**# # 패키지**

import numpy as np

import pandas as pd

import json

import datetime as dt

import matplotlib.pyplot as plt

import matplotlib.ticker as ticker

get\_ipython().run\_line\_magic('matplotlib', 'inline')

import seaborn as sns

plt.style.use('seaborn-whitegrid')

# import missingno

import re

import glob

import os

from scipy import stats

from scipy.integrate import trapz

# import missingno as msno

import sys

import warnings

warnings.filterwarnings('ignore')

pd.set\_option('max\_columns', 20, 'max\_rows', 20, 'max\_colwidth', 20)

# 알고리즘(시계열)

import statsmodels.api as sm

from scipy import stats

import pmdarima as pm

from pmdarima import model\_selection

# 전처리

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.preprocessing import Binarizer

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

# Model selection

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import TimeSeriesSplit

from sklearn.metrics import make\_scorer

# 차원축소

from sklearn.decomposition import PCA

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

## 알고리즘(회귀)

from sklearn import linear\_model

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.neural\_network import MLPRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.svm import SVR

from sklearn.linear\_model import LinearRegression

from sklearn.linear\_model import Ridge, Lasso, ElasticNet

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

import statsmodels.api as sm

# Evaluation metrics

# for regression

from sklearn.metrics import mean\_squared\_log\_error, mean\_squared\_error, r2\_score, mean\_absolute\_error

from sklearn.metrics import r2\_score

#

from matplotlib.patches import Patch

from itertools import product

from tqdm import tqdm

**# # 데이터 확인**

**# # 전처리**

**# ## 시계열 패턴 추출**

## Functinalize

### Feature engineering of default

def non\_feature\_engineering(raw):

raw\_nfe = raw.copy()

if 'datetime' in raw\_nfe.columns:

raw\_nfe['datetime'] = pd.to\_datetime(raw\_nfe['datetime'])

raw\_nfe['DateTime'] = pd.to\_datetime(raw\_nfe['datetime'])

if raw\_nfe.index.dtype == 'int64':

raw\_nfe.set\_index('DateTime', inplace=True)

# bring back

# if raw\_nfe.index.dtype != 'int64':

# raw\_nfe.reset\_index(drop=False, inplace=True)

raw\_nfe = raw\_nfe.asfreq('H', method='ffill')

return raw\_nfe

# raw\_rd = non\_feature\_engineering(raw\_all)

### Feature engineering of all

def feature\_engineering(raw):

raw\_fe = raw.copy()

if 'datetime' in raw\_fe.columns:

raw\_fe['datetime'] = pd.to\_datetime(raw\_fe['datetime'])

raw\_fe['DateTime'] = pd.to\_datetime(raw\_fe['datetime'])

if raw\_fe.index.dtype == 'int64':

raw\_fe.set\_index('DateTime', inplace=True)

raw\_fe = raw\_fe.asfreq('H', method='ffill')

result = sm.tsa.seasonal\_decompose(raw\_fe['count'], model='additive')

Y\_trend = pd.DataFrame(result.trend)

Y\_trend.fillna(method='ffill', inplace=True)

Y\_trend.fillna(method='bfill', inplace=True)

Y\_trend.columns = ['count\_trend']

Y\_seasonal = pd.DataFrame(result.seasonal)

Y\_seasonal.fillna(method='ffill', inplace=True)

Y\_seasonal.fillna(method='bfill', inplace=True)

Y\_seasonal.columns = ['count\_seasonal']

pd.concat([raw\_fe, Y\_trend, Y\_seasonal], axis=1).isnull().sum()

if 'count\_trend' not in raw\_fe.columns:

if 'count\_seasonal' not in raw\_fe.columns:

raw\_fe = pd.concat([raw\_fe, Y\_trend, Y\_seasonal], axis=1)

Y\_count\_Day = raw\_fe[['count']].rolling(24).mean()

Y\_count\_Day.fillna(method='ffill', inplace=True)

Y\_count\_Day.fillna(method='bfill', inplace=True)

Y\_count\_Day.columns = ['count\_Day']

Y\_count\_Week = raw\_fe[['count']].rolling(24\*7).mean()

Y\_count\_Week.fillna(method='ffill', inplace=True)

Y\_count\_Week.fillna(method='bfill', inplace=True)

Y\_count\_Week.columns = ['count\_Week']

if 'count\_Day' not in raw\_fe.columns:

raw\_fe = pd.concat([raw\_fe, Y\_count\_Day], axis=1)

if 'count\_Week' not in raw\_fe.columns:

raw\_fe = pd.concat([raw\_fe, Y\_count\_Week], axis=1)

Y\_diff = raw\_fe[['count']].diff()

Y\_diff.fillna(method='ffill', inplace=True)

Y\_diff.fillna(method='bfill', inplace=True)

Y\_diff.columns = ['count\_diff']

if 'count\_diff' not in raw\_fe.columns:

raw\_fe = pd.concat([raw\_fe, Y\_diff], axis=1)

raw\_fe['temp\_group'] = pd.cut(raw\_fe['temp'], 10)

raw\_fe['Year'] = raw\_fe.datetime.dt.year

raw\_fe['Quater'] = raw\_fe.datetime.dt.quarter

raw\_fe['Quater\_ver2'] = raw\_fe['Quater'] + (raw\_fe.Year - raw\_fe.Year.min()) \* 4

raw\_fe['Month'] = raw\_fe.datetime.dt.month

raw\_fe['Day'] = raw\_fe.datetime.dt.day

raw\_fe['Hour'] = raw\_fe.datetime.dt.hour

raw\_fe['DayofWeek'] = raw\_fe.datetime.dt.dayofweek

raw\_fe['count\_lag1'] = raw\_fe['count'].shift(1)

raw\_fe['count\_lag2'] = raw\_fe['count'].shift(2)

raw\_fe['count\_lag1'].fillna(method='bfill', inplace=True)

raw\_fe['count\_lag2'].fillna(method='bfill', inplace=True)

if 'Quater' in raw\_fe.columns:

if 'Quater\_Dummy' not in ['\_'.join(col.split('\_')[:2]) for col in raw\_fe.columns]:

raw\_fe = pd.concat([raw\_fe, pd.get\_dummies(raw\_fe['Quater'], prefix='Quater\_Dummy', drop\_first=True)], axis=1)

del raw\_fe['Quater']

return raw\_fe

# raw\_fe = feature\_engineering(raw\_all)

**# ## 함수 없이 전처리**

# 시간 컬럼 datetime으로 변환

raw\_df['datetime'] = pd.to\_datetime(raw\_df['datetime'])

raw\_df['DateTime'] = pd.to\_datetime(raw\_df['datetime'])

# 시간 컬럼 index로 설정

raw\_df.set\_index('DateTime', inplace=True)

# 시계열 축 시간 단위로 설정하여 null값 시간이 없는지 확인

raw\_df = raw\_df.asfreq('H', method='ffill')

# 트랜드, 계절성을 additive방식으로 뽑아 컬럼으로 만들기

result = sm.tsa.seasonal\_decompose(raw\_df['count'], model='additive')

Y\_trend = pd.DataFrame(result.trend)

Y\_trend.fillna(method='ffill', inplace=True)

Y\_trend.fillna(method='bfill', inplace=True)

Y\_trend.columns = ['count\_trend']

Y\_seasonal = pd.DataFrame(result.seasonal)

Y\_seasonal.fillna(method='ffill', inplace=True)

Y\_seasonal.fillna(method='bfill', inplace=True)

Y\_seasonal.columns = ['count\_seasonal']

pd.concat([raw\_df, Y\_trend, Y\_seasonal], axis=1).isnull().sum()

# Moving Average(MA)를 하루(24시간), 일주일(24시간 \* 7)로 하여 컬럼으로 만들기

Y\_count\_Day = raw\_df[['count']].rolling(24).mean()

Y\_count\_Day.fillna(method='ffill', inplace=True)

Y\_count\_Day.fillna(method='bfill', inplace=True)

Y\_count\_Day.columns = ['count\_Day']

Y\_count\_Week = raw\_df[['count']].rolling(24\*7).mean()

Y\_count\_Week.fillna(method='ffill', inplace=True)

Y\_count\_Week.fillna(method='bfill', inplace=True)

Y\_count\_Week.columns = ['count\_Week']

# 전 행 데이터와의 차이(차분)를 컬럼으로 만들기

Y\_diff = raw\_df[['count']].diff()

Y\_diff.fillna(method='ffill', inplace=True)

Y\_diff.fillna(method='bfill', inplace=True)

Y\_diff.columns = ['count\_diff']

# 추가 분류 컬럼과 시간관련 컬럼으로 만들기

raw\_df['temp\_group'] = pd.cut(raw\_df['temp'], 10)

raw\_df['Year'] = raw\_df.datetime.dt.year

raw\_df['Quater'] = raw\_df.datetime.dt.quarter

raw\_df['Quater\_ver2'] = raw\_df['Quater'] + (raw\_df.Year - raw\_df.Year.min()) \* 4

raw\_df['Month'] = raw\_df.datetime.dt.month

raw\_df['Day'] = raw\_df.datetime.dt.day

raw\_df['Hour'] = raw\_df.datetime.dt.hour

raw\_df['DayofWeek'] = raw\_df.datetime.dt.dayofweek

# lag 값 컬럼으로 만들기

raw\_df['count\_lag1'] = raw\_df['count'].shift(1)

raw\_df['count\_lag2'] = raw\_df['count'].shift(2)

raw\_df['count\_lag1'].fillna(method='bfill', inplace=True)

raw\_df['count\_lag2'].fillna(method='bfill', inplace=True)

# 분기 값 one-hot-incoding 하기

raw\_df = pd.concat([raw\_df, pd.get\_dummies(raw\_df['Quater'], prefix='Quater\_Dummy', drop\_first=True)], axis=1)

del raw\_df['Quater']

**# ## 시간현실반영(미완) - 실제론 안쓸 것 같음**

### Functionalize

### duplicate previous year values to next one

def feature\_engineering\_year\_duplicated(raw, target):

raw\_fe = raw.copy()

for col in target:

raw\_fe.loc['2012-01-01':'2012-02-28', col] = raw.loc['2011-01-01':'2011-02-28', col].values

raw\_fe.loc['2012-03-01':'2012-12-31', col] = raw.loc['2011-03-01':'2011-12-31', col].values

step = (raw.loc['2011-03-01 00:00:00', col] - raw.loc['2011-02-28 23:00:00', col])/25

step\_value = np.arange(raw.loc['2011-02-28 23:00:00', col]+step, raw.loc['2011-03-01 00:00:00', col], step)

step\_value = step\_value[:24]

raw\_fe.loc['2012-02-29', col] = step\_value

return raw\_fe

# target = ['count\_trend', 'count\_seasonal', 'count\_Day', 'count\_Week', 'count\_diff']

# raw\_fe = feature\_engineering\_year\_duplicated(raw\_fe, target)

### modify lagged values of X\_test

def feature\_engineering\_lag\_modified(Y\_test, X\_test, target):

X\_test\_lm = X\_test.copy()

for col in target:

X\_test\_lm[col] = Y\_test.shift(1).values

X\_test\_lm[col].fillna(method='bfill', inplace=True)

X\_test\_lm[col] = Y\_test.shift(2).values

X\_test\_lm[col].fillna(method='bfill', inplace=True)

return X\_test\_lm

# target = ['count\_lag1', 'count\_lag2']

# X\_test\_fe = feature\_engineering\_lag\_modified(Y\_test\_fe, X\_test\_fe, target)

raw\_fe.loc['2012-01-01':'2012-02-28', col] = raw\_df.loc['2011-01-01':'2011-02-28', col].values

raw\_fe.loc['2012-03-01':'2012-12-31', col] = raw\_df.loc['2011-03-01':'2011-12-31', col].values

step = (raw\_df.loc['2011-03-01 00:00:00', col] - raw\_df.loc['2011-02-28 23:00:00', col])/25

step\_value = np.arange(raw\_df.loc['2011-02-28 23:00:00', col]+step, raw\_df.loc['2011-03-01 00:00:00', col], step)

step\_value = step\_value[:24]

raw\_fe.loc['2012-02-29', col] = step\_value

### modify lagged values of X\_test

X\_test\_lm[col] = y\_test.shift(1).values

X\_test\_lm[col].fillna(method='bfill', inplace=True)

X\_test\_lm[col] = y\_test.shift(2).values

X\_test\_lm[col].fillna(method='bfill', inplace=True)

X\_test\_lm[col] = Y\_test.shift(1).values

X\_test\_lm[col].fillna(method='bfill', inplace=True)

X\_test\_lm[col] = Y\_test.shift(2).values

X\_test\_lm[col].fillna(method='bfill', inplace=True)

**# ## 데이터 분리**

**# ### 불필요 컬럼 제거 및 target 컬럼 분리**

# Confirm of input and output

y\_colname = ['count']

X\_remove = ['datetime', 'temp\_group', 'casual', 'registered']

X\_colname = [x for x in raw\_df.columns if x not in y\_colname+X\_remove]

len(X\_colname)

# 전처리가 모두 끝나고 scaling은 하지 않은 상태에서 데이터 분리를 시행한다.

X\_features = raw\_df.drop(X\_remove, axis=1)

y\_target = raw\_df[y\_colname]

X\_features.shape[0]/raw\_df.shape[0]

y\_target.shape[0]/raw\_df.shape[0]

print('- X\_features 컬럼 :',X\_features.columns)

print('- y\_target 컬럼 :',y\_target.columns)

**# ### 시간으로 train, test 나누기**

raw\_train = raw\_df.loc[raw\_df.index < '2012-07-01',:]

raw\_test = raw\_df.loc[raw\_df.index >= '2012-07-01',:]

y\_train = raw\_train[y\_colname]

X\_train = raw\_train[X\_colname]

y\_test = raw\_test[y\_colname]

X\_test = raw\_test[X\_colname]

print('Train\_size:', raw\_train.shape, 'Test\_size:', raw\_test.shape)

print('X\_train:', X\_train.shape, 'y\_train:', y\_train.shape)

print('X\_test:', X\_test.shape, 'y\_test:', y\_test.shape)

**# ### 시계열 교차검증**

cmap\_data = plt.cm.Paired

cmap\_cv = plt.cm.coolwarm

plt.style.use('fivethirtyeight')

def plot\_cv\_indices(cv, X, n\_splits, lw=10):

fig, ax = plt.subplots()

"""Create a sample plot for indices of a cross-validation object."""

# Generate the training/testing visualizations for each CV split

for ii, (tr, tt) in enumerate(cv.split(X=X)):

# Fill in indices with the training/test groups

indices = np.array([np.nan] \* len(X))

indices[tt] = 1

indices[tr] = 0

# Visualize the results

ax.scatter(range(len(indices)), [ii + .5] \* len(indices),

c=indices, marker='\_', lw=lw, cmap=cmap\_cv,

vmin=-.2, vmax=1.2)

# Formatting

yticklabels = list(range(n\_splits))

ax.set(yticks=np.arange(n\_splits) + .5, yticklabels=yticklabels,

xlabel='Sample index', ylabel="CV iteration",

ylim=[n\_splits+0.1, -.1], xlim=[0, len(X)])

ax.set\_title('{}'.format(type(cv).\_\_name\_\_), fontsize=15)

ax.legend([Patch(color=cmap\_cv(.8)), Patch(color=cmap\_cv(.02))],

['Testing set', 'Training set'], loc=(1.02, .8))

# plotting with a simple array data

n\_split = 6

tscv = TimeSeriesSplit(n\_splits=n\_split)

print(tscv)

for train, test in tscv.split(raw\_df):

print("%s %s" % (train, test))

plot\_cv\_indices(tscv, raw\_df, n\_splits=n\_split)

from sklearn.linear\_model import LinearRegression

from sklearn.neural\_network import MLPRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.metrics import r2\_score

# Spot Check Algorithms

models = []

models.append(('LR', LinearRegression()))

models.append(('NN', MLPRegressor(solver = 'lbfgs'))) #neural network

models.append(('KNN', KNeighborsRegressor()))

models.append(('RF', RandomForestRegressor(n\_estimators = 10)))

# Ensemble method - collection of many decision trees

models.append(('SVR', SVR(gamma='auto'))) # kernel = linear

# Evaluate each model in turn

results = []

names = []

for name, model in models:

# TimeSeries Cross validation

tscv = TimeSeriesSplit(n\_splits=10)

# score = 'neg\_mean\_squared\_error'

cv\_results = cross\_val\_score(model, X\_train, y\_train, cv=tscv, scoring='neg\_mean\_squared\_error')

results.append(cv\_results)

names.append(name)

print('%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std()))

# Compare Algorithms

plt.boxplot(results, labels=names)

plt.title('Algorithm Comparison')

plt.show()

**# ### 블럭 교차 검증**

class BlockingTimeSeriesSplit():

def \_\_init\_\_(self, n\_splits):

self.n\_splits = n\_splits

def get\_n\_splits(self, groups):

return self.n\_splits

def split(self, X, y=None, groups=None):

n\_samples = len(X)

k\_fold\_size = n\_samples // self.n\_splits

indices = np.arange(n\_samples)

margin = 0

for i in range(self.n\_splits):

start = i \* k\_fold\_size

stop = start + k\_fold\_size

mid = int(0.8 \* (stop - start)) + start

yield indices[start: mid], indices[mid + margin: stop]

n\_split = 6

btscv = BlockingTimeSeriesSplit(n\_splits=n\_split)

for train, test in btscv.split(raw\_df):

print("%s %s" % (train, test))

plot\_cv\_indices(btscv, raw\_df, n\_splits=n\_split)

# Spot Check Algorithms

models = []

models.append(('LR', LinearRegression()))

models.append(('NN', MLPRegressor(solver = 'lbfgs', max\_iter=2000))) #neural network

models.append(('KNN', KNeighborsRegressor()))

models.append(('RF', RandomForestRegressor(n\_estimators = 10)))

# Ensemble method - collection of many decision trees

models.append(('SVR', SVR(gamma='auto'))) # kernel = linear

# Evaluate each model in turn

results = []

names = []

for name, model in models:

# blocked Cross-validation

btscv = BlockingTimeSeriesSplit(n\_splits=10)

cv\_results = cross\_val\_score(model, X\_train, y\_train, cv=btscv, scoring='r2')

results.append(cv\_results)

names.append(name)

print('%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std()))

# Compare Algorithms

plt.boxplot(results, labels=names)

plt.title('Algorithm Comparison (blocked CV)')

plt.show()

**# ## 범주 데이터 수치화**

**# ### get\_dummies**

col\_list = []

col\_ohe = pd.get\_dummies(X\_train[col\_list])

X\_train\_ohe = pd.concat([X\_train, col\_ohe], axis=1)

X\_train = X\_train\_ohe.drop(col\_list, axis=1)

**# ### Label Incoding**

from sklearn.preprocessing import LabelEncoder

col\_list=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']

# LabelEncoder를 객체로 생성한 후 , fit( ) 과 transform( ) 으로 label 인코딩 수행.

encoder = LabelEncoder()

encoder.fit(col\_list)

labels = encoder.transform(col\_list)

print('인코딩 변환값:',labels)

print('인코딩 클래스:',encoder.classes\_)

print('디코딩 원본 값:',encoder.inverse\_transform([4, 5, 2, 0, 1, 1, 3, 3]))

**# ### one-hot Incoding**

from sklearn.preprocessing import OneHotEncoder

import numpy as np

col\_list=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']

# 먼저 숫자값으로 변환을 위해 LabelEncoder로 변환합니다.

encoder = LabelEncoder()

encoder.fit(col\_list)

labels = encoder.transform(col\_list)

# 2차원 데이터로 변환합니다.

labels = labels.reshape(-1,1)

# 원-핫 인코딩을 적용합니다.

oh\_encoder = OneHotEncoder()

oh\_encoder.fit(labels)

oh\_labels = oh\_encoder.transform(labels)

print('원-핫 인코딩 데이터')

print(oh\_labels.toarray())

print('원-핫 인코딩 데이터 차원')

print(oh\_labels.shape)

**# ## Scaler**

**# ### StandardScaler**

# X\_features scaling

scaler = StandardScaler()

scaler = scaler.fit(X\_train)

X\_train\_S = pd.DataFrame(scaler.transform(X\_train),

index=X\_train.index, columns=X\_train.columns)

X\_test\_S = pd.DataFrame(scaler.transform(X\_test),

index=X\_test.index, columns=X\_test.columns)

display(X\_test\_S.describe().T)

# y\_target scaling

scaler = StandardScaler()

scaler = scaler.fit(y\_train)

y\_train\_S = pd.DataFrame(scaler.transform(y\_train),

index=y\_train.index, columns=y\_train.columns)

y\_test\_S = pd.DataFrame(scaler.transform(y\_test),

index=y\_test.index, columns=y\_test.columns)

display(y\_test\_S.describe().T)

**# ### MinMaxScaler**

# X\_features scaling

scaler = MinMaxScaler()

scaler = scaler.fit(X\_train)

X\_train\_S = pd.DataFrame(scaler.transform(X\_train),

index=X\_train.index, columns=X\_train.columns)

X\_test\_S = pd.DataFrame(scaler.transform(X\_test),

index=X\_test.index, columns=X\_test.columns)

display(X\_test\_S.describe().T)

# y\_target scaling

scaler = MinMaxScaler()

scaler = scaler.fit(y\_train)

y\_train\_S = pd.DataFrame(scaler.transform(y\_train),

index=y\_train.index, columns=y\_train.columns)

y\_test\_S = pd.DataFrame(scaler.transform(y\_test),

index=y\_test.index, columns=y\_test.columns)

display(y\_test\_S.describe().T)

y\_train

**# ### Normalizer**

# X\_features scaling

scaler = Normalizer()

scaler = scaler.fit(X\_train)

X\_train\_S = pd.DataFrame(scaler.transform(X\_train),

index=X\_train.index, columns=X\_train.columns)

X\_test\_S = pd.DataFrame(scaler.transform(X\_test),

index=X\_test.index, columns=X\_test.columns)

display(X\_test\_S.describe().T)

# y\_target scaling

scaler = Normalizer()

scaler = scaler.fit(y\_train)

y\_train\_S = pd.DataFrame(scaler.transform(y\_train),

index=y\_train.index, columns=y\_train.columns)

y\_test\_S = pd.DataFrame(scaler.transform(y\_test),

index=y\_test.index, columns=y\_test.columns)

display(y\_test\_S.describe().T)

**# ### Log1p**

# X\_features scaling

X\_train\_S = pd.DataFrame(np.log1p(X\_train), index=X\_train.index, columns=X\_train.columns)

X\_test\_S = pd.DataFrame(np.log1p(X\_test), index=X\_test.index, columns=X\_test.columns)

display(X\_test\_S.describe().T)

# y\_target scaling

y\_train\_S = pd.DataFrame(np.log1p(y\_train), index=X\_train.index, columns=y\_train.columns)

y\_test\_S = pd.DataFrame(np.log1p(y\_test), index=y\_test.index, columns=y\_test.columns)

display(y\_test\_S.describe().T)

**# ## 다중공선성제거**

**# ### VIF**

- Vif factor는 X와 X관의 관계를 나타내는 것으로 높을 수록 안 좋다.

- 상관관계가 높은 컬럼들하고 겹치는 것들을 우선 사용한다.

- 해당 변수가 다른 변수와 전혀 상관 관계가 없다면 VIF = 1이고, 해당 변수의 R-squared 값은 0이다.

- 10 이상이면 서로 강한 상관관계를 보이는 것이라고 함

- 1 에서 10 미만의 값이면 다중공선성이 별 문제가 되지 않는 것으로 판단한다

- VIFk = 1 / (1 - Rj2)

- 결정계수 Rj2 값이 0에 가깝다는 것은 입력변수 k가 다른 입력변수들과 상관성이 거의 없다는 것을 의미하고,

- 결정계수 Rj2 값이 1에 가깝다는 것은 입력변수 k가 다른 입력변수들과 상관성, 즉 다중공선성이 크다는 것을 의미한다.

**# #### 상관도 확인**

corr = X\_features.corr()

plt.figure(figsize=(14,14))

sns.heatmap(corr, annot=True, fmt='.1g')

# correlation from features

raw\_df.corr().loc[X\_colname, X\_colname].style.background\_gradient().set\_precision(2).set\_properties(\*\*{'font-size': '11pt'})

corr = X\_features.corr().loc[:, ['count']]

corr = X\_features.corr().loc[:, ['count']].style.background\_gradient().set\_precision(2).set\_properties(\*\*{'font-size': '11pt'})

corr

raw\_df.describe()

for col in raw\_df.describe().columns:

target = raw\_df[col]

figure, axes = plt.subplots(2,1,figsize=(16,10))

sm.graphics.tsa.plot\_acf(target, lags=100, use\_vlines=True, ax=axes[0], title=col)

sm.graphics.tsa.plot\_pacf(target, lags=100, use\_vlines=True, ax=axes[1], title=col)

vif = pd.DataFrame()

vif['VIF\_Factor'] = [variance\_inflation\_factor(X\_train\_S.values, i)

for i in range(X\_train\_S.shape[1])]

vif['Feature'] = X\_train\_S.columns

vif.sort\_values(by='VIF\_Factor', ascending=True)

# extract effective features using variance inflation factor

vif = pd.DataFrame()

vif['VIF\_Factor'] = [variance\_inflation\_factor(X\_train\_S.values, i)

for i in range(X\_train\_S.shape[1])]

vif['Feature'] = X\_train\_S.columns

vif.sort\_values(by='VIF\_Factor', ascending=True)['Feature'][:10].values

**# #### 독립변수 일부 반영**

### Functionalize

### extract non-multicollinearity variables by VIF

def feature\_engineering\_XbyVIF(X\_train, num\_variables):

vif = pd.DataFrame()

vif['VIF\_Factor'] = [variance\_inflation\_factor(X\_train.values, i)

for i in range(X\_train.shape[1])]

vif['Feature'] = X\_train.columns

X\_colname\_vif = vif.sort\_values(by='VIF\_Factor', ascending=True)['Feature'][:num\_variables].values

return X\_colname\_vif

# X\_colname\_vif = feature\_engineer

feature\_engineering\_XbyVIF(X\_train, 8)

**# ### PCA**

pca = PCA(n\_components=2)

#fit( )과 transform( ) 을 호출하여 PCA 변환 데이터 반환

pca.fit(X\_features)

X\_features\_pca = pca.transform(X\_features)

print('PCA Component별 변동성:', pca.explained\_variance\_ratio\_)

**# ### LDA**

lda = LinearDiscriminantAnalysis(n\_components=2)

# fit()호출 시 target값 입력

lda.fit(X\_features, y\_target)

X\_features\_lda = lda.transform(X\_features)

print(iris\_lda.shape)

lda\_columns=['lda\_component\_1','lda\_component\_2']

X\_lda\_df = pd.DataFrame(X\_features\_lda,columns=lda\_columns)

X\_lda\_df['target']=y\_target

**# ### SVD(미완)**

**# ### Truncated SVD(미완)**

**# ### NMF(미완)**

**# ## 정상성 테스트**

# ### 정상성 확인(ARIMA에서 차분 실행 / SARIMA에서 트랜드, 계절성 차분 실행

# \* 차분/로그변환/계절성제거 등을 통해 데이터를 정상성 형태로 변환

## 정상성 테스트 및 모수추론(p=1, q=1, d=0, P=1, Q=1, D(m)=12)

result = pd.Series(sm.tsa.stattools.adfuller(y\_train.values.flatten())[0:4],

index=['Test Statistics', 'p-value', 'Used Lag', 'Used Observations'])

display(result)

plt.figure(figsize=(14,4))

sm.tsa.graphics.plot\_acf(y\_train, lags=100, alpha=0.05, use\_vlines=True, ax=plt.subplot(121))

sm.tsa.graphics.plot\_pacf(y\_train, lags=100, alpha=0.05, use\_vlines=True, ax=plt.subplot(122))

plt.tight\_layout()

plt.show()

**# ### 정상성 확보**

y\_train

## 정상성 확보

plt.figure(figsize=(12,8))

y\_train.plot(ax=plt.subplot(221), title='Y', legend=False)

np.log(y\_train).plot(ax=plt.subplot(222), title='log(Y)', legend=False)

y\_train.diff(1).plot(ax=plt.subplot(223), title='diff1(Y)', legend=False)

np.log(y\_train).diff(1).plot(ax=plt.subplot(224), title='diff1(log(Y))', legend=False)

plt.show()

**# # 분석 및 평가**

**# ## 함수**

### Functionalize

### Evaluation of 1 pair of set

def evaluation(Y\_real, Y\_pred, graph\_on=False):

loss\_length = len(Y\_real.values.flatten()) - len(Y\_pred)

if loss\_length != 0:

Y\_real = Y\_real[loss\_length:]

if graph\_on == True:

pd.concat([Y\_real, pd.DataFrame(Y\_pred, index=Y\_real.index, columns=['prediction'])], axis=1).plot(kind='line', figsize=(20,6),

xlim=(Y\_real.index.min(),Y\_real.index.max()),

linewidth=3, fontsize=20)

plt.title('Time Series of Target', fontsize=20)

plt.xlabel('Index', fontsize=15)

plt.ylabel('Target Value', fontsize=15)

MAE = abs(Y\_real.values.flatten() - Y\_pred).mean()

MSE = ((Y\_real.values.flatten() - Y\_pred)\*\*2).mean()

MAPE = (abs(Y\_real.values.flatten() - Y\_pred)/Y\_real.values.flatten()\*100).mean()

Score = pd.DataFrame([MAE, MSE, MAPE], index=['MAE', 'MSE', 'MAPE'], columns=['Score']).T

Residual = pd.DataFrame(Y\_real.values.flatten() - Y\_pred, index=Y\_real.index, columns=['Error'])

return Score, Residual

# Score\_tr, Residual\_tr = evaluation(Y\_train, pred\_tr\_reg1, graph\_on=True)

### Evaluation of train/test pairs

def evaluation\_trte(Y\_real\_tr, Y\_pred\_tr, Y\_real\_te, Y\_pred\_te, graph\_on=False):

Score\_tr, Residual\_tr = evaluation(Y\_real\_tr, Y\_pred\_tr, graph\_on=graph\_on)

Score\_te, Residual\_te = evaluation(Y\_real\_te, Y\_pred\_te, graph\_on=graph\_on)

Score\_trte = pd.concat([Score\_tr, Score\_te], axis=0)

Score\_trte.index = ['Train', 'Test']

return Score\_trte, Residual\_tr, Residual\_te

# Score\_reg1, Resid\_tr\_reg1, Resid\_te\_reg1 = evaluation\_trte(Y\_train, pred\_tr\_reg1, Y\_test, pred\_te\_reg1, graph\_on=True)

# Error Analysis: 분석모형이 데이터패턴을 모두 추출하여 분석을 종료해도 되는지 판단하기

### Functionalize

### Error analysis

def stationarity\_adf\_test(Y\_Data, Target\_name):

if len(Target\_name) == 0:

Stationarity\_adf = pd.Series(sm.tsa.stattools.adfuller(Y\_Data)[0:4],

index=['Test Statistics', 'p-value', 'Used Lag', 'Used Observations'])

for key, value in sm.tsa.stattools.adfuller(Y\_Data)[4].items():

Stationarity\_adf['Critical Value(%s)'%key] = value

Stationarity\_adf['Maximum Information Criteria'] = sm.tsa.stattools.adfuller(Y\_Data)[5]

Stationarity\_adf = pd.DataFrame(Stationarity\_adf, columns=['Stationarity\_adf'])

else:

Stationarity\_adf = pd.Series(sm.tsa.stattools.adfuller(Y\_Data[Target\_name])[0:4],

index=['Test Statistics', 'p-value', 'Used Lag', 'Used Observations'])

for key, value in sm.tsa.stattools.adfuller(Y\_Data[Target\_name])[4].items():

Stationarity\_adf['Critical Value(%s)'%key] = value

Stationarity\_adf['Maximum Information Criteria'] = sm.tsa.stattools.adfuller(Y\_Data[Target\_name])[5]

Stationarity\_adf = pd.DataFrame(Stationarity\_adf, columns=['Stationarity\_adf'])

return Stationarity\_adf

def stationarity\_kpss\_test(Y\_Data, Target\_name):

if len(Target\_name) == 0:

Stationarity\_kpss = pd.Series(sm.tsa.stattools.kpss(Y\_Data)[0:3],

index=['Test Statistics', 'p-value', 'Used Lag'])

for key, value in sm.tsa.stattools.kpss(Y\_Data)[3].items():

Stationarity\_kpss['Critical Value(%s)'%key] = value

Stationarity\_kpss = pd.DataFrame(Stationarity\_kpss, columns=['Stationarity\_kpss'])

else:

Stationarity\_kpss = pd.Series(sm.tsa.stattools.kpss(Y\_Data[Target\_name])[0:3],

index=['Test Statistics', 'p-value', 'Used Lag'])

for key, value in sm.tsa.stattools.kpss(Y\_Data[Target\_name])[3].items():

Stationarity\_kpss['Critical Value(%s)'%key] = value

Stationarity\_kpss = pd.DataFrame(Stationarity\_kpss, columns=['Stationarity\_kpss'])

return Stationarity\_kpss

**# ## Y예측을 위한 Base분석 실행**

**# 상단**

# \* Model :

# \* Method :

# \* **\*\*\*R-squared : 높을 수록 좋음.\*\*\*** 사회과학에서는 보통 0.4 이상 이면 괜찮은 모델이라고 봄

# - 데이터에 잘 적합된 정도, 모델에 의해 설명되는 데이터 분산의 정도(퍼센트)

# \* **\*\*\*Adj. R-squared\*\*\*** : 독립변수의 개수와 표본의 크기를 고려하여 R-squared를 보정

# - 독립변수가 여러 개인 다중회귀분석에서 사용

# - 서로 다른 모형을 비교할 때는 이 지표가 높은 쪽은 선택한다

# **\* \*\*\*F-statistic\*\*\*** : F-test의 검정통계량으로 변수들 전체의 유의성을 체크함

**# \* \*\*\*Prob (F-statistic) : F-검정의 p-value로 p-value < 0.05 이여야 함\*\*\***

**# \* \*\*\*Log-likelihood : 우도는 높을수록 좋음. 우도의 값은 확률임(0 < 확률 < 1)\*\*\***

# - 회귀모델이 실제값이랑 잘 맞으면 우도가 높아짐

# - 종속변수가 정규분포라 가정했을 때 그 우도

# - 로그우도도 R제곱과 마찬가지로 독립변수가 많아지면 증가한다

# - 모델과 추정치가 데이터와 잘 맞는 정도를 확률로 표현한 것

# - 확률은 모델과 추정치가 원인이고 데이터가 결과이다.(모델과 추정치에 의해 이러한 데이터가 나올 확률/데이터에 대한 확률)

# - 우도는 데이터가 원인이고 모델과 추정치는 결과이다.(데이터에 의해 이러한 모델과 추정치가 나올 우도/ 모델과 추정치에 대한 우도)

# - 최대우도법 : 데이터에 제일 잘 맞는 모델과 추정치를 계산하는 방법

# - 최우추정치 : 데이터에 제일 잘 맞는 추정치

**# \* \*\*\*AIC : 낮을수록 좋음. -2log(우도) + 2P\*\*\***

**# \* \*\*\*BIC : 낮을수록 좋음. -2log(우도) + plog(n)\*\*\***

# - -2log(우도) = Deviance

# - 로그우도를 독립변수의 수로 보정한 값

# - 높을수록 안좋음. 어떤 모형이 가장 적절하냐를 판단할때 사용

# - 표준오차랑 비슷한 것

# - 계산하기 위해서는 우도를 알아야 함 / -2log(우도) + (something more)

# - -2를 곱한건 통계적인 이유가 있음

# - log를 씌운것은 값이 너무 작아서 그걸 우리가 보기 편하게 하기 위해서임

**# \* \*\*\*No. Observations : 해당 개수의 데이터 쌍을 가지고 회귀분석을 실시하였다는 것을 알 수 있다.\*\*\***

# **\* \*\*\*Df Model : 회귀분석 전체 파라미터 숫자 - 1.\*\*\*** 회귀분석의 "예측변수의 숫자(k)"를 의미

# - 회귀분석의 전체 파라미터는 1개의 종속변수를 포함하므로 Df Model은 다른 식으로는 "회귀분석 전체 파라미터 숫자 - 1" 이다

# **\* \*\*\*Df Residuals : Df Residuals는 "No.observations - (Df Model + 1)"로 산출\*\*\***

# - 전체 관찰데이터의 수에서 회귀모형의 전체 파라미터의 수를 뺀 값

#

#

# 하단

# \* Omnibus :

# \* Prob(Omnibus) :

# **\* \*\*\*Durbin-Watson : 중간이 좋음. 자기상관 테스트(잔차의 독립성)로 p-value >= 0.05 이여야 함\*\*\***

# **- \*\*\*보통 1.5 ~ 2.5사이이면 독립으로 판단하고 회귀모형이 적합하다는 것을 의미\*\*\***

# - 0이면 잔차들이 양의 자기상관을 갖고, 2이면 자기상관이 없는 독립성을 갖고, 4이면 잔차들이 음의 자기상관을 갖는다고 해석

# - **\*\*\*DW검정값이 0 또는 4에 가깝다는 것은 잔차들이 자기상관을 가지고 있다는 의미\*\*\***이고, 이는 t값, F값, R제곱을 실제보다 증가시켜 실제로 유의미하지 않은 결과를 유의미한 결과로 왜곡하게 된다.

# \* **\*\*\*Jaque-Bera (JB) : 0에서 멀리 떨어진 값이 나오면, 그때는 정규분포(normal distribution)에 적합하지 않다고 판정\*\*\***

# - 정규분포 테스트의 검정통계량

# - JB = (n/6) \* (S²+ (C²/4) : n:관찰된 표본 수, S:표본의 왜도(skewness), C:표본의 첨도(kurtosis)

#

# **\* \*\*\*Prob(JB) : Jaque\_Bera의 유의수준으로 p-value >= 0.05 이여야 함\*\*\***

# \* **\*\*\*Skew : 왜도(중심에서 기울어진 정도) / 절대값 3 미만 기준에 부합 / -2~2 치우침은 왜도가 크지 않다고 판단\*\*\***

# **\* \*\*\*Kutosis : 첨도(뾰족한 정도) / 절대값 7 미만 기준에 부합 / 첨도가 크면 이상치가 많음, \*\*\***

# **\* \*\*\*Cond. No. : 공분산행렬 (XT)X 의 조건수가 크면 회귀분석을 사용한 예측값도 오차가 커진다.\*\*\***

# - 조건수(conditional number)는 가장 큰 고유치와 가장 작은 고유치의 비율을 뜻한다. 회귀분석에서는 공분산행렬 (XT)X 의 가장 큰 고유치와 가장 작은 고유치의 비율

# - 조건수는 가장 작은 경우의 예는 행렬 A 가 단위 행렬인 경우다. 이 때 조건수의 값은 1이다.(cond(I)=1)

# - 연립방정식을 이루는 행렬의 조건수가 커지면 상수항 오차가 작은 경우에도 해의 오차가 커지게 된다

# - 회귀분석에서 조건수가 커지는 경우는 크게 두 가지가 있다.

# - 변수들의 단위 차이로 인해 숫자의 스케일이 크게 달라지는 경우. 이 경우에는 스케일링(scaling)으로 해결한다.

# - 다중 공선성 즉, 상관관계가 큰 독립 변수들이 있는 경우, 이 경우에는 변수 선택이나 PCA를 사용한 차원 축소 등으로 해결한다.

# LinearRegression (using statsmodels)

fit\_reg1 = sm.OLS(y\_train, X\_train).fit()

display(fit\_reg1.summary())

pred\_tr\_reg1 = fit\_reg1.predict(X\_train).values

pred\_te\_reg1 = fit\_reg1.predict(X\_test).values

# comparison of precision

display(Score\_reg1\_rd)

display(Score\_reg1\_feR)

display(Score\_reg1\_feRS)

display(Score\_reg1\_feRSM)

**# ## Y예측을 위한 ML분석 실행**

# ### Regularization

**# #### Ridge**

### Regularization

# Ridge

ridge\_reg = Ridge(alpha=0.5, fit\_intercept=True, normalize=False, random\_state=123)

ridge\_reg.fit(X\_train, y\_train)

X\_ridge\_pred = ridge\_reg.predict(X\_train).flatten()

y\_ridge\_pred = ridge\_reg.predict(X\_test).flatten()

# Evaluation

Score\_reg2\_feRSM, Resid\_tr\_reg2\_feRSM, Resid\_te\_reg2\_feRSM = evaluation\_trte(y\_train, X\_ridge\_pred,

y\_test, y\_ridge\_pred, graph\_on=True)

display(Score\_reg2\_feRSM)

**# #### Lasso**

# Lasso

lasso\_reg = Lasso(alpha=0.5, fit\_intercept=True, normalize=False, random\_state=123)

lasso\_reg.fit(X\_train, y\_train)

X\_lasso\_pred = lasso\_reg.predict(X\_train).flatten()

y\_lasso\_pred = lasso\_reg.predict(X\_test).flatten()

# Evaluation

lasso\_trte\_score, lasso\_tr\_resid, lasso\_te\_resid = evaluation\_trte(y\_train, X\_lasso\_pred,

y\_test, y\_lasso\_pred, graph\_on=False)

display(lasso\_trte\_score)

**# #### ElasticNet**

# ElasticNet

elastic\_reg = ElasticNet(alpha=0.01, l1\_ratio=1, fit\_intercept=True, normalize=False, random\_state=123)

elastic\_reg.fit(X\_train, y\_train)

X\_elastic\_pred = elastic\_reg.predict(X\_train).flatten()

y\_elastic\_pred = elastic\_reg.predict(X\_test).flatten()

# Evaluation

Score\_reg2\_feRSM, Resid\_tr\_reg2\_feRSM, Resid\_te\_reg2\_feRSM = evaluation\_trte(y\_train, X\_elastic\_pred,

y\_test, y\_elastic\_pred, graph\_on=False)

display(Score\_reg2\_feRSM)

**# ### Bagging**

**# #### DecisionTree**

# DecisionTree

dt\_reg = DecisionTreeRegressor()

dt\_reg.fit(X\_train, y\_train)

X\_dt\_pred = dt\_reg.predict(X\_train)

y\_dt\_pred = dt\_reg.predict(X\_test)

# Evaluation

Score\_reg2\_feRSM, Resid\_tr\_reg2\_feRSM, Resid\_te\_reg2\_feRSM = evaluation\_trte(y\_train, X\_dt\_pred,

y\_test, y\_dt\_pred, graph\_on=True)

display(Score\_reg2\_feRSM)

**# #### RandomForestRegressor**

# RandomForestRegressor

rf\_reg = RandomForestRegressor(n\_estimators=100, random\_state=123)

rf\_reg.fit(X\_train, y\_train)

X\_rf\_pred = rf\_reg.predict(X\_train)

y\_rf\_pred = rf\_reg.predict(X\_test)

# Evaluation

Score\_reg2\_feRSM, Resid\_tr\_reg2\_feRSM, Resid\_te\_reg2\_feRSM = evaluation\_trte(y\_train, X\_rf\_pred,

y\_test, y\_rf\_pred, graph\_on=False)

display(Score\_reg2\_feRSM)

**# ### Boosting**

# #### GradientBoostingRegression

# GradientBoostingRegression

gb\_reg = GradientBoostingRegressor(alpha=0.1, learning\_rate=0.05, loss='huber', criterion='friedman\_mse',

n\_estimators=1000, random\_state=123)

gb\_reg.fit(X\_train, y\_train)

X\_gb\_pred = gb\_reg.predict(X\_train)

y\_gb\_pred = gb\_reg.predict(X\_test)

# Evaluation

Score\_reg2\_feRSM, Resid\_tr\_reg2\_feRSM, Resid\_te\_reg2\_feRSM = evaluation\_trte(y\_train, X\_gb\_pred,

y\_test, y\_gb\_pred, graph\_on=False)

display(Score\_reg2\_feRSM)

**# #### XGBRegressor**

# XGBoost

xgb\_reg = XGBRegressor(learning\_rate=0.05, n\_estimators=100, random\_state=123)

xgb\_reg.fit(X\_train, y\_train)

X\_xgb\_pred = xgb\_reg.predict(X\_train)

y\_xgb\_pred = xgb\_reg.predict(X\_test)

# Evaluation

Score\_reg2\_feRSM, Resid\_tr\_reg2\_feRSM, Resid\_te\_reg2\_feRSM = evaluation\_trte(y\_train, X\_xgb\_pred,

y\_test, y\_xgb\_pred, graph\_on=False)

display(Score\_reg2\_feRSM)

**# #### LGBMRegressor**

# LightGMB

lgbm\_reg = LGBMRegressor(learning\_rate=0.05, n\_estimators=100, random\_state=123)

lgbm\_reg.fit(X\_train, y\_train)

X\_lgbm\_pred = lgbm\_reg.predict(X\_train)

y\_lgbm\_pred = lgbm\_reg.predict(X\_test)

# Evaluation

Score\_reg2\_feRSM, Resid\_tr\_reg2\_feRSM, Resid\_te\_reg2\_feRSM = evaluation\_trte(y\_train, X\_lgbm\_pred,

y\_test, y\_lgbm\_pred, graph\_on=False)

display(Score\_reg2\_feRSM)

**# ### 중요도 보기**

# - 중요도가 제공되는 5가지 모델

# - gb\_clf\_imp = pd.DataFrame({'Feature':X.columns, 'gb\_clf importance':gb\_clf.feature\_importances\_})

# - xgb\_clf\_imp = pd.DataFrame({'Feature':X.columns, 'xgb\_clf importance':xgb\_clf.feature\_importances\_})

# - rf\_clf\_imp = pd.DataFrame({'Feature':X.columns, 'rf\_clf importance':rf\_clf.feature\_importances\_})

# - ext\_clf\_imp = pd.DataFrame({'Feature':X.columns, 'ext\_clf importance':ext\_clf.feature\_importances\_})

# - ada\_clf\_imp = pd.DataFrame({'Feature':X.columns, 'ada\_clf importance':ada\_clf.feature\_importances\_})

**# #### randomforest 중요도**

# Prediction Effect of Variables

Variable\_Importances = pd.DataFrame([rf\_reg.feature\_importances\_],

columns=X\_train.columns,

index=['importance']).T.sort\_values(by=['importance'], ascending=False)

display(Variable\_Importances)

Variable\_Importances.plot.bar(figsize=(12,6), fontsize=15)

plt.title('Variable Importances', fontsize=15)

plt.show()

**# #### XGBRegressor 중요도**

# Prediction Effect of Variables

Variable\_Importances = pd.DataFrame([xgb\_reg.feature\_importances\_],

columns=X\_train.columns,

index=['importance']).T.sort\_values(by=['importance'], ascending=False)

display(Variable\_Importances)

Variable\_Importances.plot.bar(figsize=(12,6), fontsize=15)

plt.title('Variable Importances', fontsize=15)

plt.show()

**# ## Y예측을 위한 TS분석 실행**

**# ### ARMAX**

# LinearRegression (using statsmodels)

fit\_reg1 = sm.OLS(y\_train, X\_train).fit()

display(fit\_reg1.summary())

pred\_tr\_reg1 = fit\_reg1.predict(X\_train).values

pred\_te\_reg1 = fit\_reg1.predict(X\_test).values

import requests

from io import BytesIO

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

# 데이터 로딩 및 확인

source\_url = requests.get('http://www.stata-press.com/data/r12/friedman2.dta').content

raw = pd.read\_stata(BytesIO(source\_url))

raw.index = raw.time

raw\_using = raw.loc['1960':'1981',["consump", "m2"]]

raw\_using.plot()

plt.show()

**# 모델링**

**## 회귀분석**

fir\_reg1 = sm.OLS(y\_train, X\_train).fit()

display(fir\_reg1.summary())

## 잔차 확인

fir\_reg1.resid.plot()

plt.show()

## 잔차 ACF/PACF

plt.figure(figsize=(10, 8))

sm.graphics.tsa.plot\_acf(fir\_reg1.resid, lags=50, ax=plt.subplot(211))

plt.xlim(-1, 51)

plt.ylim(-1.1, 1.1)

plt.title("Residual ACF")

sm.graphics.tsa.plot\_pacf(fir\_reg1.resid, lags=50, ax=plt.subplot(212))

plt.xlim(-1, 51)

plt.ylim(-1.1, 1.1)

plt.title("Residual PACF")

plt.tight\_layout()

plt.show()

**# 모델링**

**## ARIMAX**

# exog에 넣을 X\_train의 컬럼 뽑기

exog\_tr = ['season', 'holiday', 'workingday', 'weather', 'temp', 'atemp',

'humidity', 'windspeed', 'Year', 'Quater\_ver2', 'Month', 'Day', 'Hour',

'DayofWeek', 'count\_lag1', 'count\_lag2', 'Quater\_Dummy\_2',

'Quater\_Dummy\_3', 'Quater\_Dummy\_4']

exog\_te = []

fit\_reg2 = sm.tsa.ARMA(y\_train, (1,1), exog=X\_train[exog\_tr]).fit()

display(fit\_reg2.summary())

## 잔차 확인

fit\_reg2.resid.plot()

plt.show()

## 잔차 ACF/PACF

plt.figure(figsize=(10, 8))

sm.graphics.tsa.plot\_acf(fit\_reg2.resid, lags=50, ax=plt.subplot(211))

plt.xlim(-1, 51)

plt.ylim(-1.1, 1.1)

plt.title("Residual ACF")

sm.graphics.tsa.plot\_pacf(fit\_reg2.resid, lags=50, ax=plt.subplot(212))

plt.xlim(-1, 51)

plt.ylim(-1.1, 1.1)

plt.title("Residual PACF")

plt.tight\_layout()

plt.show()

**# ### ARIMAX**

# \* ACF, PACF가 잘 적합될 경우 파라미터마다 나오는 그래프를 알고 있어야 함

# ### SARIMAX(그래프가 차수별로 어떻게 나오는지 봐야 함, 프린트 하는 것도 좋을 것 같음)

# - AIC, BIC가 낮은걸로 적합도를 확인함

# #### 첫번째 모델링

# 데이터 준비

data = sm.datasets.get\_rdataset("AirPassengers")

raw = data.data.copy()

# 데이터 전처리

## 시간 인덱싱

if 'time' in raw.columns:

raw.index = pd.date\_range(start='1/1/1949', periods=len(raw['time']), freq='M')

del raw['time']

## 정상성 확보

plt.figure(figsize=(12,8))

raw.plot(ax=plt.subplot(221), title='Y', legend=False)

np.log(raw).plot(ax=plt.subplot(222), title='log(Y)', legend=False)

raw.diff(1).plot(ax=plt.subplot(223), title='diff1(Y)', legend=False)

np.log(raw).diff(1).plot(ax=plt.subplot(224), title='diff1(log(Y))', legend=False)

plt.show()

# 데이터로딩 및 확인

data = sm.datasets.get\_rdataset("deaths", "MASS")

raw = data.data

raw.value = np.log(raw.value)

raw.plot(x='time', y='value')

plt.show()

# ACF/PACF 확인

plt.figure(figsize=(10, 8))

sm.graphics.tsa.plot\_acf(raw.value.values, lags=50, ax=plt.subplot(211))

plt.xlim(-1, 51)

plt.ylim(-1.1, 1.1)

plt.title("ACF")

sm.graphics.tsa.plot\_pacf(raw.value.values, lags=50, ax=plt.subplot(212))

plt.xlim(-1, 51)

plt.ylim(-1.1, 1.1)

plt.title("PACF")

plt.tight\_layout()

plt.show()

# ARMA(1,1) 모델링

fit = sm.tsa.SARIMAX(raw.value, trend='c', order=(1,0,1), seasonal\_order=(0,0,0,0)).fit()

display(fit.summary())

# 잔차진단

fit.plot\_diagnostics(figsize=(10,8))

plt.tight\_layout()

plt.show()

# ARIMA 모델링 (raw)

fit = sm.tsa.SARIMAX(raw.value, trend='c', order=(1,1,1), seasonal\_order=(0,0,0,0)).fit()

display(fit.summary())

# 잔차진단

fit.plot\_diagnostics(figsize=(10,8))

plt.tight\_layout()

plt.show()

# ARIMA 모델링 (log(raw))

fit = sm.tsa.SARIMAX(np.log(raw.value), trend='c', order=(1,1,1), seasonal\_order=(0,0,0,0)).fit()

display(fit.summary())

# 잔차진단

fit.plot\_diagnostics(figsize=(10,8))

plt.tight\_layout()

plt.show()

**# #### (이걸로 돌리기)두번째 모델링(전체 실행하는 것이라고 보면 됨)**

X\_train.columns

## 최종 타겟 선정 및 Train/Test 데이터 분리

# candidate = candidate\_trend.copy()

# split\_date = '1958-01-01'

# Y\_train = candidate[candidate.index < split\_date]

# Y\_test = candidate[candidate.index >= split\_date]

## 시각화 및 모수추론(p=1, q=0, d=1, P=1, Q=1, D(m)=12)

plt.figure(figsize=(14,4))

sm.tsa.graphics.plot\_acf(y\_train, lags=50, alpha=0.05, use\_vlines=True, ax=plt.subplot(121))

sm.tsa.graphics.plot\_pacf(y\_train, lags=50, alpha=0.05, use\_vlines=True, ax=plt.subplot(122))

plt.show()

# 모델링

**## SARIMAX**

**### 파라미터 조정 ###**

# 스케일링 돌리기

logarithm, differencing = False, False

# 이그자지너스

exog\_tr = X\_train[['DayofWeek', 'count\_lag1', 'count\_lag2', 'Quater\_Dummy\_2']]

exog\_te = X\_test[['DayofWeek', 'count\_lag1', 'count\_lag2', 'Quater\_Dummy\_2']]

# 차분 넣기(트랜드)

p = 1

d = 1

q = 0

# 차분 넣기(트랜드)

P = 1

D = 0

Q = 1

seasonal\_diff\_order = 12

#############################

# SARIMAX(y\_train, / exog=exog\_te, exog=exog\_te 이 3개를 스케일링 할때마다 같이 바꿔줘서 예측해야 함

sarimax\_reg = sm.tsa.SARIMAX(y\_train, order=(p,d,q),

seasonal\_order=(P,D,Q,seasonal\_diff\_order), trend='c', exog=exog\_tr).fit()

display(sarimax\_reg.summary())

sarimax\_reg = sarimax\_reg.predict()

sarimax\_pred = sarimax\_reg.get\_forecast(len(y\_test), exog=exog\_te).predicted\_mean

sarimax\_pred\_ci = sarimax\_reg.get\_forecast(len(y\_test), exog=exog\_te).conf\_int()

## 비정상성으로 변환

if logarithm:

Y\_train = np.exp(y\_train).copy()

Y\_test = np.exp(y\_test).copy()

pred\_tr\_ts\_sarimax = np.exp(pred\_tr\_ts\_sarimax).copy()

pred\_te\_ts\_sarimax = np.exp(pred\_te\_ts\_sarimax).copy()

pred\_te\_ts\_sarimax\_ci = np.exp(pred\_te\_ts\_sarimax\_ci).copy()

if differencing:

pred\_tr\_ts\_sarimax = np.cumsum(pred\_tr\_ts\_sarimax).copy()

# 검증

get\_ipython().run\_line\_magic('reload\_ext', 'autoreload')

get\_ipython().run\_line\_magic('autoreload', '2')

Score\_ts\_sarimax, Resid\_tr\_ts\_sarimax, Resid\_te\_ts\_sarimax = evaluation\_trte(y\_train, pred\_tr\_ts\_sarimax,

y\_test, pred\_te\_ts\_sarimax, graph\_on=True)

display(Score\_ts\_sarimax)

ax = pd.DataFrame(y\_test).plot(figsize=(12,4))

pd.DataFrame(pred\_te\_ts\_sarimax, index=y\_test.index, columns=['prediction']).plot(kind='line',

xlim=(y\_test.index.min(),y\_test.index.max()),

linewidth=3, fontsize=20, ax=ax)

ax.fill\_between(pd.DataFrame(pred\_te\_ts\_sarimax\_ci, index=y\_test.index).index,

pd.DataFrame(pred\_te\_ts\_sarimax\_ci, index=y\_test.index).iloc[:,0],

pd.DataFrame(pred\_te\_ts\_sarimax\_ci, index=y\_test.index).iloc[:,1], color='k', alpha=0.15)

plt.show()

# 잔차진단

error\_analysis(Resid\_tr\_ts\_sarimax, ['Error'], y\_train, graph\_on=True)

**# #### auto\_SARIMAX(스케일링부터 전체)**

# Data Loading

# location = 'https://raw.githubusercontent.com/cheonbi/DataScience/master/Data/Bike\_Sharing\_Demand\_Full.csv'

location = './Data/BikeSharingDemand/Bike\_Sharing\_Demand\_Full.csv'

raw\_all = pd.read\_csv(location)

# Feature Engineering

raw\_fe = feature\_engineering(raw\_all)

### Reality ###

target = ['count\_trend', 'count\_seasonal', 'count\_Day', 'count\_Week', 'count\_diff']

raw\_feR = feature\_engineering\_year\_duplicated(raw\_fe, target)

###############

# Data Split

# Confirm of input and output

Y\_colname = ['count']

X\_remove = ['datetime', 'DateTime', 'temp\_group', 'casual', 'registered']

X\_colname = [x for x in raw\_fe.columns if x not in Y\_colname+X\_remove]

X\_train\_feR, X\_test\_feR, Y\_train\_feR, Y\_test\_feR = datasplit\_ts(raw\_feR, Y\_colname, X\_colname, '2012-07-01')

### Reality ###

target = ['count\_lag1', 'count\_lag2']

X\_test\_feR = feature\_engineering\_lag\_modified(Y\_test\_feR, X\_test\_feR, target)

###############

### Scaling ###

X\_train\_feRS, X\_test\_feRS = feature\_engineering\_scaling(preprocessing.Normalizer(), X\_train\_feR, X\_test\_feR)

###############

### Multicollinearity ###

print('Number\_of\_Total\_X: ', len(X\_train\_feRS.columns))

X\_colname\_vif = feature\_engineering\_XbyVIF(X\_train\_feRS, 12)

print('Number\_of\_Selected\_X: ', len(X\_colname\_vif))

X\_train\_feRSM, X\_test\_feRSM = X\_train[X\_colname\_vif].copy(), X\_test\_feRS[X\_colname\_vif].copy()

### 모델링 ###

## Additional Features

exog\_tr = X\_train\_feRSM[['count\_seasonal', 'weather', 'count\_lag2', 'count\_diff', 'Quater\_ver2', 'Hour', 'workingday', 'DayofWeek']]

exog\_te = X\_test\_feRSM[['count\_seasonal', 'weather', 'count\_lag2', 'count\_diff', 'Quater\_ver2', 'Hour', 'workingday', 'DayofWeek']]

## Parameter Setting

p, q = range(1,3), range(1,3)

d = range(0,1)

P, Q = range(1,3), range(1,3)

D = range(1,2)

m = 12

trend\_pdq = list(product(p, d, q))

seasonal\_pdq = [(candi[0], candi[1], candi[2], m) for candi in list(product(P, D, Q))]

## SARIMAX

logarithm, differencing = True, False

AIC = []

SARIMAX\_order = []

for trend\_param in tqdm(trend\_pdq):

for seasonal\_params in seasonal\_pdq:

try:

result =sm.tsa.SARIMAX(Y\_train\_feR, trend='c',

order=trend\_param, seasonal\_order=seasonal\_params, exog=exog\_tr).fit()

print('Fit SARIMAX: trend\_order={} seasonal\_order={} AIC={}, BIC={}'.format(trend\_param, seasonal\_params, result.aic, result.bic, end='\r'))

AIC.append(result.aic)

SARIMAX\_order.append([trend\_param, seasonal\_params])

except:

continue

## Parameter Selection

print('The smallest AIC is {} for model SARIMAX{}x{}'.format(min(AIC), SARIMAX\_order[AIC.index(min(AIC))][0],

SARIMAX\_order[AIC.index(min(AIC))][1]))

## Auto-SARIMAX Fitting

fit\_ts\_sarimax = sm.tsa.SARIMAX(Y\_train\_feR, trend='c', order=SARIMAX\_order[AIC.index(min(AIC))][0],

seasonal\_order=SARIMAX\_order[AIC.index(min(AIC))][1], exog=exog\_tr).fit()

display(fit\_ts\_sarimax.summary())

pred\_tr\_ts\_sarimax = fit\_ts\_sarimax.predict()

pred\_te\_ts\_sarimax = fit\_ts\_sarimax.get\_forecast(len(Y\_test\_feR), exog=exog\_te).predicted\_mean

pred\_te\_ts\_sarimax\_ci = fit\_ts\_sarimax.get\_forecast(len(Y\_test\_feR), exog=exog\_te).conf\_int()

## 비정상성으로 변환

if logarithm:

Y\_train = np.exp(Y\_train).copy()

Y\_test = np.exp(Y\_test).copy()

pred\_tr\_ts\_sarimax = np.exp(pred\_tr\_ts\_sarimax).copy()

pred\_te\_ts\_sarimax = np.exp(pred\_te\_ts\_sarimax).copy()

pred\_te\_ts\_sarimax\_ci = np.exp(pred\_te\_ts\_sarimax\_ci).copy()

if differencing:

pred\_tr\_ts\_sarimax = np.cumsum(pred\_tr\_ts\_sarimax).copy()

# 검증

Score\_ts\_sarimax, Resid\_tr\_ts\_sarimax, Resid\_te\_ts\_sarimax = evaluation\_trte(Y\_train\_feR, pred\_tr\_ts\_sarimax,

Y\_test\_feR, pred\_te\_ts\_sarimax, graph\_on=True)

display(Score\_ts\_sarimax)

ax = pd.DataFrame(Y\_test\_feR).plot(figsize=(12,4))

pd.DataFrame(pred\_te\_ts\_sarimax, index=Y\_test\_feR.index, columns=['prediction']).plot(kind='line',

xlim=(Y\_test\_feR.index.min(),Y\_test\_feR.index.max()),

linewidth=3, fontsize=20, ax=ax)

ax.fill\_between(pd.DataFrame(pred\_te\_ts\_sarimax\_ci, index=Y\_test\_feR.index).index,

pd.DataFrame(pred\_te\_ts\_sarimax\_ci, index=Y\_test\_feR.index).iloc[:,0],

pd.DataFrame(pred\_te\_ts\_sarimax\_ci, index=Y\_test\_feR.index).iloc[:,1], color='k', alpha=0.15)

plt.show()

# 잔차진단

error\_analysis(Resid\_tr\_ts\_sarimax, ['Error'], Y\_train\_feR, graph\_on=True)

### Multicollinearity ###

print('Number\_of\_Total\_X: ', len(X\_train.columns))

X\_colname\_vif = feature\_engineering\_XbyVIF(X\_train, 12)

print('Number\_of\_Selected\_X: ', len(X\_colname\_vif))

X\_train\_feRSM, X\_test\_feRSM = X\_train[X\_colname\_vif].copy(), X\_test[X\_colname\_vif].copy()

### 모델링 ###

## Additional Features

exog\_tr = X\_train[['weather', 'count\_lag2', 'Quater\_ver2', 'Hour', 'workingday', 'DayofWeek']]

exog\_te = X\_test[['weather', 'count\_lag2', 'Quater\_ver2', 'Hour', 'workingday', 'DayofWeek']]

## Parameter Setting

p, q = range(1,3), range(1,3)

d = range(0,1)

P, Q = range(1,3), range(1,3)

D = range(1,2)

m = 12

trend\_pdq = list(product(p, d, q))

seasonal\_pdq = [(candi[0], candi[1], candi[2], m) for candi in list(product(P, D, Q))]

## SARIMAX

logarithm, differencing = True, False

AIC = []

SARIMAX\_order = []

for trend\_param in tqdm(trend\_pdq):

for seasonal\_params in seasonal\_pdq:

try:

result =sm.tsa.SARIMAX(y\_train, trend='c',

order=trend\_param, seasonal\_order=seasonal\_params, exog=exog\_tr).fit()

print('Fit SARIMAX: trend\_order={} seasonal\_order={} AIC={}, BIC={}'.format(trend\_param, seasonal\_params, result.aic, result.bic, end='\r'))

AIC.append(result.aic)

SARIMAX\_order.append([trend\_param, seasonal\_params])

except:

continue

## Parameter Selection

print('The smallest AIC is {} for model SARIMAX{}x{}'.format(min(AIC), SARIMAX\_order[AIC.index(min(AIC))][0],

SARIMAX\_order[AIC.index(min(AIC))][1]))

## Auto-SARIMAX Fitting

fit\_ts\_sarimax = sm.tsa.SARIMAX(y\_train, trend='c', order=SARIMAX\_order[AIC.index(min(AIC))][0],

seasonal\_order=SARIMAX\_order[AIC.index(min(AIC))][1], exog=exog\_tr).fit()

display(fit\_ts\_sarimax.summary())

pred\_tr\_ts\_sarimax = fit\_ts\_sarimax.predict()

pred\_te\_ts\_sarimax = fit\_ts\_sarimax.get\_forecast(len(Y\_test\_feR), exog=exog\_te).predicted\_mean

pred\_te\_ts\_sarimax\_ci = fit\_ts\_sarimax.get\_forecast(len(Y\_test\_feR), exog=exog\_te).conf\_int()

## 비정상성으로 변환

if logarithm:

Y\_train = np.exp(y\_train).copy()

Y\_test = np.exp(y\_test).copy()

pred\_tr\_ts\_sarimax = np.exp(pred\_tr\_ts\_sarimax).copy()

pred\_te\_ts\_sarimax = np.exp(pred\_te\_ts\_sarimax).copy()

pred\_te\_ts\_sarimax\_ci = np.exp(pred\_te\_ts\_sarimax\_ci).copy()

if differencing:

pred\_tr\_ts\_sarimax = np.cumsum(pred\_tr\_ts\_sarimax).copy()

# 검증

Score\_ts\_sarimax, Resid\_tr\_ts\_sarimax, Resid\_te\_ts\_sarimax = evaluation\_trte(y\_train, pred\_tr\_ts\_sarimax,

y\_train, pred\_te\_ts\_sarimax, graph\_on=True)

display(Score\_ts\_sarimax)

ax = pd.DataFrame(y\_test).plot(figsize=(12,4))

pd.DataFrame(pred\_te\_ts\_sarimax, index=y\_test.index, columns=['prediction']).plot(kind='line',

xlim=(y\_train.index.min(),y\_test.index.max()),

linewidth=3, fontsize=20, ax=ax)

ax.fill\_between(pd.DataFrame(pred\_te\_ts\_sarimax\_ci, index=y\_test.index).index,

pd.DataFrame(pred\_te\_ts\_sarimax\_ci, index=y\_test.index).iloc[:,0],

pd.DataFrame(pred\_te\_ts\_sarimax\_ci, index=y\_test.index).iloc[:,1], color='k', alpha=0.15)

plt.show()

# 잔차진단

error\_analysis(Resid\_tr\_ts\_sarimax, ['Error'], y\_train, graph\_on=True)

**# ## 잔차진단(미완)**

**# \* 통계량**

**# - \*\*\*Stationarity\_adf\*\*\* :**

# - 의사결정 : **\*\*\*p-value < 0.05\*\*\***

# - 가설확인 : H0 = 비정상 상태이다 / H1 = 정상 상태이다.

# - ADF 정상성 확인(추세 제거 확인용) / ADF 검정통계량은 정상이라고 해도 데이터에 계절성이 포함되면 ACF의 비정상 Lag 존재하는 비정상데이터 가능

# \_ 종류 : ADF-GLS test, Phillips-Perron(PP) test(ADF와 동일) / Kwiatkowski Phillips Schmidt Shin(KPSS) test(ADF와 반대)

**# - \*\*\*Stationarity\_kpss\*\*\* :**

**#** - 의사결정 : **\*\*\*p-value >= 0.05\*\*\***

# - 가설확인 : H0 = 정상 상태이다 / H1 = 비정상 상태이다.

# - KPSS 정상성 확인(계절성 제거 확인용) / KPSS 검정통계량은 정상이라고 해도 데이터에 추세가 포함되면 ACF의 비정상 Lag 존재하는 비정상데이터 가능

# - **\*\*\*Normality\*\*\* :**

# - 의사결정 : **\*\*\*p-value >= 0.05\*\*\***

# - 가설확인 : H0 = 정규분포 형태이다 / H1 = 정규분포가 아닌 형태이다.

# - 종류 : Shapiro\_Wilk test, Kolmogorov-Smirnov test, Lilliefors teset, Anderson\_Darling test, Jarque-Bera test, Pearson's chi-squared test, D'Agostino's K-squared test (가설확인 모두 동일)

# - **\*\*Autocorr(lag1, lag5, lag10, lag50)\*\*** : 자기상관, 잔차의 독립성

# - 의사결정 : **\*\*\*p-value >= 0.05\*\*\***

# - 가설확인 : H0 = 시계열 데이터의 Autocorrelation은 0이다.(존재하지 않는다) / H1 = 0이 아니다(존재한다).

# - 종류 : Ljung-Box test, Portmanteau test, Breusch-Godfrey test, Durbin-Watson statistic (Ljung-Box와 동일)

# - **\*\*\*Heteroscedasticity\*\*\* :**

# - 의사결정 : **\*\*\*p-value < 0.05\*\*\***

# - 가설확인 : H0 = 시계열 데이터의 Homoscedasticity 상태다(등분산이다.) / H1 = 등분산이 아니다.

# - 종류 : Goldfeld-Quandt test, Breuch-Pagan test, Bartlett's test (가설확인 모두 동일)

#

**# \* 그래프**

# - ACF : MA(q)에서 q번째까지 ACF 그래프에서 튀는 개수 / ARMA(p,q)에서 q번째까지 지수적 감소 혹은 진동하는 형태

# - PACF : AR(p)에서 p번째까지 PACF 그래프에서 튀는 개수 / ARMA(p,q)에서 p번째까지 지수적 감소 혹은 진동하는 형태

**# ### 함수**

def error\_analysis(Y\_Data, Target\_name, X\_Data, graph\_on=False):

for x in Target\_name:

Target\_name = x

X\_Data = X\_Data.loc[Y\_Data.index]

if graph\_on == True:

##### Error Analysis(Plot)

Y\_Data['RowNum'] = Y\_Data.reset\_index().index

# Stationarity(Trend) Analysis

sns.set(palette="muted", color\_codes=True, font\_scale=2)

sns.lmplot(x='RowNum', y=Target\_name, data=Y\_Data, fit\_reg='True', size=5.2, aspect=2, ci=99, sharey=True)

del Y\_Data['RowNum']

# Normal Distribution Analysis

figure, axes = plt.subplots(figsize=(12,8))

sns.distplot(Y\_Data[Target\_name], norm\_hist='True', fit=stats.norm, ax=axes)

# Lag Analysis

length = int(len(Y\_Data[Target\_name])/10)

figure, axes = plt.subplots(1, 4, figsize=(12,3))

pd.plotting.lag\_plot(Y\_Data[Target\_name], lag=1, ax=axes[0])

pd.plotting.lag\_plot(Y\_Data[Target\_name], lag=5, ax=axes[1])

pd.plotting.lag\_plot(Y\_Data[Target\_name], lag=10, ax=axes[2])

pd.plotting.lag\_plot(Y\_Data[Target\_name], lag=50, ax=axes[3])

# Autocorrelation Analysis

figure, axes = plt.subplots(2,1,figsize=(12,5))

sm.tsa.graphics.plot\_acf(Y\_Data[Target\_name], lags=100, use\_vlines=True, ax=axes[0])

sm.tsa.graphics.plot\_pacf(Y\_Data[Target\_name], lags=100, use\_vlines=True, ax=axes[1])

##### Error Analysis(Statistics)

# Checking Stationarity

# Null Hypothesis: The Time-series is non-stationalry

Stationarity\_adf = stationarity\_adf\_test(Y\_Data, Target\_name)

Stationarity\_kpss = stationarity\_kpss\_test(Y\_Data, Target\_name)

# Checking of Normality

# Null Hypothesis: The residuals are normally distributed

Normality = pd.DataFrame([stats.shapiro(Y\_Data[Target\_name])],

index=['Normality'], columns=['Test Statistics', 'p-value']).T

# Checking for Autocorrelation

# Null Hypothesis: Autocorrelation is absent

Autocorrelation = pd.concat([pd.DataFrame(sm.stats.diagnostic.acorr\_ljungbox(Y\_Data[Target\_name], lags=[1,5,10,50])[0], columns=['Test Statistics']),

pd.DataFrame(sm.stats.diagnostic.acorr\_ljungbox(Y\_Data[Target\_name], lags=[1,5,10,50])[1], columns=['p-value'])], axis=1).T

Autocorrelation.columns = ['Autocorr(lag1)', 'Autocorr(lag5)', 'Autocorr(lag10)', 'Autocorr(lag50)']

# Checking Heteroscedasticity

# Null Hypothesis: Error terms are homoscedastic

Heteroscedasticity = pd.DataFrame([sm.stats.diagnostic.het\_goldfeldquandt(Y\_Data[Target\_name], X\_Data.values, alternative='two-sided')],

index=['Heteroscedasticity'], columns=['Test Statistics', 'p-value', 'Alternative']).T

Score = pd.concat([Stationarity\_adf, Stationarity\_kpss, Normality, Autocorrelation, Heteroscedasticity], join='outer', axis=1)

index\_new = ['Test Statistics', 'p-value', 'Alternative', 'Used Lag', 'Used Observations',

'Critical Value(1%)', 'Critical Value(5%)', 'Critical Value(10%)', 'Maximum Information Criteria']

Score.reindex(index\_new)

return Score

# error\_analysis(Resid\_tr\_reg1[1:], ['Error'], X\_train, graph\_on=True)

# Error Analysis(Plot)

Resid\_tr\_reg1['RowNum'] = Resid\_tr\_reg1.reset\_index().index

# Stationarity(Trend) Analysis

sns.set(palette="muted", color\_codes=True, font\_scale=2)

sns.lmplot(x='RowNum', y='Error', data=Resid\_tr\_reg1.iloc[1:],

fit\_reg='True', size=5.2, aspect=2, ci=99, sharey=True)

# Normal Distribution Analysis

figure, axes = plt.subplots(figsize=(12,8))

sns.distplot(Resid\_tr\_reg1['Error'], norm\_hist='True', fit=stats.norm)

# Lag Analysis

length = int(len(Resid\_tr\_reg1['Error'])/10)

figure, axes = plt.subplots(1, 4, figsize=(12,3))

pd.plotting.lag\_plot(Resid\_tr\_reg1['Error'], lag=1, ax=axes[0])

pd.plotting.lag\_plot(Resid\_tr\_reg1['Error'], lag=5, ax=axes[1])

pd.plotting.lag\_plot(Resid\_tr\_reg1['Error'], lag=10, ax=axes[2])

pd.plotting.lag\_plot(Resid\_tr\_reg1['Error'], lag=50, ax=axes[3])

# Autocorrelation Analysis

figure, axes = plt.subplots(2,1,figsize=(12,5))

figure = sm.graphics.tsa.plot\_acf(Resid\_tr\_reg1['Error'], lags=100, use\_vlines=True, ax=axes[0])

figure = sm.graphics.tsa.plot\_pacf(Resid\_tr\_reg1['Error'], lags=100, use\_vlines=True, ax=axes[1])

# Error Analysis(Statistics)

# Checking Stationarity

# Null Hypothesis: The Time-series is non-stationalry

Stationarity = pd.Series(sm.tsa.stattools.adfuller(Resid\_tr\_reg1['Error'])[0:4], index=['Test Statistics', 'p-value', 'Used Lag', 'Used Observations'])

for key, value in sm.tsa.stattools.adfuller(Resid\_tr\_reg1['Error'])[4].items():

Stationarity['Critical Value(%s)'%key] = value

Stationarity['Maximum Information Criteria'] = sm.tsa.stattools.adfuller(Resid\_tr\_reg1['Error'])[5]

Stationarity = pd.DataFrame(Stationarity, columns=['Stationarity'])

# Checking of Normality

# Null Hypothesis: The residuals are normally distributed

Normality = pd.DataFrame([stats.shapiro(Resid\_tr\_reg1['Error'])], index=['Normality'], columns=['Test Statistics', 'p-value']).T

# Checking for Autocorrelation

# Null Hypothesis: Autocorrelation is absent

Autocorrelation = pd.concat([pd.DataFrame(sm.stats.diagnostic.acorr\_ljungbox(Resid\_tr\_reg1['Error'], lags=[1,5,10,50])[0], columns=['Test Statistics']),

pd.DataFrame(sm.stats.diagnostic.acorr\_ljungbox(Resid\_tr\_reg1['Error'], lags=[1,5,10,50])[1], columns=['p-value'])], axis=1).T

Autocorrelation.columns = ['Autocorr(lag1)', 'Autocorr(lag5)', 'Autocorr(lag10)', 'Autocorr(lag50)']

# Checking Heteroscedasticity

# Null Hypothesis: Error terms are homoscedastic

Heteroscedasticity = pd.DataFrame([sm.stats.diagnostic.het\_goldfeldquandt(Resid\_tr\_reg1['Error'], X\_train.values, alternative='two-sided')],

index=['Heteroscedasticity'], columns=['Test Statistics', 'p-value', 'Alternative']).T

Error\_Analysis = pd.concat([Stationarity, Normality, Autocorrelation, Heteroscedasticity], join='outer', axis=1)

Error\_Analysis = Error\_Analysis.loc[['Test Statistics', 'p-value', 'Alternative', 'Used Lag', 'Used Observations',

'Critical Value(1%)', 'Critical Value(5%)', 'Critical Value(10%)',

'Maximum Information Criteria'],:]

Error\_Analysis

error\_analysis(Resid\_tr\_reg1, ['Error'], X\_train, graph\_on=True)

# 정상성 / 비정규분포 / 자기상관없음 / 등분산아님

**# ### 두 번째 잔차진단**

# 잔차진단

fit.plot\_diagnostics(figsize=(10,8))

plt.tight\_layout()

plt.show()

**# # 실행**

# ## Base 분석결과

# ### Raw Data

# Data Loading

# location = 'https://raw.githubusercontent.com/cheonbi/DataScience/master/Data/Bike\_Sharing\_Demand\_Full.csv'

location = './Data/BikeSharingDemand/Bike\_Sharing\_Demand\_Full.csv'

raw\_all = pd.read\_csv(location)

# Feature Engineering

raw\_rd = non\_feature\_engineering(raw\_all)

# Data Split

# Confirm of input and output

Y\_colname = ['count']

X\_remove = ['datetime', 'DateTime', 'temp\_group', 'casual', 'registered']

X\_colname = [x for x in raw\_rd.columns if x not in Y\_colname+X\_remove]

X\_train\_rd, X\_test\_rd, Y\_train\_rd, Y\_test\_rd = datasplit\_ts(raw\_rd, Y\_colname, X\_colname, '2012-07-01')

# Applying Base Model

fit\_reg1\_rd = sm.OLS(Y\_train\_rd, X\_train\_rd).fit()

display(fit\_reg1\_rd.summary())

pred\_tr\_reg1\_rd = fit\_reg1\_rd.predict(X\_train\_rd).values

pred\_te\_reg1\_rd = fit\_reg1\_rd.predict(X\_test\_rd).values

# Evaluation

Score\_reg1\_rd, Resid\_tr\_reg1\_rd, Resid\_te\_reg1\_rd = evaluation\_trte(Y\_train\_rd, pred\_tr\_reg1\_rd,

Y\_test\_rd, pred\_te\_reg1\_rd, graph\_on=True)

display(Score\_reg1\_rd)

# Error Analysis

error\_analysis(Resid\_tr\_reg1\_rd, ['Error'], X\_train\_rd, graph\_on=True)

# ### Feature Engineering Data

# Data Loading

# location = 'https://raw.githubusercontent.com/cheonbi/DataScience/master/Data/Bike\_Sharing\_Demand\_Full.csv'

location = './Data/BikeSharingDemand/Bike\_Sharing\_Demand\_Full.csv'

raw\_all = pd.read\_csv(location)

# Feature Engineering

raw\_fe = feature\_engineering(raw\_all)

# Data Split

# Confirm of input and output

Y\_colname = ['count']

X\_remove = ['datetime', 'DateTime', 'temp\_group', 'casual', 'registered']

X\_colname = [x for x in raw\_fe.columns if x not in Y\_colname+X\_remove]

X\_train\_fe, X\_test\_fe, Y\_train\_fe, Y\_test\_fe = datasplit\_ts(raw\_fe, Y\_colname, X\_colname, '2012-07-01')

# Applying Base Model

fit\_reg1\_fe = sm.OLS(Y\_train\_fe, X\_train\_fe).fit()

display(fit\_reg1\_fe.summary())

pred\_tr\_reg1\_fe = fit\_reg1\_fe.predict(X\_train\_fe).values

pred\_te\_reg1\_fe = fit\_reg1\_fe.predict(X\_test\_fe).values

# Evaluation

Score\_reg1\_fe, Resid\_tr\_reg1\_fe, Resid\_te\_reg1\_fe = evaluation\_trte(Y\_train\_fe, pred\_tr\_reg1\_fe,

Y\_test\_fe, pred\_te\_reg1\_fe, graph\_on=True)

display(Score\_reg1\_fe)

# Error Analysis

error\_analysis(Resid\_tr\_reg1\_fe, ['Error'], X\_train\_fe, graph\_on=True)