['datetime']) < pd.Timedelta('3000 days'))&((today-df['datet
ime']) >= pd.Timedelta('2000 days'))
cond3 = (today-df['datetime']) < pd.Timedelta('2000 days')

df.loc[cond1, 'Recency'] = 1
df.loc[cond2, 'Recency'] = 2
df.loc[cond3, 'Recency'] = 3
```

In [25]:
```
# Frequency // 아니면 발생 count
df.loc[df['count'] <= 10, 'Frequency'] = 1
df.loc[(df['count'] > 10)&(df['count']<=20), 'Frequency'] = 2
df.loc[df['count'] >20, 'Frequency'] = 3
```

In [26]:
```
# Monetary // 아니면 발생 sum
df['Monetary'] = df['count'] * df['temp']
```

In [27]:
```
df['year'] = df['datetime'].map(lambda x: x.year)
df['month'] = df['datetime'].map(lambda x: x.month)
df['day'] = df['datetime'].map(lambda x: x.day)
df['hour'] = df['datetime'].map(lambda x: x.hour)
df['minute'] = df['datetime'].map(lambda x: x.minute)
```

In [28]:
```
df.head(3)
```

Out[28]:

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

데이터 마트 DQ Check, 변수선택 및 EDA

DQ Check

In [29]:
```
df.columns
```

Out[29]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count',
       'Recency', 'Frequency', 'Monetary', 'year', 'month', 'day', 'hour',
       'minute'],
      dtype='object')
```

In [30]:
```
col_num = ['weather', 'temp', 'atemp', 'humidity', 'windspeed', 'casual', 'reg
istered', 'count', 'Monetary']
col_cat = ['season', 'holiday', 'workingday', 'Recency', 'Frequency']
```

In [31]:
```
DA3 = DA(df[col_num])
DA3
```

Out[31]:

	total	count	missing	mean	median	std	variance	skewness	kurtos
weather	10459.0	10459.0	0	1.42	1.00	0.63	0.40	1.24	0.
temp	10459.0	10459.0	0	20.30	20.50	7.79	60.62	-0.00	-0.
atemp	10459.0	10459.0	0	23.76	24.24	8.42	70.89	-0.09	-0.
humidity	10459.0	10459.0	0	62.47	62.00	19.01	361.46	-0.10	-0.
windspeed	10459.0	10459.0	0	11.57	11.00	6.92	47.89	0.14	-0.
casual	10459.0	10459.0	0	35.90	17.00	49.89	2488.45	2.50	7.
registered	10459.0	10459.0	0	155.24	118.00	151.29	22885.98	1.53	2.
count	10459.0	10459.0	0	191.14	144.00	181.35	32884.71	1.24	1.
Monetary	10459.0	10459.0	0	4428.92	2617.44	5022.60	25224070.35	1.66	2.

In [32]:
```
DA4 = DA_cat(df, col_cat)
DA4
```

Out[32]:

	col_nm	class	count	ratio
0	season	A	2480	0.24
1	season	B	2618	0.25
2	season	C	2686	0.26
3	season	D	2675	0.26
4	holiday	0	10164	0.97
5	holiday	1	295	0.03
6	workingday	0	3342	0.32
7	workingday	1	7117	0.68
8	Recency	1	9119	0.87
9	Recency	2	1340	0.13
10	Frequency	1	1196	0.11
11	Frequency	2	616	0.06
12	Frequency	3	8647	0.83

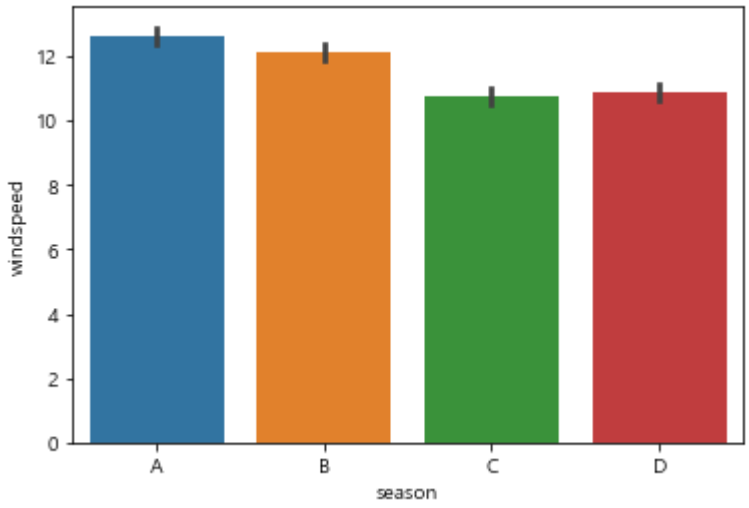
변수 제외

In [33]:
```
df = df.drop(columns = ['Frequency'], axis=1)
```

EDA

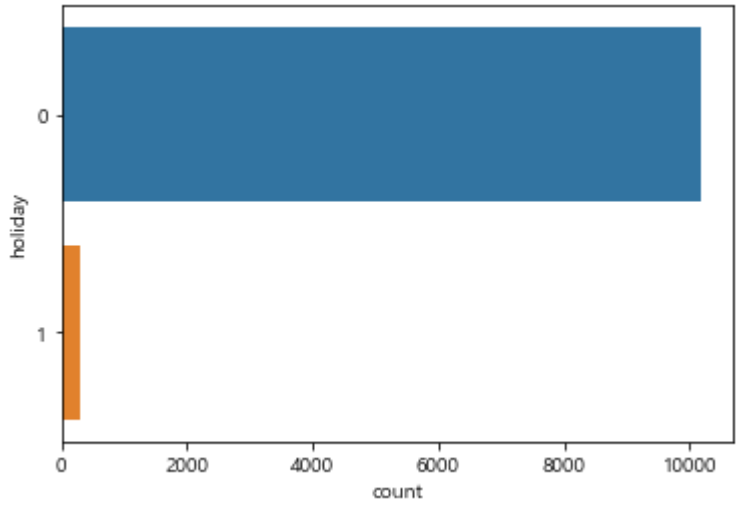
In [34]:
```
# 범주형 X별 y의 평균
sns.barplot(x ='season', y = 'windspeed', data = df)
```

Out[34]: <AxesSubplot:xlabel='season', ylabel='windspeed'>



In [35]:
```
# 범주형(또는 가지수가 많지 않은 연속형) 변수의 데이터별 count
sns.countplot(y = 'holiday', data = df)
```

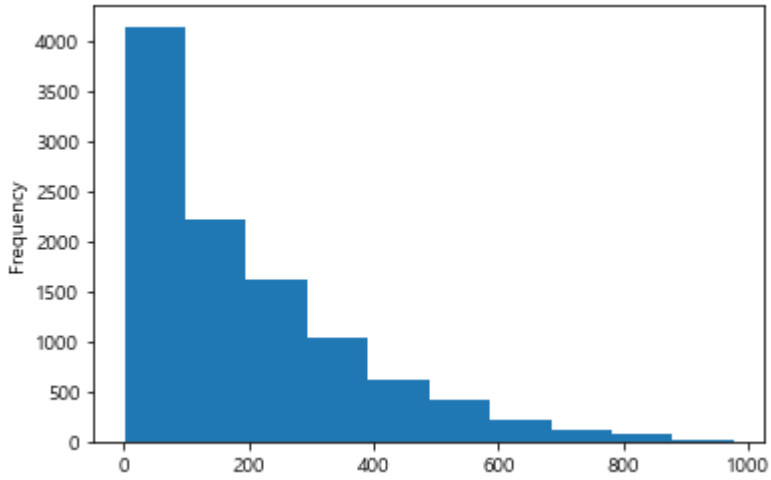
Out[35]: <AxesSubplot:xlabel='count', ylabel='holiday'>



종속변수 분포 확인 및 전처리

In [36]:
```
df['count'].plot(kind='hist')
```

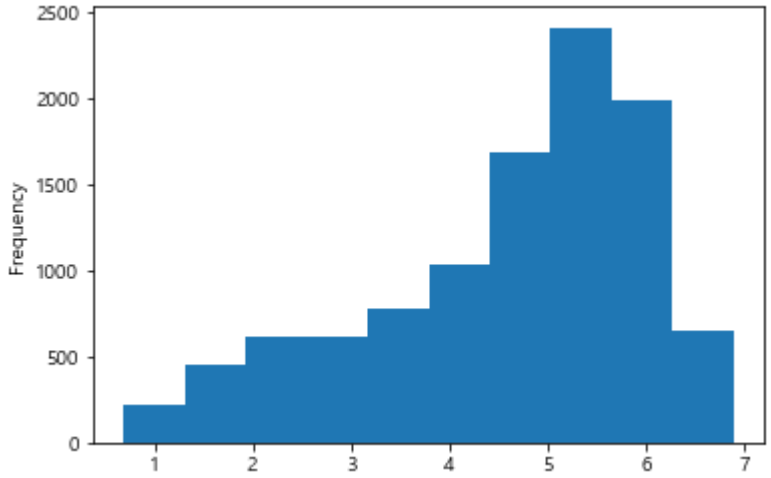
Out[36]: <AxesSubplot:ylabel='Frequency'>



In [37]:
```
df['count'] = np.log1p(df['count']) # inverse 는 np.expm1()
```

In [38]:
```
df['count'].plot(kind='hist')
```

Out[38]: <AxesSubplot:ylabel='Frequency'>



범주형 변수 더미화

In [39]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10459 entries, 0 to 10885
Data columns (total 19 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10459 non-null  datetime64[ns]
 1   season      10459 non-null  object
 2   holiday     10459 non-null  int64
 3   workingday  10459 non-null  int64
 4   weather     10459 non-null  int32
 5   temp        10459 non-null  float64
 6   atemp       10459 non-null  float64
 7   humidity    10459 non-null  int32
 8   windspeed   10459 non-null  float64
 9   casual      10459 non-null  int32
 10  registered  10459 non-null  int32
 11  count       10459 non-null  float64
 12  Recency     10459 non-null  float64
 13  Monetary    10459 non-null  float64
 14  year        10459 non-null  int64
 15  month       10459 non-null  int64
 16  day         10459 non-null  int64
 17  hour        10459 non-null  int64
 18  minute      10459 non-null  int64
dtypes: datetime64[ns](1), float64(6), int32(4), int64(7), object(1)
memory usage: 1.7+ MB
```

In [40]:
```
df.head(3)
```

Out[40]:

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

In [41]:
```
import statsmodels.api as sm
from patsy import dmatrices

y, X = dmatrices('count ~ season+ holiday+ workingday+ weather+ temp+atemp+ humidity+ windspeed+\
casual+ registered+ Recency+ Monetary+ year+ month+ day+ hour+ minute', data=df, return_type='dataframe')
```

VIF 확인 필요 (y값 섞여들어가지 않게 주의!!)

In [42]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif['VIF Factor'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['features'] = X.columns
vif
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1685: RuntimeWarning: invalid value encountered in double_scalars
  return 1 - self.ssr/self.centered_tss
```

Out[42]:

	VIF Factor	features
0	2.357161e+07	Intercept
1	4.984052e+00	season[T.B]
2	1.414252e+01	season[T.C]
3	2.612480e+01	season[T.D]
4	1.088440e+00	holiday
5	1.431576e+00	workingday
6	1.250423e+00	weather
7	4.605807e+01	temp
8	3.972747e+01	atemp
9	1.698243e+00	humidity
10	1.177114e+00	windspeed
11	3.924050e+00	casual
12	9.823384e+00	registered
13	2.382095e+00	Recency
14	1.796105e+01	Monetary
15	1.457619e+00	year
16	1.819586e+01	month
17	1.004100e+00	day
18	1.291457e+00	hour
19	NaN	minute

In [43]:
```
X = X.drop(columns = ['temp'])
```

In [44]:
```
vif = pd.DataFrame()
vif['VIF Factor'] = [variance_inflation_factor(X.values, i) for i in range(X.s
hape[1])]
vif['features'] = X.columns
vif
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmo
dels\regression\linear_model.py:1685: RuntimeWarning: invalid value encounter
ed in double_scalars
  return 1 - self.ssr/self.centered_tss
```

Out[44]:

	VIF Factor	features
0	2.353896e+07	Intercept
1	4.927431e+00	season[T.B]
2	1.384945e+01	season[T.C]
3	2.612426e+01	season[T.D]
4	1.087280e+00	holiday
5	1.425810e+00	workingday
6	1.247552e+00	weather
7	3.858072e+00	atemp
8	1.689671e+00	humidity
9	1.130916e+00	windspeed
10	3.882449e+00	casual
11	9.345486e+00	registered
12	2.381904e+00	Recency
13	1.690821e+01	Monetary
14	1.455611e+00	year
15	1.817768e+01	month
16	1.003365e+00	day
17	1.290558e+00	hour
18	NaN	minute

train, test split

In [45]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
m_state=0)
```

StandardScaler

In [46]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

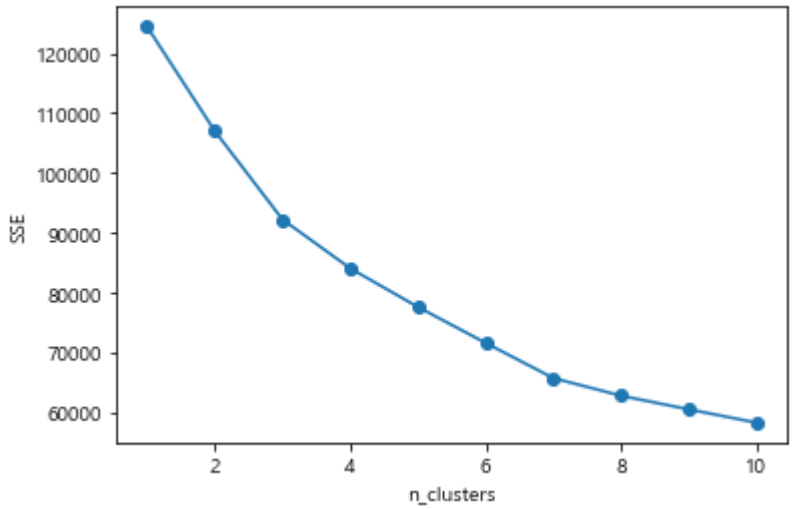
In [47]:
```
scaler.fit(X_train)
X_train_scale = scaler.transform(X_train)
X_test_scale = scaler.transform(X_test)
```

In [48]:
```
# 컬럼명 다시 붙여주기
X_train_scale = pd.DataFrame(X_train_scale, columns = X_train.columns)
X_test_scale = pd.DataFrame(X_test_scale, columns = X_test.columns)
```

군집화 수행

In [49]:
```
# X_train_scale, X_test_scale, y_train, y_test가 현재 변수

from sklearn.cluster import KMeans

def elbow(X):
    sse = []
    for i in range(1, 11) :
        km = KMeans(n_clusters=i, init ='k-means++', random_state = 0)
        km.fit(X)
        sse.append(km.inertia_)

    plt.plot(range(1, 11), sse, marker='o')
    plt.xlabel('n_clusters')
    plt.ylabel('SSE')
    plt.show()
```

In [50]:
```
elbow(X_train_scale)
```



In [51]:
```
from sklearn.metrics import silhouette_samples, silhouette_score

def sil(X):
    si = [] # 실루엣계수
    for i in range(2,11): # cluster가 2개인것 부터 10개까지!!!!
        km = KMeans(n_clusters=i, init='k-means++', random_state=0)
        km.fit(X)
        si.append(silhouette_score(X, km.labels_))
    print(np.round(si,3))
sil(X_train_scale)
```

```
[0.148 0.176 0.178 0.188 0.191 0.209 0.201 0.198 0.187]
```

군집 수 직접 지정해서 군집화

In [52]:
```
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300,random_state=0)
kmeans.fit(X_train_scale)
```

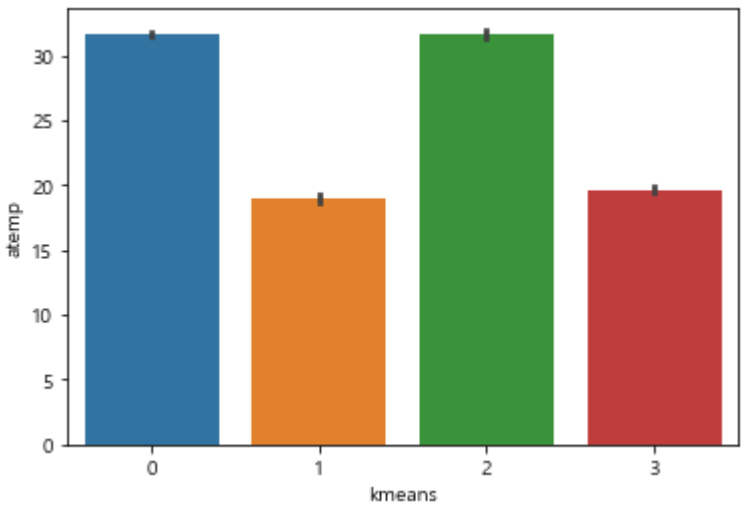
Out[52]:
```
KMeans(n_clusters=4, random_state=0)
```

군집화 결과 프로파일링

In [53]:
```
# 스케일링 풀고 프로파일링

df_profile = pd.DataFrame(scaler.inverse_transform(X_train_scale), columns = X
_train.columns)
df_profile['kmeans'] = kmeans.labels_
```

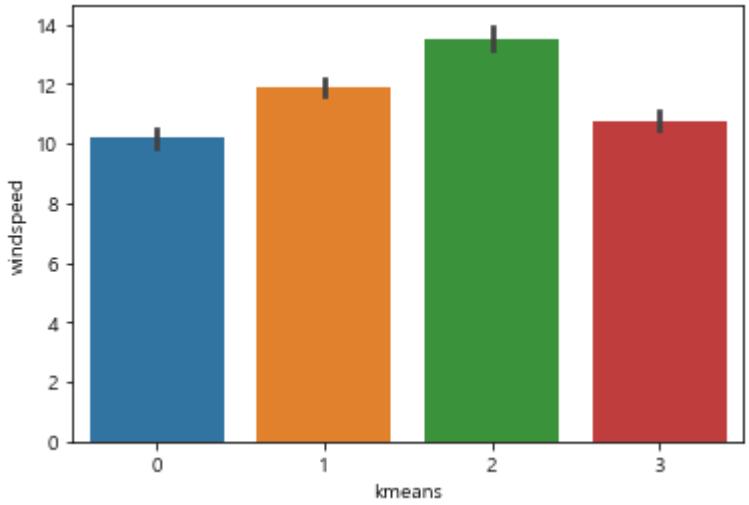
In [54]:
```
sns.barplot(df_profile['kmeans'], df_profile['atemp'])
```

Out[54]:
```
<AxesSubplot:xlabel='kmeans', ylabel='atemp'>
```



In [55]:
```
sns.barplot(df_profile['kmeans'], df_profile['windspeed'])
```

Out[55]: <AxesSubplot:xlabel='kmeans', ylabel='windspeed'>



군집화 결과를 새로운 컬럼으로 추가(train, test 모두 수행)

In [56]:
```
X_train_scale['kmeans'] = kmeans.labels_
```

In [57]:
```
kmeans_test = kmeans.predict(X_test_scale)
X_test_scale['kmeans'] = kmeans_test
```

모델링

In [58]:
```
from sklearn.linear_model import Ridge, Lasso, ElasticNet, HuberRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR

from sklearn.model_selection import GridSearchCV, train_test_split, KFold, cro
ss_val_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

In [59]:
```
models = []
models.append(('Ridge', Ridge()))
models.append(('Lasso', Lasso()))
models.append(('ElasticNet', ElasticNet()))
models.append(('Huber', HuberRegressor()))
models.append(('DT', DecisionTreeRegressor()))
models.append(('RF', RandomForestRegressor()))
models.append(('KNN', KNeighborsRegressor()))
models.append(('KernelRidge', KernelRidge()))
models.append(('MLP', MLPRegressor()))
models.append(('SVR', SVR()))
```

In [60]:
```
models
```

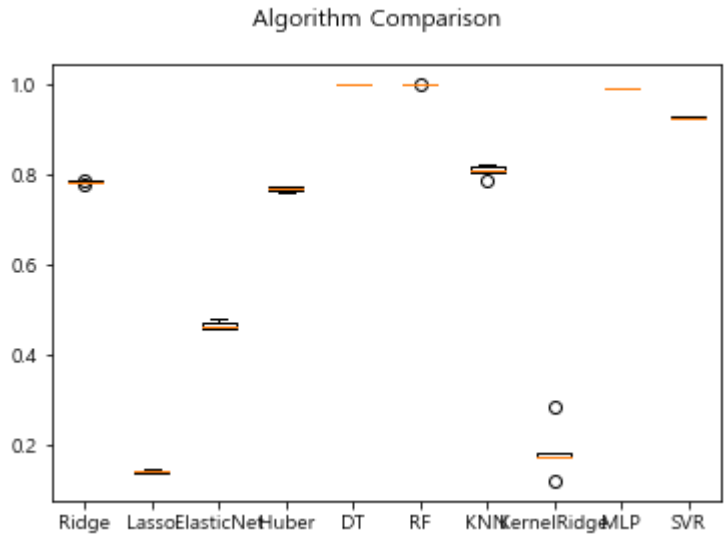
Out[60]:
```
[('Ridge', Ridge()),
 ('Lasso', Lasso()),
 ('ElasticNet', ElasticNet()),
 ('Huber', HuberRegressor()),
 ('DT', DecisionTreeRegressor()),
 ('RF', RandomForestRegressor()),
 ('KNN', KNeighborsRegressor()),
 ('KernelRidge', KernelRidge()),
 ('MLP', MLPRegressor()),
 ('SVR', SVR())]
```

In [61]:
```
num_folds = 5
seed = 7
```

In [63]:
```
names = []
results = []

kfold = KFold(n_splits = num_folds, shuffle = True, random_state=seed)

for name, model in models:
    score = cross_val_score(model, X_train_scale, y_train.values.ravel(), cv = 
kfold)
    names.append(name)
    results.append(score)
    print(name, score.mean().round(3))
```

```
Ridge 0.784
Lasso 0.141
ElasticNet 0.465
Huber 0.769
DT 0.999
RF 1.0
KNN 0.808
KernelRidge 0.186

C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\neural_network\_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\neural_network\_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\neural_network\_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\neural_network\_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\neural_network\_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn't conve
rged yet.
  % self.max_iter, ConvergenceWarning)

MLP 0.993
SVR 0.927
```

In [64]:
```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



========== 여기는 GridSearchCV 참고 내용

In [ ]:
```
models = []
params = []
```

In [ ]:
```
model = ('Ridge', Ridge())
param = {
    'alpha': [0.1, 0.3, 0.5, 1.0, 3.0, 5.0, 10.0]
}

models.append(model)
params.append(param)
```

In [ ]:
```
model = ('Lasso', Lasso())
param = {
    'alpha': [0.1, 0.3, 0.5, 1.0, 3.0, 5.0, 10.0]
}

models.append(model)
params.append(param)
```

In [ ]:
```
model = ('ElasticNet', ElasticNet())
param = {
    'alpha': [0.1, 0.3, 0.5, 1.0, 3.0, 5.0, 10.0],
    'l1_ratio': [0.3, 0.5, 0.7]
}

models.append(model)
params.append(param)
```

In [ ]:
```
model = ('HuberReg', HuberRegressor())
param = {
    'alpha': [0.0001, 0.001, 0.01]
}

models.append(model)
params.append(param)
```

In [ ]:
```
model = ('CART', DecisionTreeRegressor())
param = {
    'max_depth': [2, 3, 4, 5],
    'min_samples_split': [0.02, 0.05]
}

models.append(model)
params.append(param)
```

In [ ]:
```
model = ('RandomForest', RandomForestRegressor())
param = {
    'n_estimators': [50, 60, 70, 80, 90, 100],
    'max_features': [6, 7, 8, 9, 10]
}

models.append(model)
params.append(param)
```

In [ ]:
```
model = ('KNN', KNeighborsRegressor())
param = {
    'KNN__n_neighbors': [5, 10, 15, 20, 25, 30],
    'KNN__weights': ['uniform', 'distance']
}

models.append(model)
params.append(param)
```

In [ ]:
```
model = ('KernelRidge', KernelRidge())
param = [
    {'kernel': ['linear'], 'alpha': [0.01, 0.05, 0.1, 0.5, 1.0]},
    {'kernel': ['rbf'], 'alpha': [0.01, 0.05, 0.1, 0.5, 1.0], 'gamma': [0.01,
0.05, 0.1, 0.5, 1.0, 5.0, 10.0]}
]

models.append(model)
params.append(param)
```

In [ ]:
```
model = ('MLP', MLPRegressor())
param = {
    'hidden_layer_sizes': [(50, ), (100, ), (50, 50), (100, 100)],
    'solver': ['lbfgs'],
    'alpha': [0.0001, 0.001, 0.005],
    'max_iter': [200, 300, 400]
}

models.append(model)
params.append(param)
```

In [ ]:
```
model = ('SVR', SVR())
param = [
    {'kernel': ['linear'], 'C': [1.0, 10.0, 50.0, 100.0]},
    {'kernel': ['rbf'], 'C': [1.0, 10.0, 50.0, 100.0], 'gamma': [0.01, 0.05,
0.1, 0.5, 1.0]},
    {'kernel': ['poly'], 'C': [1.0, 10.0, 50.0, 100.0], 'degree': [3, 4, 5]}
]

models.append(model)
params.append(param)
```

파라미터 튜닝 및 최종 모델 선정

In [65]:
```
model = RandomForestRegressor()

n_estimators_set = [50, 60, 70, 80, 90, 100]
max_features_set = [6, 7, 8, 9, 10]
param_grid = dict(n_estimators = n_estimators_set,
                  max_features = max_features_set)

grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=kfold)
grid_result = grid.fit(X_train_scale, y_train.values.ravel())
print('Best : %f using %s' % (grid_result.best_score_, grid_result.best_params
_))

a = grid_result.cv_results_

for i in range(len(a['rank_test_score'])):
    print('%f (%f) with: %r' %(a['mean_test_score'][i], a['std_test_score'][i
], a['params'][i]))

# for params, mean_score, scores in grid_result.cv_results_:  ## 얘 에러난다
#     print('%f (%f) with: %r' %(mean_test_score.mean(), std_test_score.mean
(), params))
```

```
Best : 0.999577 using {'max_features': 10, 'n_estimators': 80}
0.998899 (0.000211) with: {'max_features': 6, 'n_estimators': 50}
0.999006 (0.000133) with: {'max_features': 6, 'n_estimators': 60}
0.999004 (0.000152) with: {'max_features': 6, 'n_estimators': 70}
0.999037 (0.000151) with: {'max_features': 6, 'n_estimators': 80}
0.999041 (0.000138) with: {'max_features': 6, 'n_estimators': 90}
0.999033 (0.000161) with: {'max_features': 6, 'n_estimators': 100}
0.999202 (0.000089) with: {'max_features': 7, 'n_estimators': 50}
0.999215 (0.000105) with: {'max_features': 7, 'n_estimators': 60}
0.999236 (0.000107) with: {'max_features': 7, 'n_estimators': 70}
0.999275 (0.000096) with: {'max_features': 7, 'n_estimators': 80}
0.999261 (0.000096) with: {'max_features': 7, 'n_estimators': 90}
0.999285 (0.000088) with: {'max_features': 7, 'n_estimators': 100}
0.999332 (0.000089) with: {'max_features': 8, 'n_estimators': 50}
0.999377 (0.000052) with: {'max_features': 8, 'n_estimators': 60}
0.999384 (0.000076) with: {'max_features': 8, 'n_estimators': 70}
0.999391 (0.000055) with: {'max_features': 8, 'n_estimators': 80}
0.999432 (0.000075) with: {'max_features': 8, 'n_estimators': 90}
0.999401 (0.000089) with: {'max_features': 8, 'n_estimators': 100}
0.999472 (0.000087) with: {'max_features': 9, 'n_estimators': 50}
0.999487 (0.000048) with: {'max_features': 9, 'n_estimators': 60}
0.999502 (0.000056) with: {'max_features': 9, 'n_estimators': 70}
0.999507 (0.000052) with: {'max_features': 9, 'n_estimators': 80}
0.999513 (0.000077) with: {'max_features': 9, 'n_estimators': 90}
0.999515 (0.000054) with: {'max_features': 9, 'n_estimators': 100}
0.999528 (0.000051) with: {'max_features': 10, 'n_estimators': 50}
0.999560 (0.000052) with: {'max_features': 10, 'n_estimators': 60}
0.999558 (0.000058) with: {'max_features': 10, 'n_estimators': 70}
0.999577 (0.000061) with: {'max_features': 10, 'n_estimators': 80}
0.999569 (0.000070) with: {'max_features': 10, 'n_estimators': 90}
0.999569 (0.000063) with: {'max_features': 10, 'n_estimators': 100}
```

In [66]:
```
fine_tuned_RF = grid_result.best_estimator_
print('best params: ', grid_result.best_params_)
fine_tuned_RF.feature_importances_
```

```
best params:  {'max_features': 10, 'n_estimators': 80}
```

Out[66]:
```
array([0.00000000e+00, 6.00736488e-05, 1.24486569e-04, 7.39703667e-05,
       1.68125859e-05, 3.73768980e-04, 5.87169085e-05, 4.40508303e-03,
       4.80124268e-04, 1.62852854e-04, 4.17968046e-02, 5.33920391e-01,
       4.32636020e-05, 2.87708776e-01, 1.59035414e-04, 1.14429896e-03,
       1.92752420e-04, 1.20272922e-01, 0.00000000e+00, 9.00586699e-03])
```

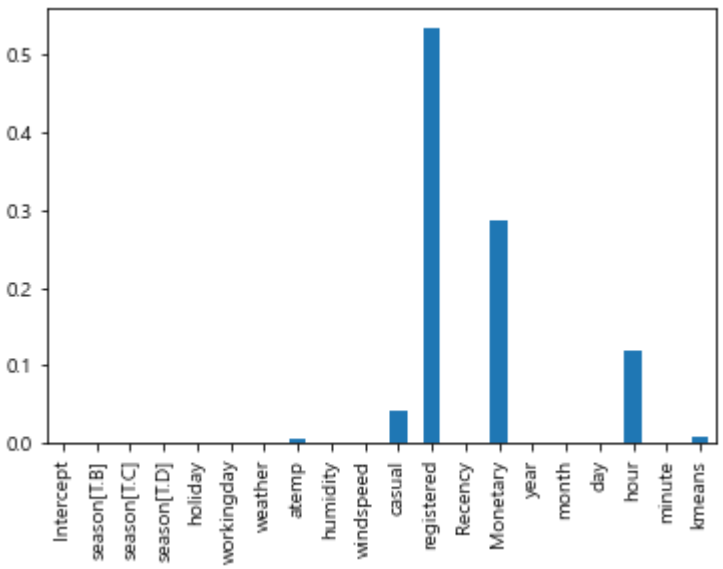
In [67]:
```
pd.DataFrame({'col':X_train_scale.columns, 'FI':fine_tuned_RF.feature_importances_}).sort_values('FI', ascending=False)
```

Out[67]:

	col	FI
11	registered	0.533920
13	Monetary	0.287709
17	hour	0.120273
10	casual	0.041797
19	kmeans	0.009006
7	atemp	0.004405
15	month	0.001144
8	humidity	0.000480
5	workingday	0.000374
16	day	0.000193
9	windspeed	0.000163
14	year	0.000159
2	season[T.C]	0.000124
3	season[T.D]	0.000074
1	season[T.B]	0.000060
6	weather	0.000059
12	Recency	0.000043
4	holiday	0.000017
18	minute	0.000000
0	Intercept	0.000000

In [68]:
```
importances = pd.Series(fine_tuned_RF.feature_importances_, index=X_train_scale.columns)
importances.plot(kind='bar')
```

Out[68]: <AxesSubplot:>



Test set 활용하여 예측 수행

In [69]:
```
y_pred = fine_tuned_RF.predict(X_test_scale)
```

In [70]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

R^2

In [71]:
```
r2 = r2_score(np.expm1(y_test), np.expm1(y_pred)) # 종속변수에 log 처리를 했을 경우 expm1을 해줘야함, 아니면 그냥 y_test, y_pred
r2
```

Out[71]: 0.9989086476374622

MSE

In [72]:
```
mse = mean_squared_error(np.expm1(y_test), np.expm1(y_pred))
# 종속변수에 log 처리를 했을 경우 expm1을 해줘야함, 아니면 그냥 y_test, y_pred
mse
```

Out[72]: 37.043451427771124

RMSE

In [73]:
```
rmse = np.sqrt(mse)
rmse
```

Out[73]: 6.086333167661061

MAE

In [74]:
```
mae = mean_absolute_error(np.expm1(y_test), np.expm1(y_pred))
 # 종속변수에 log 처리를 했을 경우 expm1을 해줘야함, 아니면 그냥 y_test, y_pred
mae
```

Out[74]: 2.8376408402440068

MAPE

In [75]:
```
def mp(y_test, y_pred):
    y_test, y_pred = np.array(y_test), np.array(y_pred)
    return np.mean(np.abs(y_test - y_pred)/y_test) * 100
# 평균 절대 백분율 오차(MAPE)는 정확도를 오차의 백분율로 표시합니다.
# MAPE는 백분율이기 때문에 다른 정확도 측도 통계량보다 더 쉽게 이해할 수 있습니다.
# 예를 들어 MAPE가 5이면 예측 값은 평균 5% 벗어납니다
```

In [76]:
```
mape = mp(np.expm1(y_test), np.expm1(y_pred))
 # 종속변수에 log 처리를 했을 경우 expm1을 해줘야함, 아니면 그냥 y_test, y_pred
mape
```

Out[76]: 776.490280981951