기계학습

분류

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() #CCCCC
```

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()  #CCCCC
```

Out[5]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	ca
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]:
```
# type별로 컬럼 분류
col_id = []
col_dt = ['datetime']
col_cat = ['season']
col_int = ['weather', 'humidity','casual','registered','count']
col_float = ['temp','atemp','windspeed']
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]:

```python
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() # 결측값
    df1.name = 'missing'
    df2 = data.median() # 중앙값
    df2.name = 'median'
    df3 = np.var(data) # 분산
    df3.name = 'variance'
    df4 = data.skew() # 왜도 : 양수면 왼쪽으로 치우침
    df4.name = 'skewness'
    df5 = data.kurtosis() # 첨도 : 0보다 클수록 뾰족함
    df5.name = 'kurtosis'

    da = pd.concat([da,df1,df2,df3,df4,df5], axis=1) # 모두 합침
    da['total'] = da['count'] + da['missing'] # 전체 데이터 수

    # 컬럼 순서 보기 좋게 정렬
    col_nm = da.columns.tolist()
    order = ['total','count','missing','mean','median','std','variance','skewness','kurtosis','min',
             '5%','10%','25%','50%','75%','90%','95%','max']
    col_nm_new=[]
    for i in order:
        col_nm_new.append(i)
#     col_nm_new.extend(col_nm[3:12])
    da = da[col_nm_new]

    # 소수점 둘째자리 반올림
    da = da.round(2)
    return da
```

In [12]:

```python
DA1 = DA(df[col_num])
DA1.to_csv('빈도분포분석_연속형.csv', encoding='cp949')
```

In [13]:

```python
DA1
```

Out[13]:

	total	count	missing	mean	median	std	variance	skewness	kurtosis	n
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.
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.
windspeed	10886.0	10886.0	0	12.80	13.00	8.16	66.65	0.59	0.63	0.

범주형 변수

In [14]:
```
# 범주형 변수 빈도분석
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 [15]:
```
DA2 = DA_cat(df,col_cat+col_bool)
# DA2.to_csv('빈도분포분석_범주형.csv', encoding='cp949', errors='ignore')
DA2
```

Out[15]:

	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 [16]:

```
# 중복값 확인
df[df.duplicated(keep=False)]
```

Out[16]:

datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual

In [ ]:

```
df.drop_duplicates() # 행 내용이 동일한 경우 제거
df.drop_duplicates(['col1'], keep='last') # col1기준 중복값중 마지막만 남겨둠
```

결측치

In [17]:

```
df.isna().sum()
```

Out[17]:

```
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 [ ]:

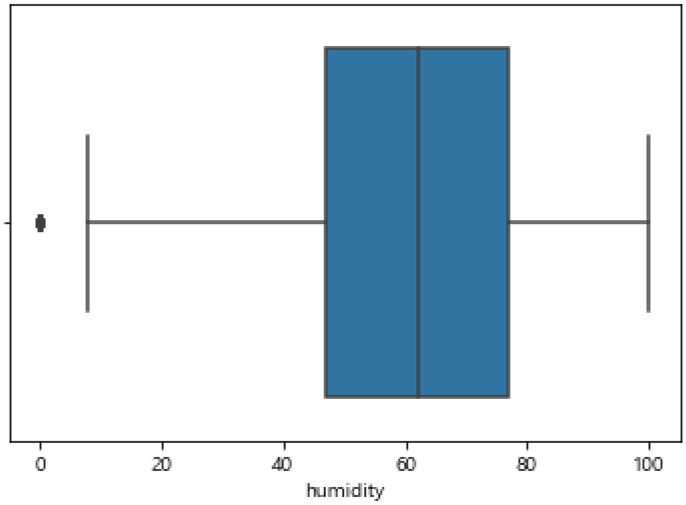
```
# na 처리 : dropna(), fillna()
df.dropna() # nan이 하나라도 들어간 행은 삭제
df.dropna(how = 'all') # 데이터가 모두 nan인 행만 삭제 / 초기값:'any'
## Parameters
# axis = 'index' / 'columns'
# subset = ['col1', 'col2', ...] # 적용 대상 컬럼 특정

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 [18]:
```
tmp = 'humidity'
```

In [19]:
```
sns.boxplot(y = tmp, data = df, orient = 'h')
```

Out[19]:
```
<AxesSubplot:xlabel='humidity'>
```



In [20]:

```
# 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
```

```
(22, 12)
```

Out[20]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	cas
1091	2011-03-10 00:00:00	A	0	1	3	13.94	15.910	0	16.9979	
1092	2011-03-10 01:00:00	A	0	1	3	13.94	15.910	0	16.9979	
1093	2011-03-10 02:00:00	A	0	1	3	13.94	15.910	0	16.9979	
1094	2011-03-10 05:00:00	A	0	1	3	14.76	17.425	0	12.9980	
1095	2011-03-10 06:00:00	A	0	1	3	14.76	16.665	0	22.0028	
1096	2011-03-10 07:00:00	A	0	1	3	15.58	19.695	0	15.0013	
1097	2011-03-10 08:00:00	A	0	1	3	15.58	19.695	0	19.0012	
1098	2011-03-10 09:00:00	A	0	1	3	16.40	20.455	0	15.0013	
1099	2011-03-10 10:00:00	A	0	1	3	16.40	20.455	0	11.0014	
1100	2011-03-10 11:00:00	A	0	1	3	16.40	20.455	0	16.9979	
1101	2011-03-10 12:00:00	A	0	1	3	17.22	21.210	0	15.0013	
1102	2011-03-10 13:00:00	A	0	1	3	17.22	21.210	0	15.0013	
1103	2011-03-10 14:00:00	A	0	1	3	18.04	21.970	0	19.9995	
1104	2011-03-10 15:00:00	A	0	1	3	18.04	21.970	0	15.0013	
1105	2011-03-10 16:00:00	A	0	1	3	17.22	21.210	0	16.9979	
1106	2011-03-10 17:00:00	A	0	1	2	18.04	21.970	0	26.0027	
1107	2011-03-10 18:00:00	A	0	1	3	18.04	21.970	0	23.9994	

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	cas
1108	2011-03-10 19:00:00	A	0	1	3	18.04	21.970	0	39.0007	
1109	2011-03-10 20:00:00	A	0	1	3	14.76	16.665	0	22.0028	
1110	2011-03-10 21:00:00	A	0	1	3	14.76	17.425	0	15.0013	
1111	2011-03-10 22:00:00	A	0	1	2	13.94	16.665	0	8.9981	
1112	2011-03-10 23:00:00	A	0	1	3	13.94	17.425	0	6.0032	

*min/max*값으로 보정

In [21]:
```
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 [22]:
```
df = df.drop(outlier_index, axis=0)
df.shape
```

Out[22]:
```
(10864, 12)
```

요약데이터로 변환

In [23]:
```
df.groupby('season').aggregate({'datetime':'count','temp': 'min', 'windspeed':
np.mean, 'count': np.sum})
```

Out[23]:

	datetime	temp	windspeed	count
season				
A	2664	0.82	14.612957	311875
B	2733	9.84	13.405607	588282
C	2733	15.58	11.508862	640662
D	2734	5.74	11.678147	544034

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['datetime']) >= 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]:
```
# Frenquency (빈도)  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 (거래규모) 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	0.0	3
1	2011-01-01 01:00:00	A	0	0	1	9.02	13.635	80.0	0.0	8
2	2011-01-01 02:00:00	A	0	0	1	9.02	13.635	80.0	0.0	5

데이터 마트 DQ Check, 변수선택및 EDA

DQ Check

In [29]:
```
col_num = ['weather', 'temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count', 'Monetary']
col_cat = ['season', 'holiday', 'workingday', 'Recency', 'Frequency']
```

In [30]:
```
DA3 = DA(df[col_num])
DA3
```

Out[30]:

	total	count	missing	mean	median	std	variance	skewness	kurtos
weather	10864.0	10864.0	0	1.42	1.00	0.63	0.40	1.25	0.
temp	10864.0	10864.0	0	20.24	20.50	7.80	60.78	0.00	-0.
atemp	10864.0	10864.0	0	23.66	24.24	8.48	71.91	-0.11	-0.
humidity	10864.0	10864.0	0	62.01	62.00	19.06	363.32	-0.04	-0.
windspeed	10864.0	10864.0	0	12.79	13.00	8.16	66.66	0.59	0.
casual	10864.0	10864.0	0	36.09	17.00	49.99	2498.53	2.49	7.
registered	10864.0	10864.0	0	155.81	119.00	151.08	22821.59	1.52	2.
count	10864.0	10864.0	0	191.90	146.00	181.17	32821.26	1.24	1.
Monetary	10864.0	10864.0	0	4440.41	2642.86	5024.94	25247714.04	1.66	2.

In [31]:
```
DA4 = DA_cat(df, col_cat)
DA4
```

Out[31]:

	col_nm	class	count	ratio
0	season	A	2664	0.25
1	season	B	2733	0.25
2	season	C	2733	0.25
3	season	D	2734	0.25
4	holiday	0	10553	0.97
5	holiday	1	311	0.03
6	workingday	0	3474	0.32
7	workingday	1	7390	0.68
8	Recency	1	9497	0.87
9	Recency	2	1367	0.13
10	Frequency	1	1225	0.11
11	Frequency	2	625	0.06
12	Frequency	3	9014	0.83

변수 제외

In [32]:

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

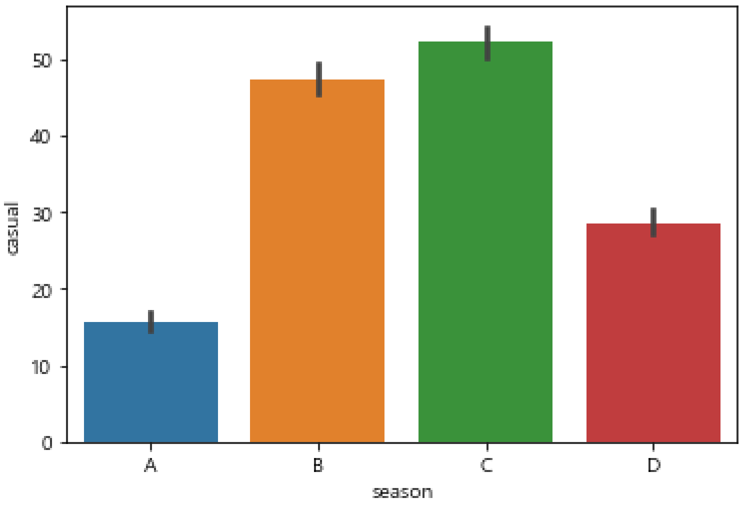
EDA

In [33]:

```
# 범주형 X별 y의 평균
sns.barplot(x='season', y='casual', data=df)
```

Out[33]:

```
<AxesSubplot:xlabel='season', ylabel='casual'>
```

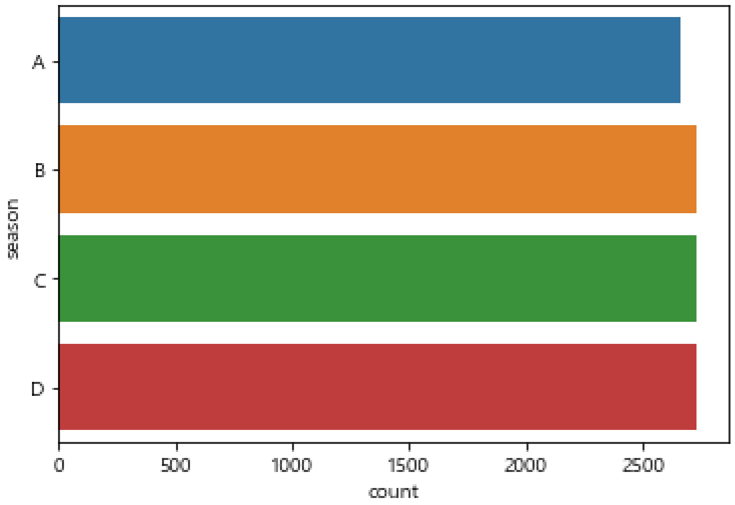


In [34]:

```
# 범주형(또는 가지수가 많지 않은 연속형) 변수의 데이터별 count
sns.countplot(y = 'season', data = df)
```

Out[34]:

```
<AxesSubplot:xlabel='count', ylabel='season'>
```



종속변수 전처리(이항 형태로 변환 / 4개 클래스로 변환)

In [35]:

```
# 이항 형태
df.loc[df['count'] <= 150, 'y1'] = 1
df.loc[df['count'] > 150, 'y1'] = 0
```

In [36]:

```
# 4개 클래스
df.loc[df['count'] <= 150, 'y2'] = 0
df.loc[(df['count']>150)&(df['count'] <= 300), 'y2'] = 1
df.loc[(df['count']>300)&(df['count'] <= 450), 'y2'] = 2
df.loc[df['count'] > 450, 'y2'] = 3
```

범주형 변수 더미화

In [37]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10864 entries, 0 to 10885
Data columns (total 21 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   datetime    10864 non-null  datetime64[ns]
 1   season      10864 non-null  object        
 2   holiday     10864 non-null  int64         
 3   workingday  10864 non-null  int64         
 4   weather     10864 non-null  int32         
 5   temp        10864 non-null  float64       
 6   atemp       10864 non-null  float64       
 7   humidity    10864 non-null  float64       
 8   windspeed   10864 non-null  float64       
 9   casual      10864 non-null  int32         
 10  registered  10864 non-null  int32         
 11  count       10864 non-null  int32         
 12  Recency     10864 non-null  float64       
 13  Monetary    10864 non-null  float64       
 14  year        10864 non-null  int64         
 15  month       10864 non-null  int64         
 16  day         10864 non-null  int64         
 17  hour        10864 non-null  int64         
 18  minute      10864 non-null  int64         
 19  y1          10864 non-null  float64       
 20  y2          10864 non-null  float64       
dtypes: datetime64[ns](1), float64(8), int32(4), int64(7), object(1)
memory usage: 2.0+ MB
```

In [38]:
```
df.head()
```

Out[38]:

	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	0.0	3
1	2011-01-01 01:00:00	A	0	0	1	9.02	13.635	80.0	0.0	8
2	2011-01-01 02:00:00	A	0	0	1	9.02	13.635	80.0	0.0	5
3	2011-01-01 03:00:00	A	0	0	1	9.84	14.395	75.0	0.0	3
4	2011-01-01 04:00:00	A	0	0	1	9.84	14.395	75.0	0.0	0

In [39]:
```
import statsmodels.api as sm
import pandas as pd
from patsy import dmatrices

y, X = dmatrices('y2 ~ season + holiday + workingday+weather+temp+atemp+humidi
ty+windspeed+casual\
+registered+Recency+Monetary', data=df, return_type='dataframe')
```

VIF 확인 필요 (y값 섞여 들어가지 않게 주의!!)

In [40]:
```
# y, X = dmatrices('price ~ area + bedrooms + bathrooms', df, return_type = 'dataframe')
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
```

Out[40]:

	VIF Factor	features
0	56.314408	Intercept
1	2.535383	season[T.B]
2	4.129738	season[T.C]
3	2.580344	season[T.D]
4	1.075556	holiday
5	1.428789	workingday
6	1.282391	weather
7	43.474120	temp
8	37.339966	atemp
9	1.743745	humidity
10	1.210113	windspeed
11	3.912857	casual
12	9.435817	registered
13	1.805088	Recency
14	17.914355	Monetary

In [41]:
```
X = X.drop(columns=['temp'])
```

In [42]:
```
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["features"] = X.columns
vif
```

Out[42]:

	VIF Factor	features
0	56.306543	Intercept
1	2.455552	season[T.B]
2	3.722530	season[T.C]
3	2.569020	season[T.D]
4	1.074492	holiday
5	1.423276	workingday
6	1.279043	weather
7	3.751375	atemp
8	1.731370	humidity
9	1.157455	windspeed
10	3.868163	casual
11	8.961080	registered
12	1.803690	Recency
13	16.802380	Monetary

train, test split

In [43]:
```
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, random_state=0)
```

StandardScaler

In [44]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

In [45]:
```
scaler.fit(X_train)
X_train_scale = scaler.transform(X_train)
X_test_scale = scaler.transform(X_test)
```

In [46]:
```
# 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)
```

오버샘플링 수행

In [47]:

```
y_train.value_counts()
```

Out[47]:

```
y2
0.0    3885
1.0    1986
2.0     971
3.0     762
dtype: int64
```

In [ ]:

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=0)
X_train_over, y_train_over = smote.fit_sample(X_train_scale, y_train)
```

In [49]:

```
# 컬럼명 다시 붙여주기
X_train_over = pd.DataFrame(X_train_over, columns= X_train.columns)
y_train_over = pd.DataFrame(y_train_over, columns=['y2'])
```

In [50]:

```
y_train_over.value_counts()
```

Out[50]:

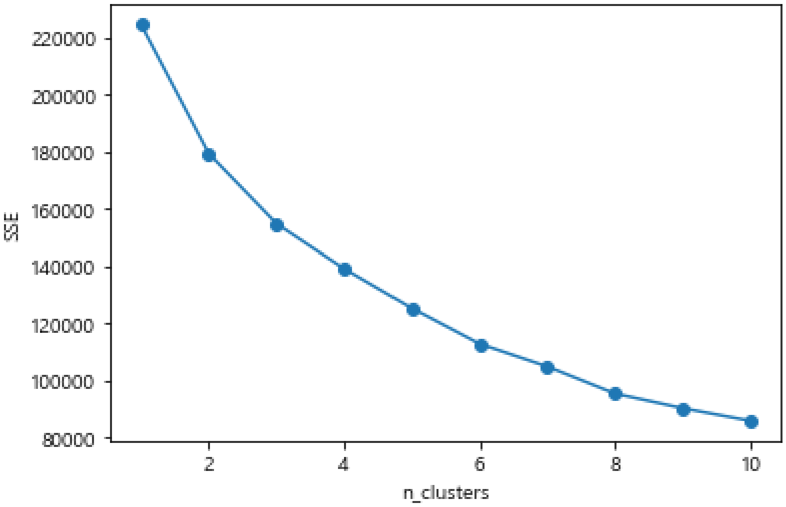
```
y2
3.0    3885
2.0    3885
1.0    3885
0.0    3885
dtype: int64
```

군집화 수행

In [51]:

```
# X_train_over, X_test_scale, y_train_over, y_test이 현재 변수

from sklearn.cluster import KMeans

# 바로 최적 개수 찾기
def elbow(X):
    sse = [] # 오차제곱합
    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 [52]:
```
elbow(X_train_over)
```



In [53]:
```
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_over)
```

```
[0.194 0.21  0.202 0.209 0.229 0.225 0.238 0.232 0.24 ]
```

군집 수 직접 지정해서 군집화

In [54]:
```
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300,random_state=0)
kmeans.fit(X_train_over)
```

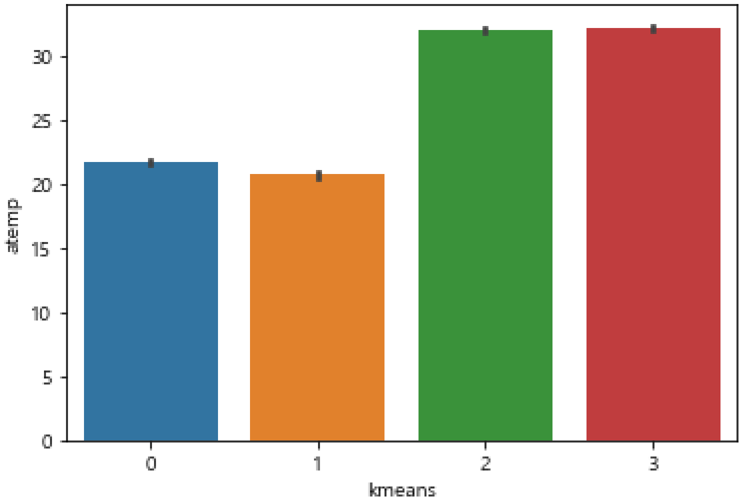
Out[54]:
```
KMeans(n_clusters=4, random_state=0)
```

군집화 결과 프로파일링

In [55]:
```
# 스케일링 풀고 프로파일링

df_profile = pd.DataFrame(scaler.inverse_transform(X_train_over), columns = X_
train.columns)
df_profile['kmeans'] = kmeans.labels_
```

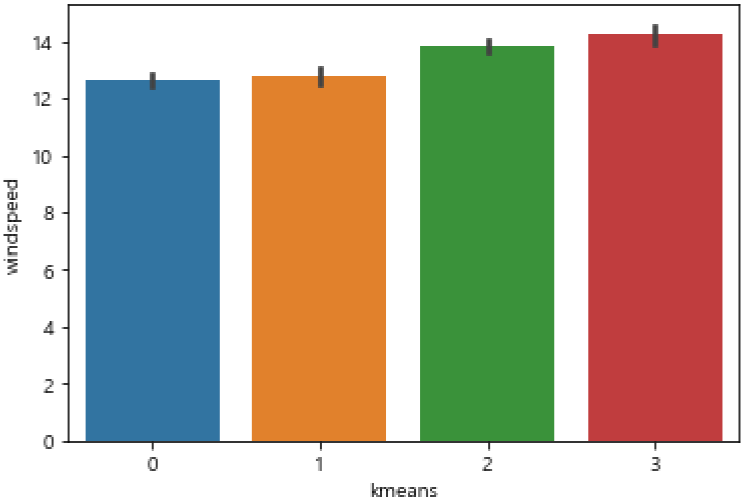
In [56]:
```
sns.barplot(df_profile['kmeans'], df_profile['atemp'])
```

Out[56]:
```
<AxesSubplot:xlabel='kmeans', ylabel='atemp'>
```



In [57]:
```
sns.barplot(df_profile['kmeans'], df_profile['windspeed'])
```

Out[57]:
```
<AxesSubplot:xlabel='kmeans', ylabel='windspeed'>
```



군집 결과 성능 평가

In [58]:
```
a = y_train_over['y2'].astype('int')
```

In [59]:
```
a.value_counts().sort_index()
```

Out[59]:
```
0    3885
1    3885
2    3885
3    3885
Name: y2, dtype: int64
```

In [60]:
```
b = df_profile['kmeans']
```

In [61]:
```
b.value_counts().sort_index()
```

Out[61]:
```
0    6853
1    2147
2    4103
3    2437
Name: kmeans, dtype: int64
```

In [62]:
```
# y의 class와 군집화 class의 이름을 맞추지 않으면 matrix가 안 맞는다
# 세로가 actual값, 가로가 예측값
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

target_names = ['class 0', 'class 1', 'class 2', 'class 3']
print(classification_report(a, b, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.51      0.90      0.65      3885
     class 1       0.22      0.12      0.16      3885
     class 2       0.33      0.35      0.34      3885
     class 3       0.49      0.31      0.38      3885

    accuracy                           0.42     15540
   macro avg       0.39      0.42      0.38     15540
weighted avg       0.39      0.42      0.38     15540
```

In [63]:
```
target_name_pred = ['예측_'+i for i in target_names]
target_name_actual = ['실제_'+i for i in target_names]
```

In [64]:
```
confusion = pd.DataFrame(confusion_matrix( a, b))
confusion.columns = target_name_pred
confusion.index = target_name_actual
confusion
```

Out[64]:

	예측_class 0	예측_class 1	예측_class 2	예측_class 3
실제_class 0	3510	356	19	0
실제_class 1	2522	477	750	136
실제_class 2	746	694	1348	1097
실제_class 3	75	620	1986	1204

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

In [65]:
```
X_train_over['kmeans'] = kmeans.labels_
```

In [66]:
```
kmeans_test = kmeans.predict(X_test_scale)
X_test_scale['kmeans'] = kmeans_test
```

모델링

In [67]:
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, train_test_split, KFold, cro
ss_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
```

In [68]:
```
# Evaluate Algorithms
# Test options and evaluation metric
num_folds = 10
seed = 7
scoring = 'accuracy'
# num_instances = len(X_train_over)
```

기초모델

In [69]:
```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('MLP', MLPClassifier()))
```

In [70]:
```
models
```

Out[70]:
```
[('LR', LogisticRegression()),
 ('LDA', LinearDiscriminantAnalysis()),
 ('KNN', KNeighborsClassifier()),
 ('CART', DecisionTreeClassifier()),
 ('NB', GaussianNB()),
 ('RF', RandomForestClassifier()),
 ('MLP', MLPClassifier())]
```

In [ ]:

```
results = []
names = []

from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=seed)
# kfold = KFold(n_splits=num_folds, shuffle=True, random_state=seed)

for name, model in models:
    cv_results = cross_val_score(model, X_train_over, y_train_over.values.rave
l(), cv=skf, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "[%s]\tmean: %f\tstd: %f" % (name, cv_results.mean(), cv_results.std
())
    print(msg)
```

In [72]:

```
names
```

Out[72]:

```
['LR', 'LDA', 'KNN', 'CART', 'NB', 'RF', 'MLP']
```

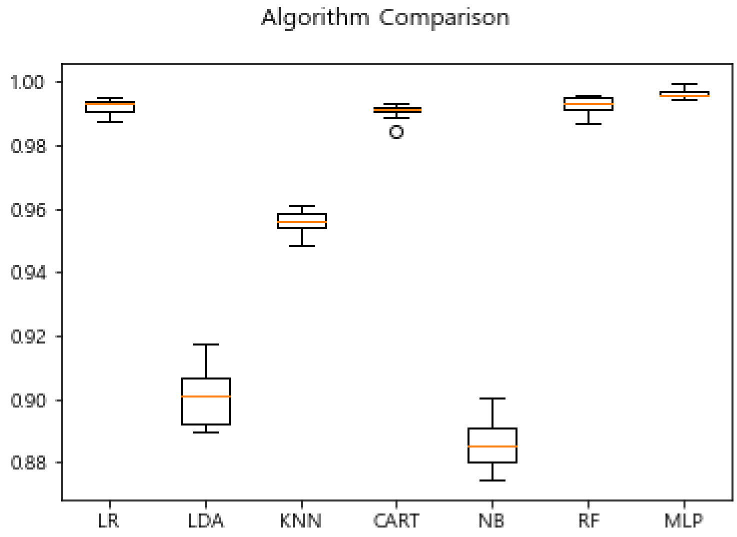
In [73]:

```
for i in range(len(names)):
    print(names[i], np.mean(results[i]))
```

```
LR 0.9923423423423424
LDA 0.9005791505791507
KNN 0.9560489060489061
CART 0.9905405405405405
NB 0.885971685971686
RF 0.9925353925353926
MLP 0.9962676962676962
```

In [74]:

```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



파라미터 튜닝 및 최종모델 선정

In [75]:

```
model = RandomForestClassifier()

n_estimators_set = [5, 10, 15, 20, 25, 30, 35, 40]
max_features_set = ["sqrt", "log2", None]
param_grid = dict(n_estimators = n_estimators_set,
                  max_features = max_features_set)

grid = GridSearchCV(estimator = model, param_grid = param_grid, scoring = scor
ing, cv = skf)
grid_result = grid.fit(X_train_over, y_train_over.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: 0.994916 using {'max_features': None, 'n_estimators': 40}
0.984363 (0.003490) with: {'max_features': 'sqrt', 'n_estimators': 5}
0.987967 (0.002116) with: {'max_features': 'sqrt', 'n_estimators': 10}
0.990991 (0.003166) with: {'max_features': 'sqrt', 'n_estimators': 15}
0.990541 (0.002339) with: {'max_features': 'sqrt', 'n_estimators': 20}
0.990734 (0.002852) with: {'max_features': 'sqrt', 'n_estimators': 25}
0.991570 (0.003092) with: {'max_features': 'sqrt', 'n_estimators': 30}
0.990798 (0.002776) with: {'max_features': 'sqrt', 'n_estimators': 35}
0.991120 (0.002571) with: {'max_features': 'sqrt', 'n_estimators': 40}
0.983655 (0.003958) with: {'max_features': 'log2', 'n_estimators': 5}
0.986937 (0.001844) with: {'max_features': 'log2', 'n_estimators': 10}
0.989897 (0.003663) with: {'max_features': 'log2', 'n_estimators': 15}
0.990347 (0.002775) with: {'max_features': 'log2', 'n_estimators': 20}
0.990798 (0.001844) with: {'max_features': 'log2', 'n_estimators': 25}
0.990862 (0.002757) with: {'max_features': 'log2', 'n_estimators': 30}
0.991120 (0.003043) with: {'max_features': 'log2', 'n_estimators': 35}
0.991248 (0.003194) with: {'max_features': 'log2', 'n_estimators': 40}
0.993372 (0.002591) with: {'max_features': None, 'n_estimators': 5}
0.994530 (0.001048) with: {'max_features': None, 'n_estimators': 10}
0.994273 (0.002483) with: {'max_features': None, 'n_estimators': 15}
0.994466 (0.001913) with: {'max_features': None, 'n_estimators': 20}
0.994402 (0.002356) with: {'max_features': None, 'n_estimators': 25}
0.994595 (0.001956) with: {'max_features': None, 'n_estimators': 30}
0.994144 (0.002737) with: {'max_features': None, 'n_estimators': 35}
0.994916 (0.001961) with: {'max_features': None, 'n_estimators': 40}
```

In [76]:

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

```
best params:  {'max_features': None, 'n_estimators': 40}
```

Out[76]:

```
array([0.00000000e+00, 1.65364715e-04, 1.26060039e-04, 2.78412675e-04,
       1.50860482e-04, 1.02359221e-04, 3.91093171e-04, 1.48941326e-02,
       2.23551451e-03, 9.95564511e-04, 2.34069370e-01, 6.49031516e-01,
       5.32592176e-05, 9.73313995e-02, 1.75093029e-04])
```

In [77]:

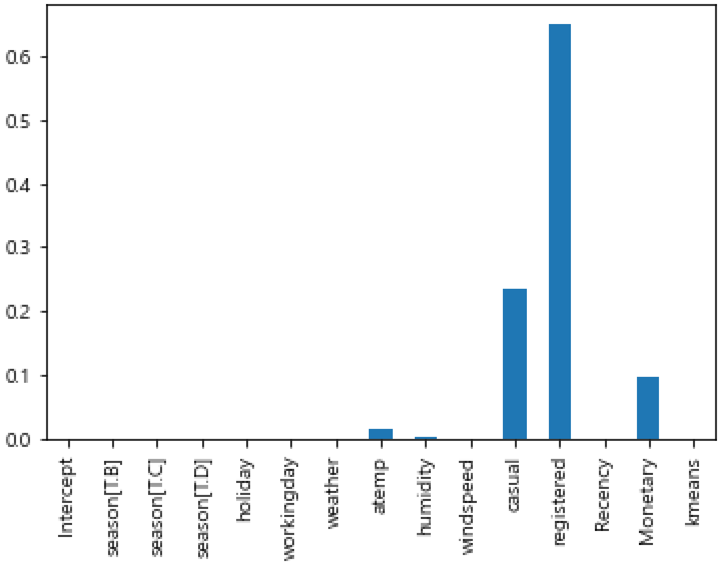
```
pd.DataFrame({'col':X_train_over.columns, 'FI':fine_tuned_RF.feature_importanc
es_}).sort_values('FI', ascending=False)
```

Out[77]:

	col	FI
11	registered	0.649032
10	casual	0.234069
13	Monetary	0.097331
7	atemp	0.014894
8	humidity	0.002236
9	windspeed	0.000996
6	weather	0.000391
3	season[T.D]	0.000278
14	kmeans	0.000175
1	season[T.B]	0.000165
4	holiday	0.000151
2	season[T.C]	0.000126
5	workingday	0.000102
12	Recency	0.000053
0	Intercept	0.000000

In [78]:
```
importances = pd.Series(fine_tuned_RF.feature_importances_, index =X_train_over.columns)
importances.plot(kind='bar')
```

Out[78]:
```
<AxesSubplot:>
```



In [79]:
```
y_train_over.values.ravel()
```

Out[79]:
```
array([0., 0., 0., ..., 3., 3., 3.])
```

XGBoost 별도 수행(시간 없으니 꼭 필요할 때만 하기)

- 컬럼명에 대괄호, 콤마, 부등호가 있으면 에러남
- 수기로 바꿔줘야함

In [80]:
```
X_train_over.columns
```

Out[80]:
```
Index(['Intercept', 'season[T.B]', 'season[T.C]', 'season[T.D]', 'holiday',
       'workingday', 'weather', 'atemp', 'humidity', 'windspeed', 'casual',
       'registered', 'Recency', 'Monetary', 'kmeans'],
      dtype='object')
```

In [81]:
```
X_train_over.columns = ['Intercept', 'season_B', 'season_C', 'season_D', 'holiday',
       'workingday', 'weather', 'atemp', 'humidity', 'windspeed', 'casual',
       'registered', 'Recency', 'Monetary', 'kmeans']
```

- GridsearchCV가 불안정하므로 수기로 max_depth만 바꿔서 두 번 해보기

In [82]:

```
# 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
from xgboost import XGBClassifier

xgb = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=3)
xgb.fit(X_train_over, y_train_over)


kfold = KFold(n_splits=num_folds, shuffle=True, random_state=seed)
# kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_state=seed)

cv_results = cross_val_score(xgb, X_train_over, y_train_over, cv=kfold, scoring=scoring)
# results.append(cv_results)
# names.append(name)
msg = "[%s]\tmean: %f\tstd: %f" % ('XGB', cv_results.mean(), cv_results.std())
print(msg)
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\dask\dataframe\utils.py:14: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[XGB]   mean: 0.995431  std: 0.001508
```

In [83]:

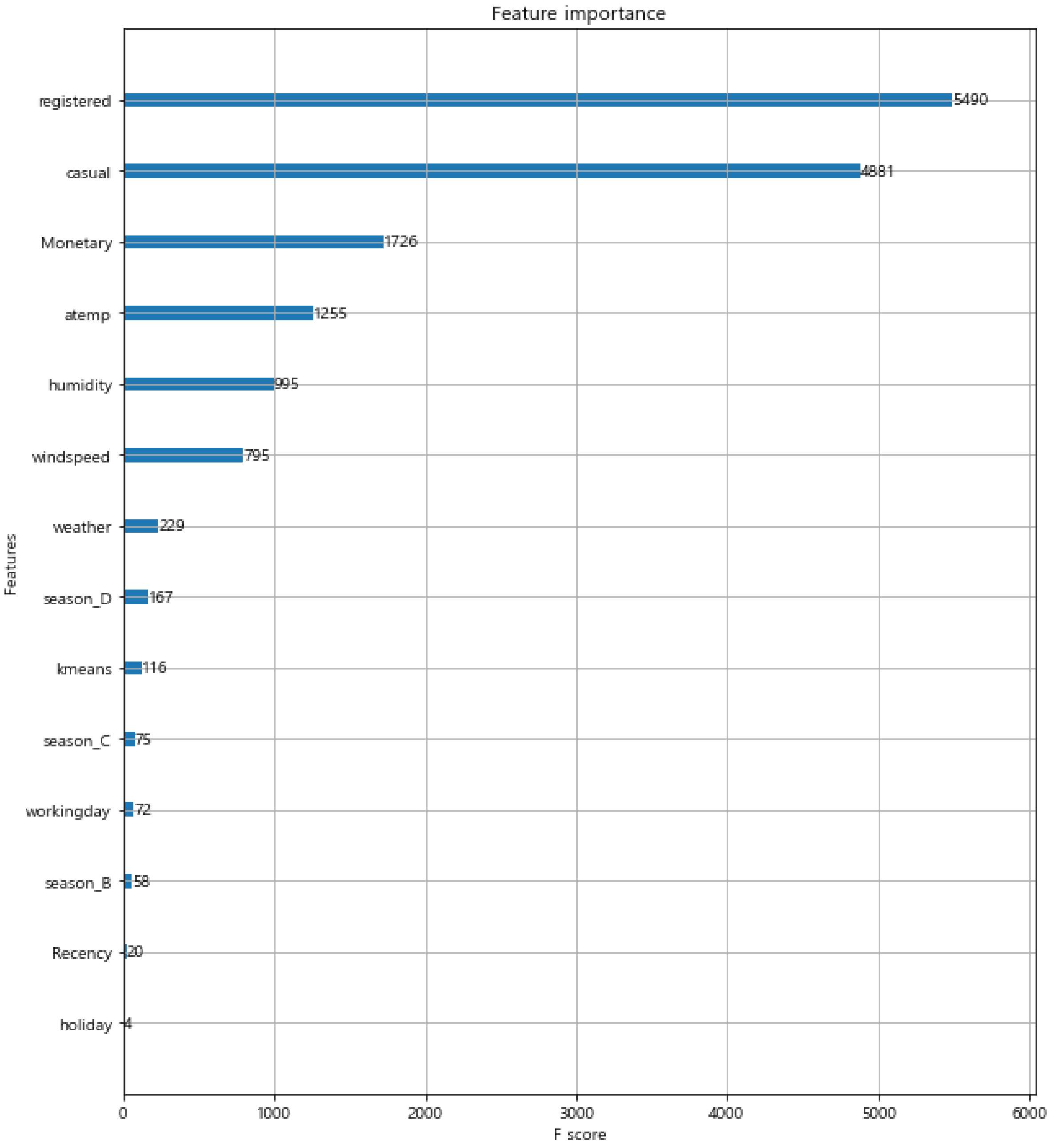
```
xgb = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=5) # max_depth를 5로만 바꿈
xgb.fit(X_train_over, y_train_over)

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

cv_results = cross_val_score(xgb, X_train_over, y_train_over, cv=kfold, scoring=scoring)
msg = "[%s]\tmean: %f\tstd: %f" % ('XGB', cv_results.mean(), cv_results.std())
print(msg)
```

```
[XGB]   mean: 0.995624  std: 0.001597
```

- 변수중요도 시각화

In [84]:
```
from xgboost import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(figsize=(10, 12))
# 사이킷런 래퍼 클래스를 입력해도 무방.
plot_importance(xgb, ax=ax)
```

Out[84]:
```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel
='Features'>
```



Test set 활용하여 예측 수행

In [85]:
```
y_pred = fine_tuned_RF.predict(X_test_scale)
```

In [86]:
```
confusion_matrix(y_test, y_pred)
```

Out[86]:
```
array([[1656,    8,    0,    0],
       [   5,  838,    4,    0],
       [   0,    4,  399,    5],
       [   0,    0,   10,  331]], dtype=int64)
```

F1 score

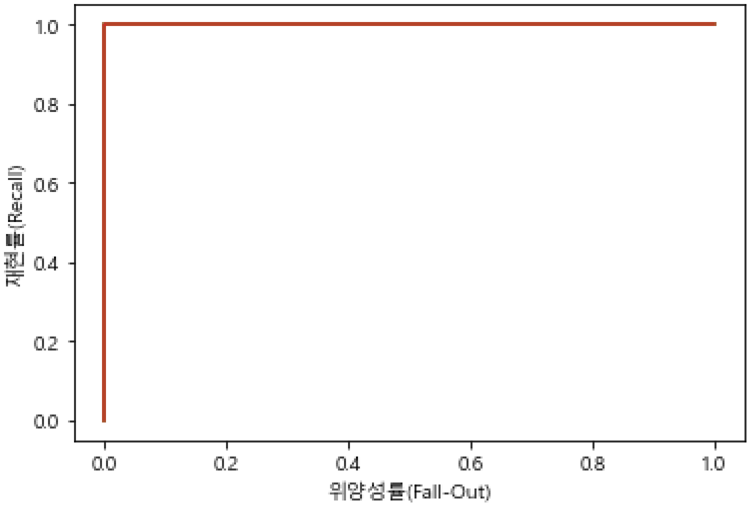
In [87]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1664
         1.0       0.99      0.99      0.99       847
         2.0       0.97      0.98      0.97       408
         3.0       0.99      0.97      0.98       341

    accuracy                           0.99      3260
   macro avg       0.98      0.98      0.98      3260
weighted avg       0.99      0.99      0.99      3260
```

ROC AUC

In [89]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
import matplotlib.pyplot as plt
# from sklearn.naive_bayes import GaussianNB
# from sklearn.datasets import load_iris
from sklearn.preprocessing import label_binarize

# iris = load_iris()
# X = iris.data   # 독립변수가 있고


# 이 아래부터 활용하면 됨
X = X_train_over
y = label_binarize(y_train_over, classes = [0, 1, 2, 3])   # 종속변수 y를 더미화를 시킴

n = 4 # class 개수만큼(여기선 4개였음)
fpr = [None] * n
tpr = [None] * n
threshold = [None] * n
roc_auc = []

for i in range(n):
    model = fine_tuned_RF.fit(X, y[:, i])   # 모델링을 함
    fpr[i], tpr[i], threshold[i] = roc_curve(y[:, i], model.predict_proba(X)[:, 1])
    plt.plot(fpr[i], tpr[i])

    roc_auc.append(auc(fpr[i], tpr[i]))

plt.xlabel('위양성률(Fall-Out)')
plt.ylabel('재현률(Recall)')
plt.show()
print('ROC_AUC : ',roc_auc)
```



```
ROC_AUC :  [1.0, 1.0, 1.0, 1.0]
```