시계열 분석

라이브러리 호출

In [68]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
%matplotlib inline
pd.options.display.max_columns = None

import os

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from pmdarima.arima import auto_arima  ## ADP 볼 때는 없을 패키지

import itertools # 내장 패키지
```

그래프 한글 깨짐 방지

In [69]:
```
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 [70]:
```
df = pd.read_csv('./data/bikeshare.csv')
```

데이터 구조 확인

In [71]:
```
df.head()
```

Out[71]:

	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 [72]:
```
df.tail()
```

Out[72]:

	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 [73]:
```
df.shape
```

Out[73]:
```
(10886, 12)
```

In [74]:
```
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
```

날짜 데이터 전처리

날짜 데이터를 Timestamp 형식으로 변환

In [75]:
```
# 날짜 형식이 연, 월, 일, 시, 분, 초 형태일 때
df['datetime'] = pd.to_datetime(df['datetime'])

# pd.to_datetime('2012-12-19 20:00', format='%Y-%m-%d %H:%M')
```

================ 참고 : 날짜형식이 13자리 숫자일 때 대비

In [76]:
```
# 13자리 숫자일 때
import datetime
timestamp = 1463460958000
datetimeobj = datetime.datetime.fromtimestamp(timestamp/1000)
print(datetimeobj, type(datetimeobj))
# 이후에 pd.to_datetime으로 변환
a = pd.to_datetime(datetimeobj)
print(a, type(a))
```

```
2016-05-17 13:55:58 <class 'datetime.datetime'>
2016-05-17 13:55:58 <class 'pandas._libs.tslibs.timestamps.Timestamp'>
```

In [77]:
```
# datetime to timestamp
import time
timestamp = time.mktime(datetimeobj.timetuple())
timestamp
```

Out[77]:
```
1463460958.0
```

In [78]:
```
import datetime
datetime.date(year=2019, month=10, day=1)
```

Out[78]:
```
datetime.date(2019, 10, 1)
```

=============== 여기까지

날짜 데이터로부터 연, 월, 일, 시, 요일 데이터 추출

In [79]:
```
df['year'] = df['datetime'].map(lambda x: x.year) # 연
df['month'] = df['datetime'].map(lambda x: x.month) # 월
df['day'] = df['datetime'].map(lambda x: x.day) # 일
df['hour'] = df['datetime'].map(lambda x: x.hour) # 시
df['dayofweek'] = df['datetime'].map(lambda x: x.dayofweek) # 요일
```

날짜 데이터를 시 기준으로 그룹핑(다른 데이터는 평균값 계산) 후 인덱스로 설정

In [80]:
```
# 그룹핑
df = df.groupby(['year','month','day','hour'])['temp','humidity', 'windspeed',
'count'].mean().reset_index()
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykern
el_launcher.py:2: FutureWarning: Indexing with multiple keys (implicitly conv
erted to a tuple of keys) will be deprecated, use a list instead.
```

In [81]:
```
# 다시 date 컬럼 만들어주기
df['date'] = df['year'].astype('str') +'-'+ df['month'].astype('str') +'-'+ df
['day'].astype('str')\
+'-'+ df['hour'].astype('str')

# date 컬럼 형식 변경 -> datetime
df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d-%H')
```

In [82]:
```
df = df.set_index('date')
```

In [83]:
```
df.head(3)
```

Out[83]:

date	year	month	day	hour	temp	humidity	windspeed	count
2011-01-01 00:00:00	2011	1	1	0	9.84	81	0.0	16
2011-01-01 01:00:00	2011	1	1	1	9.02	80	0.0	40
2011-01-01 02:00:00	2011	1	1	2	9.02	80	0.0	32

In [84]:
```
# 필요 컬럼만 선택
df = df[['temp','humidity','windspeed','count']]
```

In [85]:
```
df.head()
```

Out[85]:

date	temp	humidity	windspeed	count
2011-01-01 00:00:00	9.84	81	0.0	16
2011-01-01 01:00:00	9.02	80	0.0	40
2011-01-01 02:00:00	9.02	80	0.0	32
2011-01-01 03:00:00	9.84	75	0.0	13
2011-01-01 04:00:00	9.84	75	0.0	1

In [116]:
```
# 시계열용 데이터는 따로 빼둠
dfts = pd.DataFrame(df['count'])
```

In [117]:
```
dfts
```

Out[117]:
```
date
2011-01-01 00:00:00     16.0
2011-01-01 01:00:00     40.0
2011-01-01 02:00:00     32.0
2011-01-01 03:00:00     13.0
2011-01-01 04:00:00      1.0
                        ...
2012-12-19 19:00:00    336.0
2012-12-19 20:00:00    241.0
2012-12-19 21:00:00    168.0
2012-12-19 22:00:00    129.0
2012-12-19 23:00:00     88.0
Name: count, Length: 10886, dtype: float64
```

======== 여기부터는 y값 외에독립변수가 더 있을 경우

========== 시계열 모델만 만들 거면 비시계열 모델링 이후로 이동

데이터 타입 맞춰주기 ==================================

In [86]:
```
df.columns
```

Out[86]:
```
Index(['temp', 'humidity', 'windspeed', 'count'], dtype='object')
```

In [87]:
```
col_id = []
col_cat =[]
col_int = []
col_float = ['temp','humidity','windspeed','count']
col_bool = []
col_num = col_int+col_float
```

In [88]:
```
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 [89]:
```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()
    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 [90]:
```python
DA1 = DA(df[col_num])
DA1
```

Out[90]:

	total	count	missing	mean	median	std	variance	skewness	kurtosis	m
temp	10886.0	10886.0	0	20.23	20.5	7.79	60.70	0.00	-0.91	0.
humidity	10886.0	10886.0	0	61.89	62.0	19.25	370.34	-0.09	-0.76	0.
windspeed	10886.0	10886.0	0	12.80	13.0	8.16	66.65	0.59	0.63	0.
count	10886.0	10886.0	0	191.57	145.0	181.14	32810.30	1.24	1.30	1.

범주형 변수

In [91]:
```
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 [92]:
```
DA2 = DA_cat(df,col_cat+col_bool)
DA2
```

Out[92]:
```
_
```

전처리(중복값, 결측치, 이상치 처리)

중복값

In [93]:
```
df[df.duplicated(keep=False)].sort_values(['temp','humidity','windspeed','coun
t'])
```

Out[93]:

date	temp	humidity	windspeed	count
2012-01-04 02:00:00	0.82	34.0	19.0012	1.0
2012-01-04 03:00:00	0.82	34.0	19.0012	1.0
2011-01-09 04:00:00	3.28	53.0	12.9980	1.0
2011-01-09 05:00:00	3.28	53.0	12.9980	1.0
2011-02-10 05:00:00	4.92	50.0	15.0013	6.0
...	...	...	...	...
2012-08-08 03:00:00	28.70	84.0	0.0000	7.0
2012-08-08 04:00:00	28.70	84.0	0.0000	7.0
2011-08-07 05:00:00	28.70	89.0	12.9980	5.0
2012-07-09 02:00:00	28.70	89.0	12.9980	5.0
2012-07-09 03:00:00	28.70	89.0	12.9980	5.0

161 rows × 4 columns

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

결측치

In [94]:
```
df.isna().sum()
```

Out[94]:
```
temp         0
humidity     0
windspeed    0
count        0
dtype: int64
```

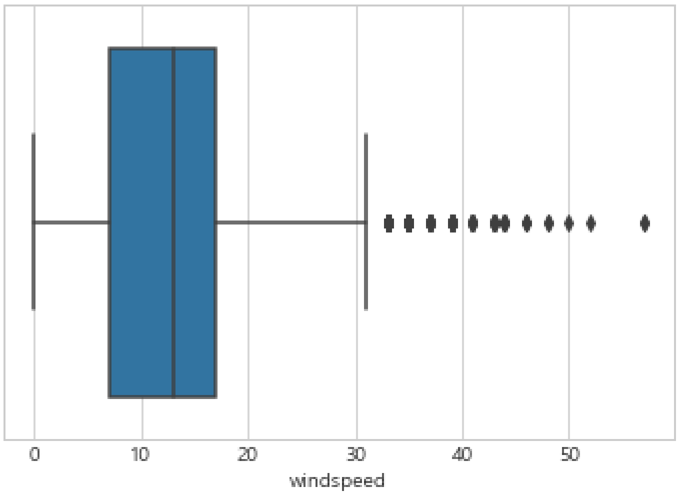
In [ ]:
```
# na 처리 : dropna(), fillna()
df.dropna() # nan이 하나라도 들어간 행은 삭제
df.dropna(how = 'all') # 데이터가 모두 nan인 행만 삭제 / 초기값:'any'
## Parameters
# axis = 'index' / 'columns'
# subset = ['col1', 'col2', ...] # 적용 대상 컬럼 특정

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 [95]:
```
tmp = 'windspeed'
```

In [96]:
```
sns.boxplot(y = tmp, data = df, orient = 'h')
```

Out[96]:
```
<AxesSubplot:xlabel='windspeed'>
```



In [97]:
```
# 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
```

```
(227, 4)
```

Out[97]:

	temp	humidity	windspeed	count
date				
2011-01-08 14:00:00	8.20	32.0	32.9975	95.0
2011-01-08 17:00:00	6.56	37.0	36.9974	69.0
2011-01-09 09:00:00	4.92	46.0	35.0008	19.0
2011-01-09 11:00:00	6.56	40.0	35.0008	49.0
2011-01-12 12:00:00	8.20	47.0	39.0007	55.0
...	...	...	...	...
2012-11-02 14:00:00	16.40	40.0	32.9975	262.0
2012-11-08 12:00:00	16.40	24.0	32.9975	235.0
2012-11-13 01:00:00	18.04	88.0	43.0006	5.0
2012-12-05 14:00:00	19.68	33.0	32.9975	218.0
2012-12-18 15:00:00	18.86	44.0	32.9975	246.0

227 rows × 4 columns

*min/max*값으로 보정

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

파생변수 생성

In [98]:
```
today = pd.to_datetime('2020-12-13')
```

In [99]:
```
# Recency
cond1 = (today-df.index) >= pd.Timedelta('3000 days')
cond2 = ((today-df.index) < pd.Timedelta('3000 days'))&((today-df.index) >= pd
.Timedelta('2000 days'))
cond3 = (today-df.index) < pd.Timedelta('2000 days')

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

In [100]:
```
# Frequency
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 [101]:
```
# Monetary
df['Monetary'] = df['count'] * df['temp']
```

In [102]:
```
df.head(3)
```

Out[102]:

date	temp	humidity	windspeed	count	Recency	Frequency	Monetary
2011-01-01 00:00:00	9.84	81.0	0.0	16.0	1.0	2.0	157.44
2011-01-01 01:00:00	9.02	80.0	0.0	40.0	1.0	3.0	360.80
2011-01-01 02:00:00	9.02	80.0	0.0	32.0	1.0	3.0	288.64

데이터 마트 DQ Check, 변수선택 및 EDA

DQ Check

In [103]:
```
df.columns
```

Out[103]:
```
Index(['temp', 'humidity', 'windspeed', 'count', 'Recency', 'Frequency',
       'Monetary'],
      dtype='object')
```

In [104]:
```
col_num = ['temp', 'humidity', 'windspeed', 'count', 'Monetary']
col_cat = ['Recency', 'Frequency']
```

In [105]:
```
DA3 = DA(df[col_num])
DA3
```

Out[105]:

	total	count	missing	mean	median	std	variance	skewness	kurtos
temp	10886.0	10886.0	0	20.23	20.50	7.79	60.70	0.00	-0.
humidity	10886.0	10886.0	0	61.89	62.00	19.25	370.34	-0.09	-0.
windspeed	10886.0	10886.0	0	12.80	13.00	8.16	66.65	0.59	0.
count	10886.0	10886.0	0	191.57	145.00	181.14	32810.30	1.24	1.
Monetary	10886.0	10886.0	0	4432.39	2629.74	5023.07	25228922.33	1.66	2.

In [106]:
```
DA4 = DA_cat(df, col_cat)
DA4
```

Out[106]:

	col_nm	class	count	ratio
0	Recency	1.0	9519	0.87
1	Recency	2.0	1367	0.13
2	Frequency	1.0	1229	0.11
3	Frequency	2.0	631	0.06
4	Frequency	3.0	9026	0.83

변수 제외

In [107]:
```
df = df.drop(columns = ['Recency'], axis=1)
```

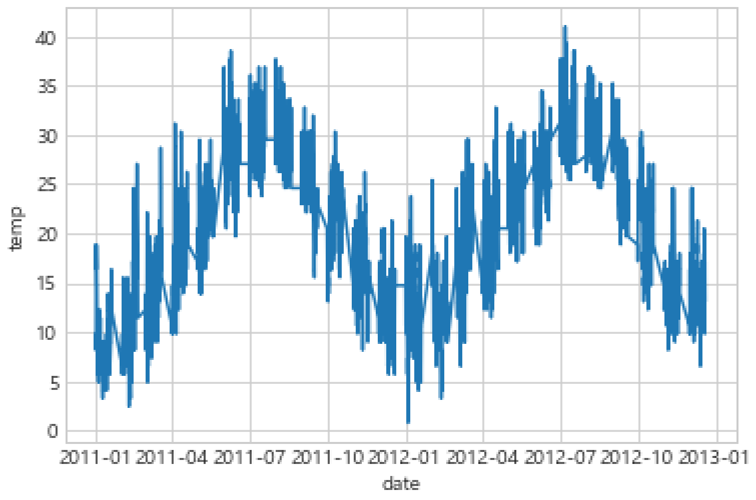
EDA

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

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

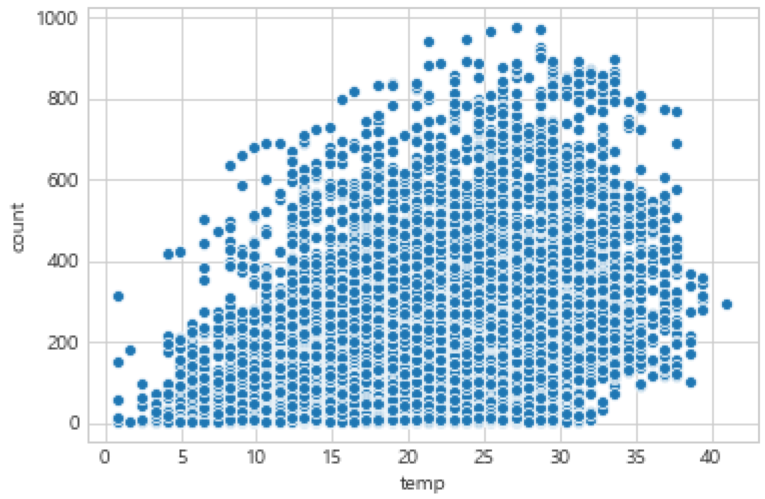
In [108]:
```
sns.lineplot(x = df.index, y = 'temp', data = df)
```

Out[108]:
```
<AxesSubplot:xlabel='date', ylabel='temp'>
```



In [109]:
```
sns.scatterplot(x = 'temp', y = 'count', data = df)
```

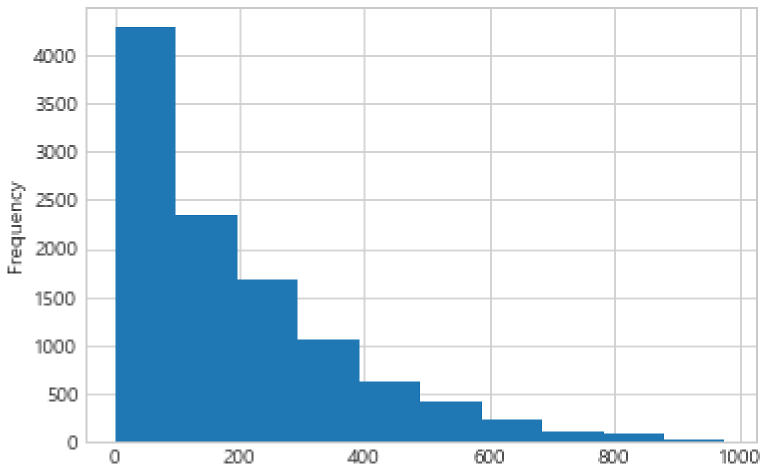
Out[109]:
```
<AxesSubplot:xlabel='temp', ylabel='count'>
```



종속변수 분포 확인 및 전처리

```
In [110]: df['count'].plot(kind='hist')
```

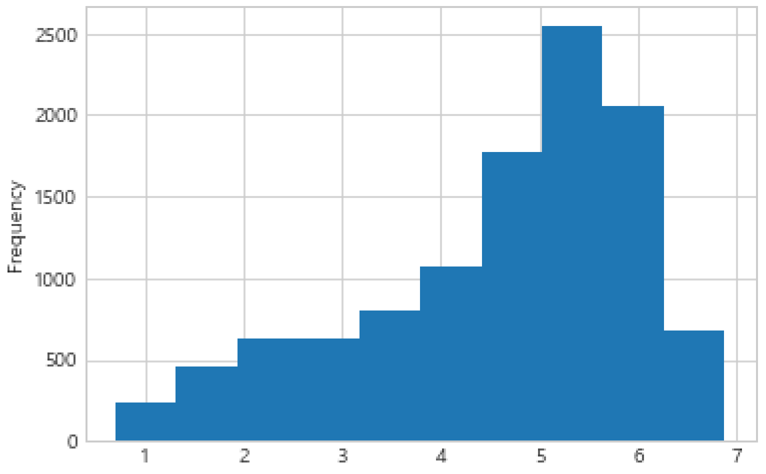
```
Out[110]: <AxesSubplot:ylabel='Frequency'>
```



```
In [111]: df['y2'] = np.log1p(df['count']) # inverse 는 np.expm1()
```

```
In [112]: df['y2'].plot(kind='hist')
```

```
Out[112]: <AxesSubplot:ylabel='Frequency'>
```



In [113]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 10886 entries, 2011-01-01 00:00:00 to 2012-12-19 23:00:00
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   temp       10886 non-null  float64
 1   humidity   10886 non-null  float64
 2   windspeed  10886 non-null  float64
 3   count      10886 non-null  float64
 4   Frequency  10886 non-null  float64
 5   Monetary   10886 non-null  float64
 6   y2         10886 non-null  float64
dtypes: float64(7)
memory usage: 680.4 KB
```

In [114]:
```
df.head(3)
```

Out[114]:

	temp	humidity	windspeed	count	Frequency	Monetary	y2
date							
2011-01-01 00:00:00	9.84	81.0	0.0	16.0	2.0	157.44	2.833213
2011-01-01 01:00:00	9.02	80.0	0.0	40.0	3.0	360.80	3.713572
2011-01-01 02:00:00	9.02	80.0	0.0	32.0	3.0	288.64	3.496508

In [118]:
```
import statsmodels.api as sm
from patsy import dmatrices

y, X = dmatrices('y2 ~ temp + humidity + windspeed + Frequency + Monetary', da
ta=df, return_type='dataframe')
```

VIF 확인 필요 (y값 섞여들어가지 않게 주의!!)

In [119]:
```
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.s
hape[1])]
vif['features'] = X.columns
vif
```

Out[119]:

	VIF Factor	features
0	47.765907	Intercept
1	1.576327	temp
2	1.257986	humidity
3	1.116640	windspeed
4	1.180948	Frequency
5	1.860919	Monetary

train, test split

In [120]:
```
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 [121]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

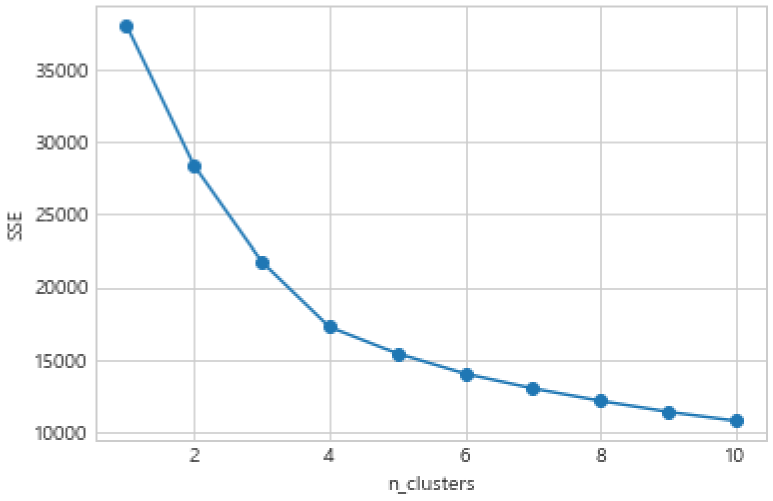
In [122]:
```
scaler.fit(X_train)
X_train_scale = scaler.transform(X_train)
X_test_scale = scaler.transform(X_test)
```

In [123]:
```
# 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 [124]:
```
# 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 [125]:
```
elbow(X_train_scale)
```



In [126]:
```
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)
```

```
[0.24  0.282 0.283 0.248 0.236 0.229 0.22  0.219 0.213]
```

군집 수 직접 지정해서 군집화

In [127]:
```
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300,random_state=0)
kmeans.fit(X_train_scale)
```

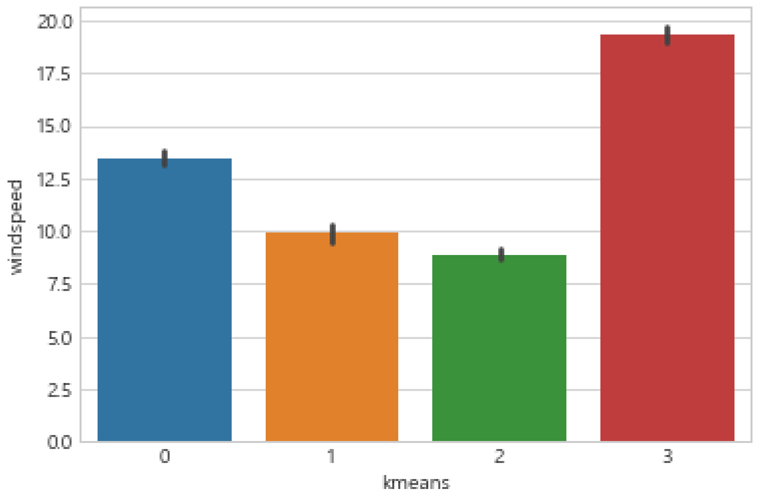
Out[127]:
```
KMeans(n_clusters=4, random_state=0)
```

군집화 결과 프로파일링

In [128]:
```
# 스케일링 풀고 프로파일링

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

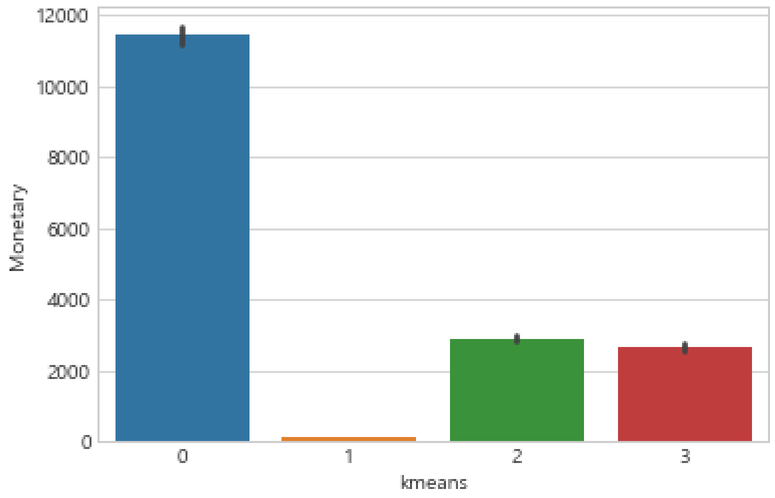
In [130]:
```
sns.barplot(df_profile['kmeans'], df_profile['windspeed'])
```

Out[130]:
```
<AxesSubplot:xlabel='kmeans', ylabel='windspeed'>
```



In [132]:
```
sns.barplot(df_profile['kmeans'], df_profile['Monetary'])
```

Out[132]:
```
<AxesSubplot:xlabel='kmeans', ylabel='Monetary'>
```



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

In [133]:
```
X_train_scale['kmeans'] = kmeans.labels_
```

In [134]:
```
kmeans_test = kmeans.predict(X_test_scale)
X_test_scale['kmeans'] = kmeans_test
```

모델링

In [136]:
```
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 [137]:
```
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 [138]:
```
models
```

Out[138]:
```
[('Ridge', Ridge()),
 ('Lasso', Lasso()),
 ('ElasticNet', ElasticNet()),
 ('Huber', HuberRegressor()),
 ('DT', DecisionTreeRegressor()),
 ('RF', RandomForestRegressor()),
 ('KNN', KNeighborsRegressor()),
 ('KernelRidge', KernelRidge()),
 ('MLP', MLPRegressor()),
 ('SVR', SVR())]
```

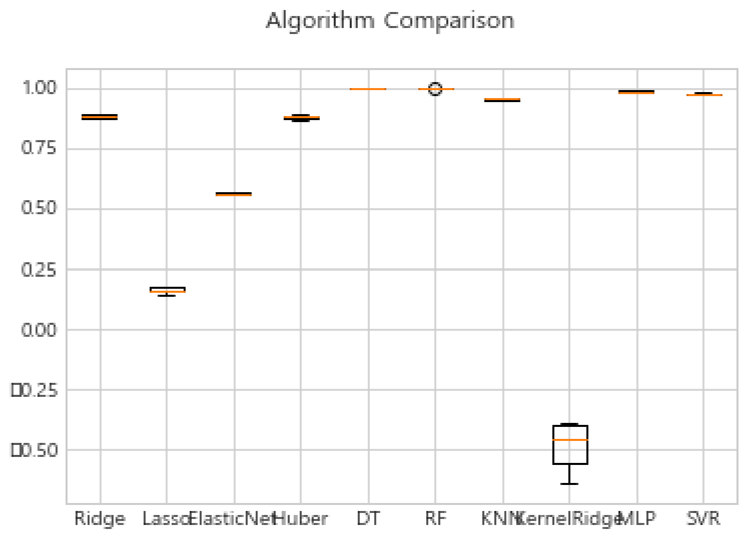
In [139]:
```
num_folds = 5
seed = 7
```

In [143]:
```
names = []
results = []

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

for name, model in models: # 에러나면 .values.ravel() 빼고  y_train으로 해보기
    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(5))
```

```
Ridge 0.88372
Lasso 0.16192
ElasticNet 0.56008
Huber 0.87945
DT 0.9986
RF 0.99925
KNN 0.95446
KernelRidge -0.4884
MLP 0.98492
SVR 0.9748
```

In [142]:
```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:238: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:201: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0, flags=flags)
```



============ 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 [ ]:
```
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())  # 에러나면 .valu
es.ravel() 빼고  y_train으로 해보기
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))
```

In [156]:
```
fine_tuned_RF = grid_result.best_estimator_
print('best params: ', grid_result.best_params_)
fine_tuned_RF.feature_importances_
```

```
best params:  {'max_features': 7, 'n_estimators': 50}
```

Out[156]:
```
array([0.00000000e+00, 3.16966872e-02, 1.46239923e-04, 1.14957090e-04,
       1.11716386e-01, 8.56316024e-01, 9.70581592e-06])
```

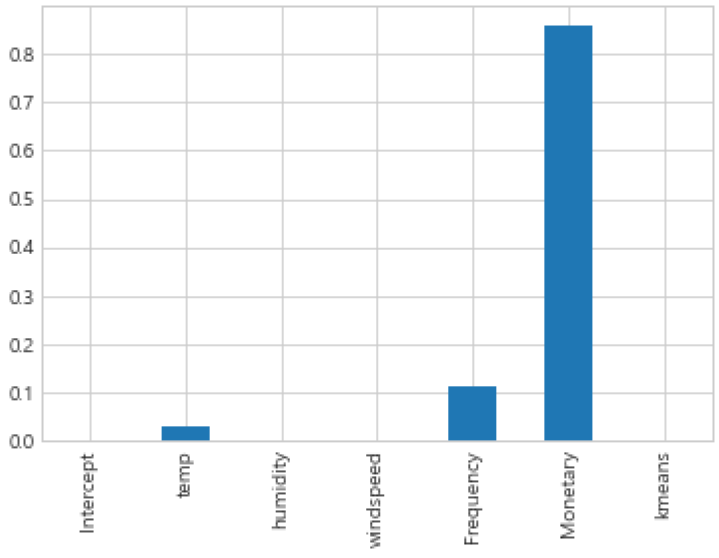
In [157]:
```
pd.DataFrame({'col':X_train_scale.columns, 'FI':fine_tuned_RF.feature_importan
ces_}).sort_values('FI', ascending=False)
```

Out[157]:

	col	FI
5	Monetary	0.856316
4	Frequency	0.111716
1	temp	0.031697
2	humidity	0.000146
3	windspeed	0.000115
6	kmeans	0.000010
0	Intercept	0.000000

In [158]:
```
importances = pd.Series(fine_tuned_RF.feature_importances_, index=X_train_scal
e.columns)
importances.plot(kind='bar')
```

Out[158]:
```
<AxesSubplot:>
```



Test set 활용하여 예측 수행

In [159]:
```
y_pred = fine_tuned_RF.predict(X_test_scale)
```

In [160]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

R^2

In [161]:
```
r2 = r2_score(np.expm1(y_test), np.expm1(y_pred)) # log처리 안 했으면 그냥 y_tes
t, y_pred
r2
```

Out[161]:
```
0.9995663230063383
```

MSE

In [162]:
```
mse = mean_squared_error(np.expm1(y_test), np.expm1(y_pred)) # log처리 안 했으면
그냥 y_test, y_pred
mse
```

Out[162]:
```
0.0008766132338490894
```

RMSE

```
In [163]: rmse = np.sqrt(mse)
          rmse
```

```
Out[163]: 0.029607654987335443
```

MAE

```
In [164]: mae = mean_absolute_error(np.expm1(y_test), np.expm1(y_pred)) # log처리 안 했으
          면 그냥 y_test, y_pred
          mae
```

```
Out[164]: 0.013092585562396663
```

MAPE

```
In [165]: 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 [166]: mape = mp(np.expm1(y_test), np.expm1(y_pred)) # log처리 안 했으면 그냥 y_test, y_
          pred
          mape
```

```
Out[166]: 50.28324823737699
```

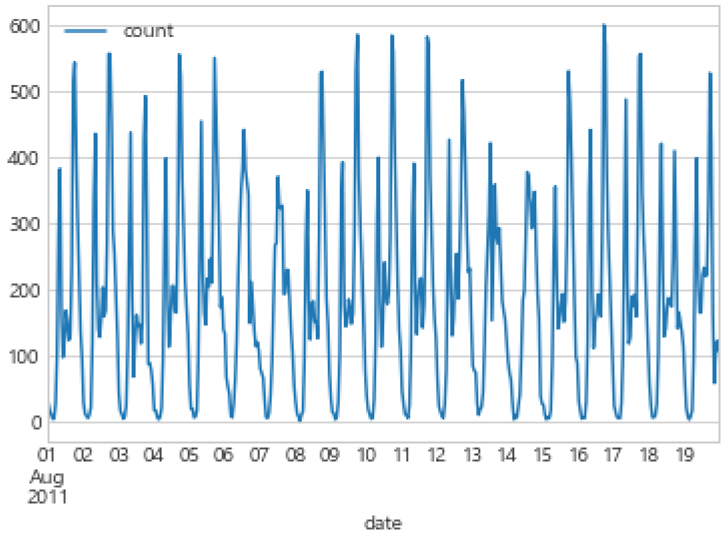
===========================

```
In [197]: # 데이터를 일부러 줄임(2011년 8월 자료로만)
          data = dfts[(dfts.index.year == 2011)&(dfts.index.month == 8)]
```

```
In [198]: data.shape
```

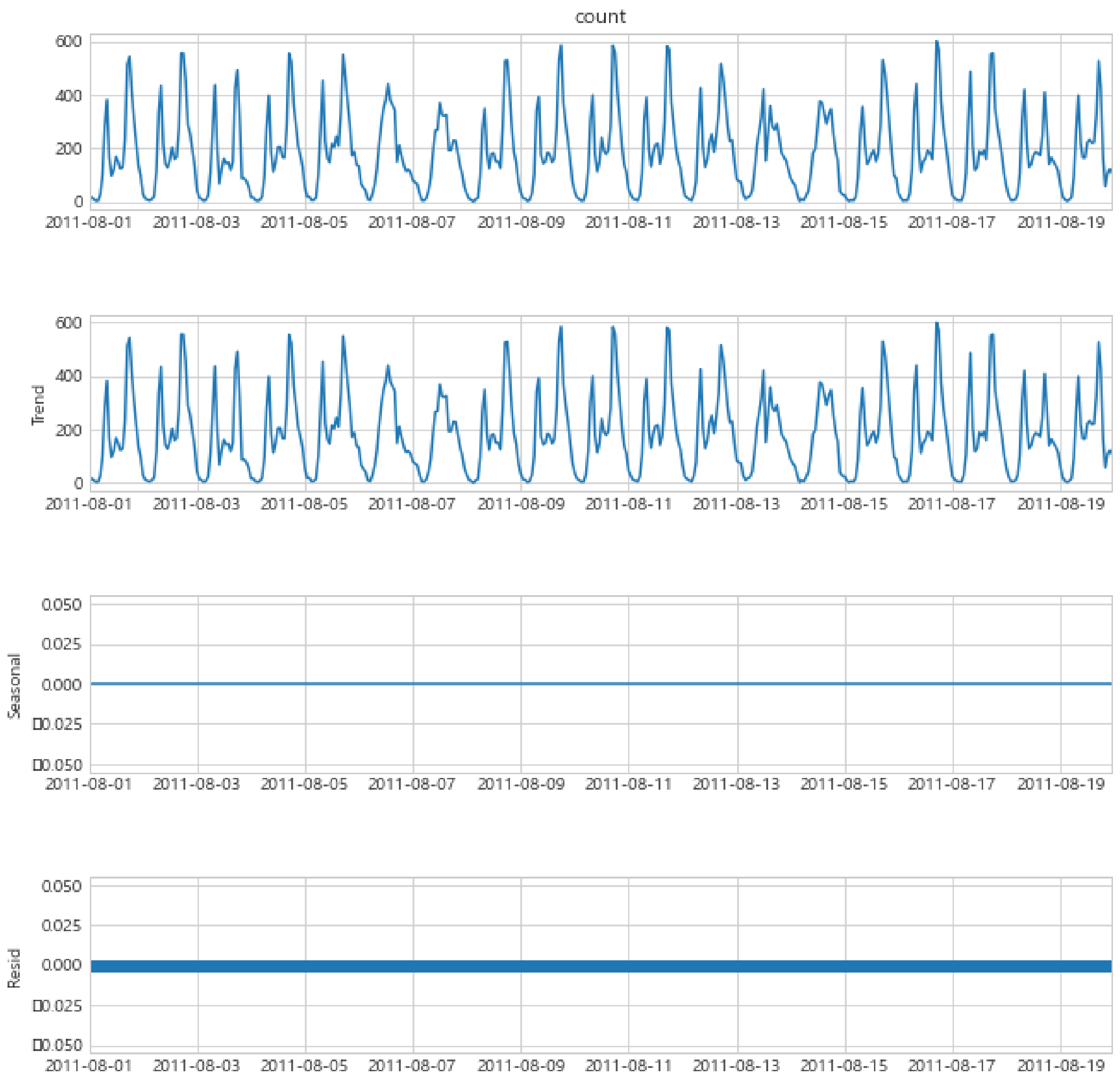
```
Out[198]: (456, 1)
```

In [199]:
```
fig = data.plot()
```



In [200]:
```
# Seasonal decomposition plot : Seasonal decomposition using moving averages.

# Observed : observed data
# Trend : The estimated trend component
# Seasonal : The estimated seasonal component
# resid : The estimated residuals
decomposition = sm.tsa.seasonal_decompose(data['count'], model = 'additive', p
eriod=1)
fig = decomposition.plot()
fig.set_size_inches(10,10)
plt.show()
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:238: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:201: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0, flags=flags)
```



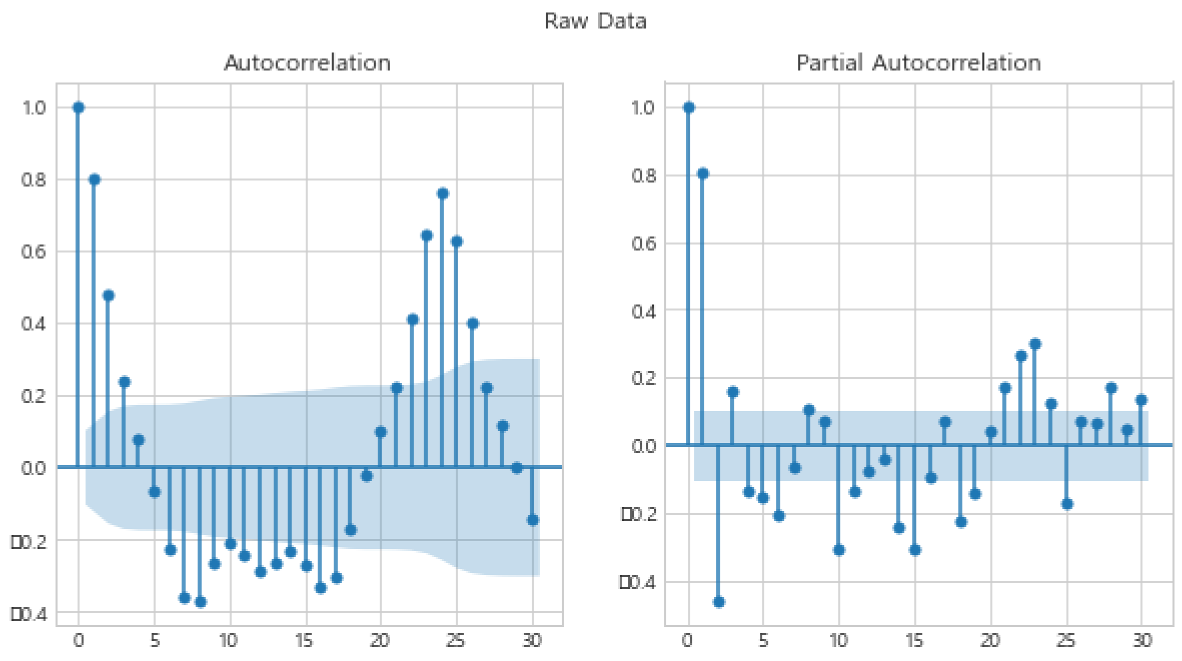
Train, test set split

In [201]:
```
# Tr, Te = 8:2
train_data, test_data = train_test_split(data, test_size=0.2, shuffle=False)
```

정상성 확인

In [203]:
```
# ACF, PACF plot

fig, ax = plt.subplots(1,2, figsize = (10, 5))
fig.suptitle('Raw Data')
sm.graphics.tsa.plot_acf(train_data.values.squeeze(), lags = 30, ax = ax[0])
sm.graphics.tsa.plot_pacf(train_data.values.squeeze(), lags = 30, ax = ax[1])
plt.show()
### ACF 그래프가 점진적으로 감소하는 것은 전형적인 Non-stationary 데이터이다 = 정상정이
 없음
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:238: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:201: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0, flags=flags)
```



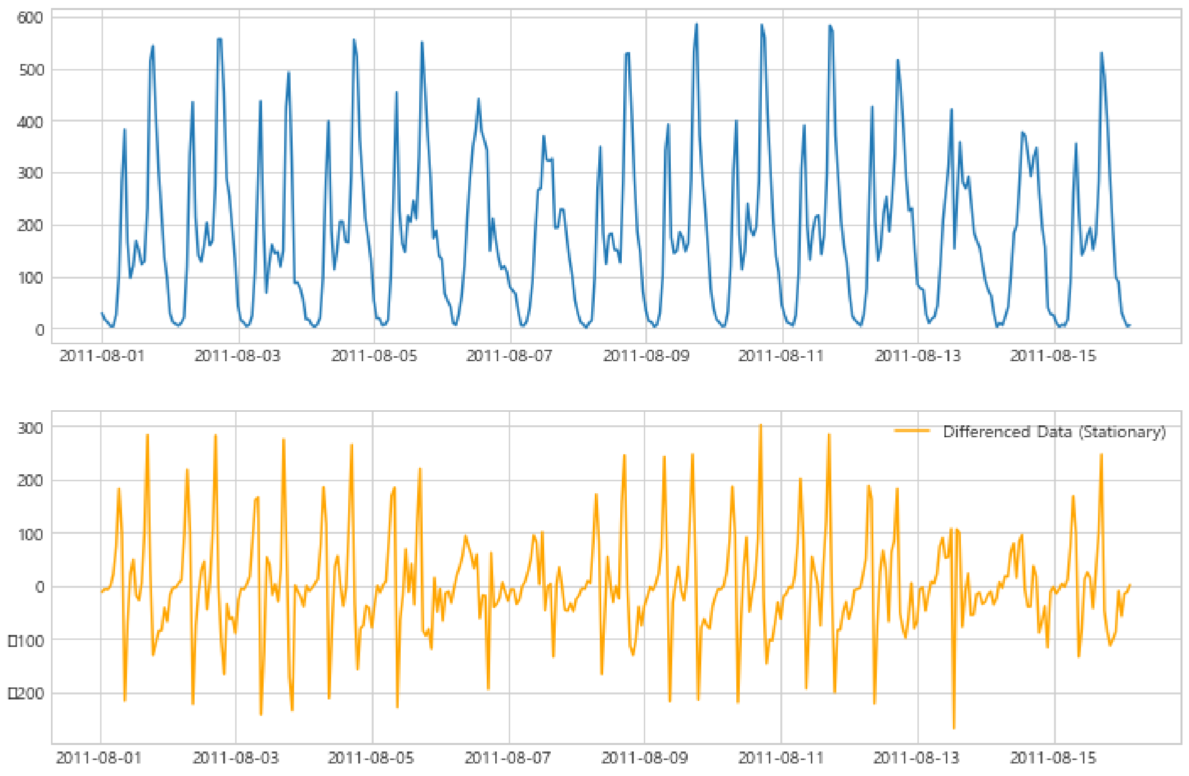
차분

In [205]:
```
# Differencing

diff_train_data = train_data.copy()
diff_train_data = diff_train_data['count'].diff()
diff_train_data = diff_train_data.dropna()
print('##### Raw Data #####')
print(train_data)
print('### Differenced Data ###')
print(diff_train_data)
```

```
##### Raw Data #####
                     count
date
2011-08-01 00:00:00   29.0
2011-08-01 01:00:00   17.0
2011-08-01 02:00:00   11.0
2011-08-01 03:00:00    4.0
2011-08-01 04:00:00    4.0
...                    ...
2011-08-15 23:00:00   88.0
2011-08-16 00:00:00   31.0
2011-08-16 01:00:00   16.0
2011-08-16 02:00:00    4.0
2011-08-16 03:00:00    6.0

[364 rows x 1 columns]
### Differenced Data ###
date
2011-08-01 01:00:00   -12.0
2011-08-01 02:00:00    -6.0
2011-08-01 03:00:00    -7.0
2011-08-01 04:00:00     0.0
2011-08-01 05:00:00    22.0
                       ...
2011-08-15 23:00:00   -10.0
2011-08-16 00:00:00   -57.0
2011-08-16 01:00:00   -15.0
2011-08-16 02:00:00   -12.0
2011-08-16 03:00:00     2.0
Name: count, Length: 363, dtype: float64
```

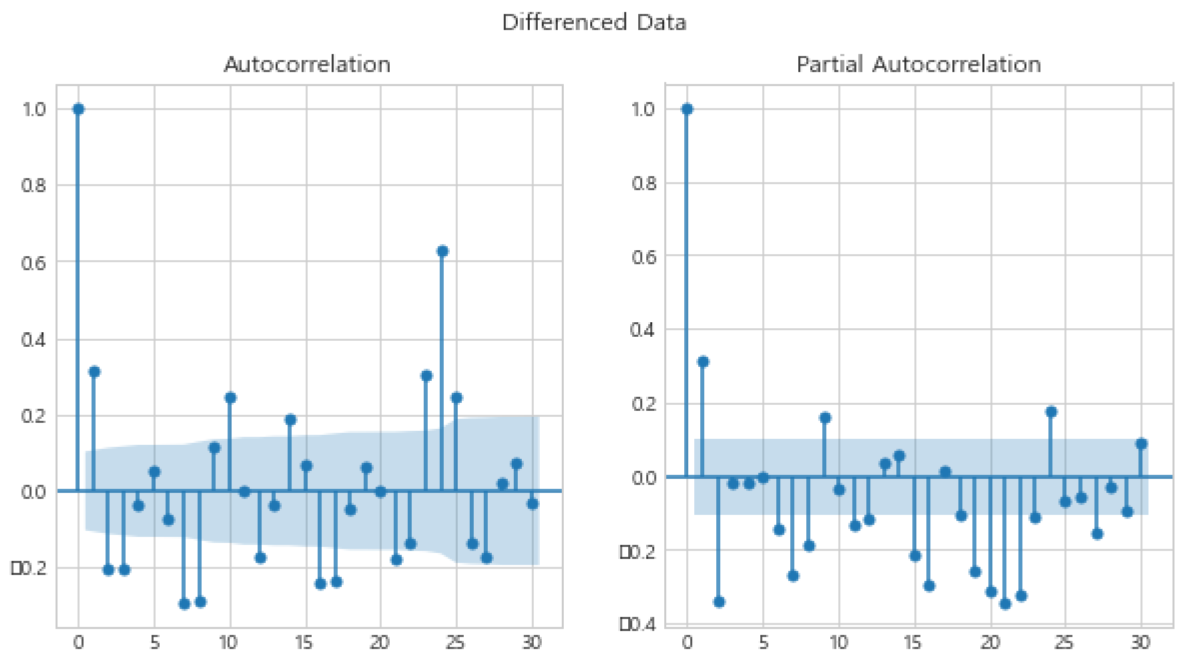
```
In [207]: # differenced data plot

          plt.figure(figsize = (12,8))
          plt.subplot(211)
          plt.plot(train_data['count'])
          plt.subplot(212)
          plt.plot(diff_train_data, 'orange') # first difference (t - (t-1))
          plt.legend(['Differenced Data (Stationary)'])
          plt.show()
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:238: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:201: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0, flags=flags)
```



```
In [208]: # ACF, PACF plot

fig, ax = plt.subplots(1,2, figsize = (10, 5))
fig.suptitle('Differenced Data')
sm.graphics.tsa.plot_acf(diff_train_data.values.squeeze(), lags = 30, ax = ax[
0])
sm.graphics.tsa.plot_pacf(diff_train_data.values.squeeze(), lags = 30, ax = ax
[1])
plt.show()
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:238: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:201: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0, flags=flags)
```



기본 모델 생성

In [209]:
```
# ARIMA model fitting
# The (p, d, q) order of the model for the number of AR parameters, difference
s, and MA parameters to use.

model = ARIMA(train_data.values, order=(1,1,1))
model_fit = model.fit()
model_fit.summary()

# AIC 값은 1069.44이고, constant의 p-value 값이 유의미하지 않게 나왔다.
```

Out[209]:

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	363
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-2122.751
Method:	css-mle	**S.D. of innovations**	83.807
Date:	Wed, 09 Dec 2020	**AIC**	4253.502
Time:	22:54:27	**BIC**	4269.079
Sample:	1	**HQIC**	4259.694

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0571	6.142	-0.009	0.993	-12.095	11.981
ar.L1.D.y	-0.1120	0.102	-1.094	0.274	-0.313	0.089
ma.L1.D.y	0.5537	0.084	6.606	0.000	0.389	0.718

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-8.9287	+0.0000j	8.9287	0.5000
MA.1	-1.8059	+0.0000j	1.8059	0.5000

In [210]:
```
model = ARIMA(train_data.values, order=(0,1,1))
model_fit = model.fit()
model_fit.summary()
```

Out[210]:

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	363
Model:	ARIMA(0, 1, 1)	**Log Likelihood**	-2123.378
Method:	css-mle	**S.D. of innovations**	83.952
Date:	Wed, 09 Dec 2020	**AIC**	4252.756
Time:	23:04:33	**BIC**	4264.439
Sample:	1	**HQIC**	4257.400

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-0.0619	6.482	-0.010	0.992	-12.766	12.642
ma.L1.D.y	0.4724	0.047	10.044	0.000	0.380	0.565

Roots

	Real	**Imaginary**	**Modulus**	**Frequency**
MA.1	-2.1170	+0.0000j	2.1170	0.5000

In [211]:
```
model = ARIMA(train_data.values, order=(1,1,0))
model_fit = model.fit()
model_fit.summary()
```

Out[211]:

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	363
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-2136.885
Method:	css-mle	**S.D. of innovations**	87.152
Date:	Wed, 09 Dec 2020	**AIC**	4279.769
Time:	23:04:38	**BIC**	4291.453
Sample:	1	**HQIC**	4284.413

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-0.0760	6.640	-0.011	0.991	-13.089	12.937
ar.L1.D.y	0.3119	0.050	6.267	0.000	0.214	0.409

Roots

	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	3.2060	+0.0000j	3.2060	0.0000

최적 모델 탐색

In [213]:
```
print('Examples of parameter combinations for Seasonal ARIMA...')
p = range(0,3)
d = range(1,2)
q = range(0,3)

pdq = list(itertools.product(p, d, q))
pdq
```

```
Examples of parameter combinations for Seasonal ARIMA...
```

Out[213]:
```
[(0, 1, 0),
 (0, 1, 1),
 (0, 1, 2),
 (1, 1, 0),
 (1, 1, 1),
 (1, 1, 2),
 (2, 1, 0),
 (2, 1, 1),
 (2, 1, 2)]
```

In [214]:
```
aic=[]
for i in pdq:
    model = ARIMA(train_data.values, order=(i))
    model_fit = model.fit()
    print(f'ARIMA: {i} >> AIC : {round(model_fit.aic, 2)}')
    aic.append(round(model_fit.aic,2))
```

```
ARIMA: (0, 1, 0) >> AIC : 4315.04
ARIMA: (0, 1, 1) >> AIC : 4252.76
ARIMA: (0, 1, 2) >> AIC : 4252.07
ARIMA: (1, 1, 0) >> AIC : 4279.77
ARIMA: (1, 1, 1) >> AIC : 4253.5
ARIMA: (1, 1, 2) >> AIC : 4198.08
ARIMA: (2, 1, 0) >> AIC : 4238.56
ARIMA: (2, 1, 1) >> AIC : 4240.4
ARIMA: (2, 1, 2) >> AIC : 4194.67
```

In [215]:
```
# Search optimal parameters

optimal = [(pdq[i], j) for i, j in enumerate(aic) if j == min(aic)]
optimal
```

Out[215]:
```
[((2, 1, 2), 4194.67)]
```

In [218]:
```
# 위 최적 값으로 만든 모델 다시 Summary

model_opt = ARIMA(train_data.values, order = optimal[0][0])
model_opt_fit = model_opt.fit()
model_opt_fit.summary()

# AIC score가 1045.66으로 임의의 모델보다 성능이 좋아졌고, p-value도 모두 유의미하게 나
옴
```

Out[218]:

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	363
Model:	ARIMA(2, 1, 2)	**Log Likelihood**	-2091.334
Method:	css-mle	**S.D. of innovations**	76.390
Date:	Wed, 09 Dec 2020	**AIC**	4194.669
Time:	23:12:34	**BIC**	4218.035
Sample:	1	**HQIC**	4203.957

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.0155	0.154	0.101	0.920	-0.286	0.317
ar.L1.D.y	0.8791	0.097	9.048	0.000	0.689	1.070
ar.L2.D.y	-0.2176	0.088	-2.465	0.014	-0.391	-0.045
ma.L1.D.y	-0.6179	0.094	-6.608	0.000	-0.801	-0.435
ma.L2.D.y	-0.3821	0.093	-4.096	0.000	-0.565	-0.199

Roots

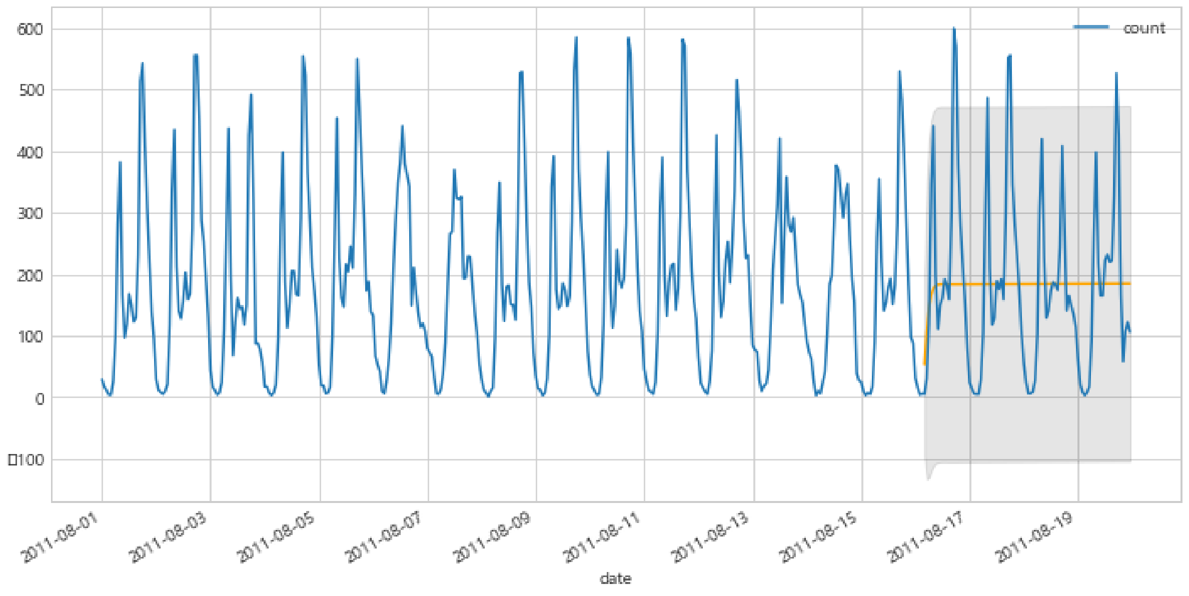
	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	2.0203	-0.7174j	2.1439	-0.0543
AR.2	2.0203	+0.7174j	2.1439	0.0543
MA.1	1.0000	+0.0000j	1.0000	0.0000
MA.2	-2.6172	+0.0000j	2.6172	0.5000

Test 데이터 예측

In [219]:
```
prediction = model_opt_fit.forecast(len(test_data))
predicted_value = prediction[0]   # predicted_value 가 y_pred
predicted_ub = prediction[2][:,0]
predicted_lb = prediction[2][:,1]
predict_index = list(test_data.index)
r2 = r2_score(test_data, predicted_value)
```

In [222]:
```
fig, ax = plt.subplots(figsize=(12,6))

ax.plot(predict_index, predicted_value, color = 'orange', label = 'Prediction'
) # 예측값(위 vline 이후 구간에 표시됨)
ax.fill_between(predict_index, predicted_lb, predicted_ub, color = 'k', alpha
= 0.1, label = '0.95 Prediction Interval')

data.plot(ax = ax);
# ax.vlines('1958-08-01', 0, 1000, linestyle = '--', color = 'r', label = 'Sta
rt of Forecast') # x좌표를 날짜로 적음
# ax.legend(loc='upper left')
# plt.suptitle(f'ARIMA {optimal[0][0]} Prediction Results (r2_score: {round(r
2,2)})')

plt.show()

# 빨간 점선 이후의 주황색 선이 예측값이며, 회색 구간이 95% interval 구간이다.
# 대체로 추세를 따라가나 피크 값을 완벽히 예측하기에는 다소 무리가 있는 것을 볼 수 있다.
# R2 score도 0.22 수준인 것을 확인할 수 있었다.
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:238: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:201: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0, flags=flags)
```



성능 평가

In [160]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

R^2

```
In [223]: r2 = r2_score(test_data, predicted_value)
          r2
```

```
Out[223]: 0.019727875700680686
```

MSE

```
In [224]: mse = mean_squared_error(test_data, predicted_value)
          mse
```

```
Out[224]: 20910.821211736336
```

RMSE

```
In [225]: rmse = np.sqrt(mse)
          rmse
```

```
Out[225]: 144.60574404820971
```

MAE

```
In [226]: mae = mean_absolute_error(test_data, predicted_value)
          mae
```

```
Out[226]: 108.6861962191569
```

MAPE

```
In [165]: 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 [227]: mape = mp(test_data, predicted_value)
          mape
```

```
Out[227]: 441.99048769365373
```

데이터 생성

In [244]:
```
# len(test_data) 자리에 원하는 만큼 숫자 넣어주면 됨

prediction = model_opt_fit.forecast(len(test_data))
predicted_value = prediction[0]   # predicted_value 가 y_pred
predicted_ub = prediction[2][:,0]
predicted_lb = prediction[2][:,1]
predict_index = list(test_data.index)
r2 = r2_score(test_data, predicted_value)
```

Out[244]:
```
Timestamp('2011-08-19 23:00:00')
```

(참고) 계절성 반영 시 SARIMA 모델링 수행

In [230]:
```
print('Examples of parameter combinations for Seasonal ARIMA...')
p = range(0,3)
d = range(1,2)
q = range(0,3)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 7 # 계절성이 7일마다 있다고 생각해서 7 입력. 얘도
 위에 range로 탐색해도 됨
                ) for x in list(itertools.product(p, d, q))]
seasonal_pdq
```

```
Examples of parameter combinations for Seasonal ARIMA...
```

Out[230]:
```
[(0, 1, 0, 7),
 (0, 1, 1, 7),
 (0, 1, 2, 7),
 (1, 1, 0, 7),
 (1, 1, 1, 7),
 (1, 1, 2, 7),
 (2, 1, 0, 7),
 (2, 1, 1, 7),
 (2, 1, 2, 7)]
```

In [231]:
```
aic = []
params = []
for i in pdq:
    for j in seasonal_pdq:
        try:
            model = SARIMAX(train_data.values, order=(i), seasonal_order = (j
))
            model_fit = model.fit()
            print(f'SARIMA: {i}{j} >> AIC : {round(model_fit.aic,2)}')
            aic.append(round(model_fit.aic,2))
            params.append((i, j))
        except:
            continue
```

```
SARIMA: (0, 1, 0)(0, 1, 0, 7) >> AIC : 4572.25
SARIMA: (0, 1, 0)(0, 1, 1, 7) >> AIC : 4261.5
SARIMA: (0, 1, 0)(0, 1, 2, 7) >> AIC : 4236.78
SARIMA: (0, 1, 0)(1, 1, 0, 7) >> AIC : 4346.58
SARIMA: (0, 1, 0)(1, 1, 1, 7) >> AIC : 4230.88
SARIMA: (0, 1, 0)(1, 1, 2, 7) >> AIC : 4227.12
SARIMA: (0, 1, 0)(2, 1, 0, 7) >> AIC : 4324.8
SARIMA: (0, 1, 0)(2, 1, 1, 7) >> AIC : 4228.58
SARIMA: (0, 1, 0)(2, 1, 2, 7) >> AIC : 4228.98
SARIMA: (0, 1, 1)(0, 1, 0, 7) >> AIC : 4472.44
SARIMA: (0, 1, 1)(0, 1, 1, 7) >> AIC : 4197.85
SARIMA: (0, 1, 1)(0, 1, 2, 7) >> AIC : 4190.14
SARIMA: (0, 1, 1)(1, 1, 0, 7) >> AIC : 4286.48
SARIMA: (0, 1, 1)(1, 1, 1, 7) >> AIC : 4187.24
SARIMA: (0, 1, 1)(1, 1, 2, 7) >> AIC : 4196.09
SARIMA: (0, 1, 1)(2, 1, 0, 7) >> AIC : 4268.11
SARIMA: (0, 1, 1)(2, 1, 1, 7) >> AIC : 4182.24

C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmo
dels\base\model.py:568: ConvergenceWarning: Maximum Likelihood optimization f
ailed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)

SARIMA: (0, 1, 1)(2, 1, 2, 7) >> AIC : 4189.68
SARIMA: (0, 1, 2)(0, 1, 0, 7) >> AIC : 4474.17
SARIMA: (0, 1, 2)(0, 1, 1, 7) >> AIC : 4197.96
SARIMA: (0, 1, 2)(0, 1, 2, 7) >> AIC : 4188.07
SARIMA: (0, 1, 2)(1, 1, 0, 7) >> AIC : 4288.25
SARIMA: (0, 1, 2)(1, 1, 1, 7) >> AIC : 4185.4
SARIMA: (0, 1, 2)(1, 1, 2, 7) >> AIC : 4194.15
SARIMA: (0, 1, 2)(2, 1, 0, 7) >> AIC : 4270.03
SARIMA: (0, 1, 2)(2, 1, 1, 7) >> AIC : 4182.81
SARIMA: (0, 1, 2)(2, 1, 2, 7) >> AIC : 4186.93
SARIMA: (1, 1, 0)(0, 1, 0, 7) >> AIC : 4519.98
SARIMA: (1, 1, 0)(0, 1, 1, 7) >> AIC : 4224.81
SARIMA: (1, 1, 0)(0, 1, 2, 7) >> AIC : 4210.56
SARIMA: (1, 1, 0)(1, 1, 0, 7) >> AIC : 4297.35
SARIMA: (1, 1, 0)(1, 1, 1, 7) >> AIC : 4205.15
SARIMA: (1, 1, 0)(1, 1, 2, 7) >> AIC : 4220.77
SARIMA: (1, 1, 0)(2, 1, 0, 7) >> AIC : 4282.27
SARIMA: (1, 1, 0)(2, 1, 1, 7) >> AIC : 4197.44

C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmo
dels\base\model.py:568: ConvergenceWarning: Maximum Likelihood optimization f
ailed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```

```
SARIMA: (1, 1, 0)(2, 1, 2, 7) >> AIC : 4207.81
SARIMA: (1, 1, 1)(0, 1, 0, 7) >> AIC : 4474.34
SARIMA: (1, 1, 1)(0, 1, 1, 7) >> AIC : 4198.97
SARIMA: (1, 1, 1)(0, 1, 2, 7) >> AIC : 4190.22
SARIMA: (1, 1, 1)(1, 1, 0, 7) >> AIC : 4288.31
SARIMA: (1, 1, 1)(1, 1, 1, 7) >> AIC : 4187.32
SARIMA: (1, 1, 1)(1, 1, 2, 7) >> AIC : 4196.35
SARIMA: (1, 1, 1)(2, 1, 0, 7) >> AIC : 4270.07
SARIMA: (1, 1, 1)(2, 1, 1, 7) >> AIC : 4183.45
SARIMA: (1, 1, 1)(2, 1, 2, 7) >> AIC : 4189.44
SARIMA: (1, 1, 2)(0, 1, 0, 7) >> AIC : 4421.86
SARIMA: (1, 1, 2)(0, 1, 1, 7) >> AIC : 4150.66
SARIMA: (1, 1, 2)(0, 1, 2, 7) >> AIC : 4141.32
SARIMA: (1, 1, 2)(1, 1, 0, 7) >> AIC : 4287.23
SARIMA: (1, 1, 2)(1, 1, 1, 7) >> AIC : 4139.87
SARIMA: (1, 1, 2)(1, 1, 2, 7) >> AIC : 4162.61
SARIMA: (1, 1, 2)(2, 1, 0, 7) >> AIC : 4226.17
SARIMA: (1, 1, 2)(2, 1, 1, 7) >> AIC : 4140.72

C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmo
dels\base\model.py:568: ConvergenceWarning: Maximum Likelihood optimization f
ailed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)

SARIMA: (1, 1, 2)(2, 1, 2, 7) >> AIC : 4142.36
SARIMA: (2, 1, 0)(0, 1, 0, 7) >> AIC : 4449.53
SARIMA: (2, 1, 0)(0, 1, 1, 7) >> AIC : 4183.9
SARIMA: (2, 1, 0)(0, 1, 2, 7) >> AIC : 4177.57
SARIMA: (2, 1, 0)(1, 1, 0, 7) >> AIC : 4284.58
SARIMA: (2, 1, 0)(1, 1, 1, 7) >> AIC : 4175.94
SARIMA: (2, 1, 0)(1, 1, 2, 7) >> AIC : 4182.55
SARIMA: (2, 1, 0)(2, 1, 0, 7) >> AIC : 4256.46
SARIMA: (2, 1, 0)(2, 1, 1, 7) >> AIC : 4175.27
SARIMA: (2, 1, 0)(2, 1, 2, 7) >> AIC : 4179.2
SARIMA: (2, 1, 1)(0, 1, 0, 7) >> AIC : 4450.49
SARIMA: (2, 1, 1)(0, 1, 1, 7) >> AIC : 4185.62
SARIMA: (2, 1, 1)(0, 1, 2, 7) >> AIC : 4145.43
SARIMA: (2, 1, 1)(1, 1, 0, 7) >> AIC : 4245.92
SARIMA: (2, 1, 1)(1, 1, 1, 7) >> AIC : 4142.92
SARIMA: (2, 1, 1)(1, 1, 2, 7) >> AIC : 4163.12

C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmo
dels\base\model.py:568: ConvergenceWarning: Maximum Likelihood optimization f
ailed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)

SARIMA: (2, 1, 1)(2, 1, 0, 7) >> AIC : 4220.79
SARIMA: (2, 1, 1)(2, 1, 1, 7) >> AIC : 4142.27
SARIMA: (2, 1, 1)(2, 1, 2, 7) >> AIC : 4142.98
SARIMA: (2, 1, 2)(0, 1, 0, 7) >> AIC : 4388.5
SARIMA: (2, 1, 2)(0, 1, 1, 7) >> AIC : 4181.84
SARIMA: (2, 1, 2)(0, 1, 2, 7) >> AIC : 4170.11

C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmo
dels\base\model.py:568: ConvergenceWarning: Maximum Likelihood optimization f
ailed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```

```
SARIMA: (2, 1, 2)(1, 1, 0, 7) >> AIC : 4246.16
SARIMA: (2, 1, 2)(1, 1, 1, 7) >> AIC : 4165.27
SARIMA: (2, 1, 2)(1, 1, 2, 7) >> AIC : 4152.32
SARIMA: (2, 1, 2)(2, 1, 0, 7) >> AIC : 4253.74
SARIMA: (2, 1, 2)(2, 1, 1, 7) >> AIC : 4153.63
SARIMA: (2, 1, 2)(2, 1, 2, 7) >> AIC : 4139.82
```

In [232]:
```
# Search optimal parameters

optimal = [(params[i], j) for i, j in enumerate(aic) if j == min(aic)]
optimal

# small pdq는 (1,1,0), large pdq는 (1,1,2) 그리고 Seasonal parameter는 12인 것을
 볼 수 있다.
```

Out[232]:
```
[(((2, 1, 2), (2, 1, 2, 7)), 4139.82)]
```

In [233]:
```
model_opt = SARIMAX(train_data.values, order=optimal[0][0][0], seasonal_order
= optimal[0][0][1])
model_opt_fit = model_opt.fit()
model_opt_fit.summary()

# ARIMA보다 SARIMA가 AIC가 훨씬 낮은 것을 볼 수 있다.
```

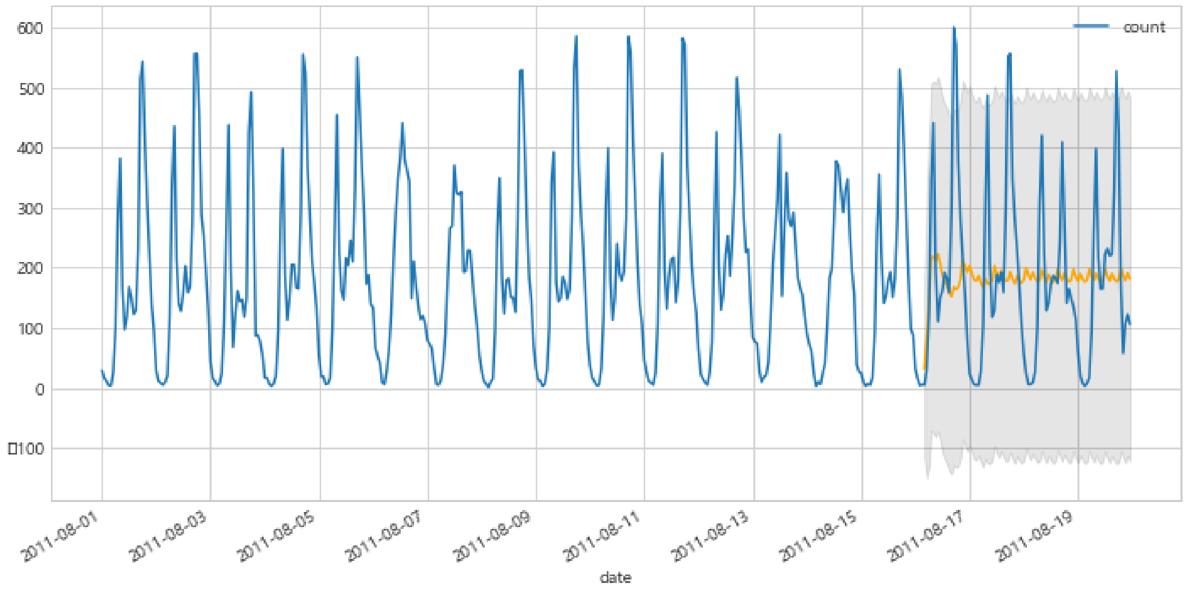
Out[233]:
```
SARIMAX Results

   Dep. Variable:                          y   No. Observations:        364
           Model:  SARIMAX(2, 1, 2)x(2, 1, 2, 7)   Log Likelihood  -2060.909
            Date:           Wed, 09 Dec 2020              AIC    4139.819
            Time:                   23:24:07              BIC    4174.693
          Sample:                          0             HQIC    4153.691
                                       - 364
Covariance Type:                         opg

                coef   std err       z   P>|z|    [0.025    0.975]
     ar.L1    0.9201     0.135   6.834   0.000     0.656     1.184
     ar.L2   -0.2209     0.131  -1.682   0.092    -0.478     0.036
     ma.L1   -0.6882     2.654  -0.259   0.795    -5.889     4.513
     ma.L2   -0.3115     0.873  -0.357   0.721    -2.023     1.400
   ar.S.L7   -0.6366     0.572  -1.112   0.266    -1.759     0.485
  ar.S.L14   -0.0065     0.146  -0.045   0.964    -0.293     0.279
   ma.S.L7   -0.5230     0.777  -0.674   0.501    -2.045     0.999
  ma.S.L14   -0.4729     0.587  -0.806   0.420    -1.623     0.677
    sigma2 5610.3109  1.45e+04   0.386   0.700  -2.29e+04  3.41e+04

         Ljung-Box (Q):  235.96  Jarque-Bera (JB):  124.42
               Prob(Q):    0.00          Prob(JB):    0.00
Heteroskedasticity (H):    0.77              Skew:    0.73
  Prob(H) (two-sided):     0.16          Kurtosis:    5.50


Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Test data 예측 및 성능평가는 위와 동일

In [234]:
```
prediction = model_opt_fit.get_forecast(len(test_data))
predicted_value = prediction.predicted_mean
predicted_ub = prediction.conf_int()[:,0]
predicted_lb = prediction.conf_int()[:,1]
predict_index = list(test_data.index)
r2 = r2_score(test_data, predicted_value)
```

In [236]:
```
fig, ax = plt.subplots(figsize=(12,6))

ax.plot(predict_index, predicted_value, color = 'orange', label = 'Prediction'
) # 예측값(위 vline 이후 구간에 표시됨)
ax.fill_between(predict_index, predicted_lb, predicted_ub, color = 'k', alpha
= 0.1, label = '0.95 Prediction Interval')

data.plot(ax = ax);
# ax.vlines('1958-08-01', 0, 700, linestyle = '--', color = 'r', label = 'Star
t of Forecast') # x좌표를 날짜로 적음
# ax.legend(loc='upper left')
# plt.suptitle(f'SARIMA {optimal[0][0][0]},{optimal[0][0][1]} Prediction Resul
ts (r2_score: {round(r2,2)})')

plt.show()

# 예측 값의 추세가 실제 값을 상당히 잘 따라가고 있으며,
# r2 score가 0.89로 훨씬 더 성능이 향상됨
# 계절성을 반영한 것이 예측 성능을 향상시키는데 기여를 했다고 볼 수 있다.
```

```
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:238: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\50008313\AppData\Local\Continuum\anaconda3\lib\site-packages\matplot
lib\backends\backend_agg.py:201: RuntimeWarning: Glyph 8722 missing from curr
ent font.
  font.set_text(s, 0, flags=flags)
```



Test data 예측 및 성능평가는 위와 동일