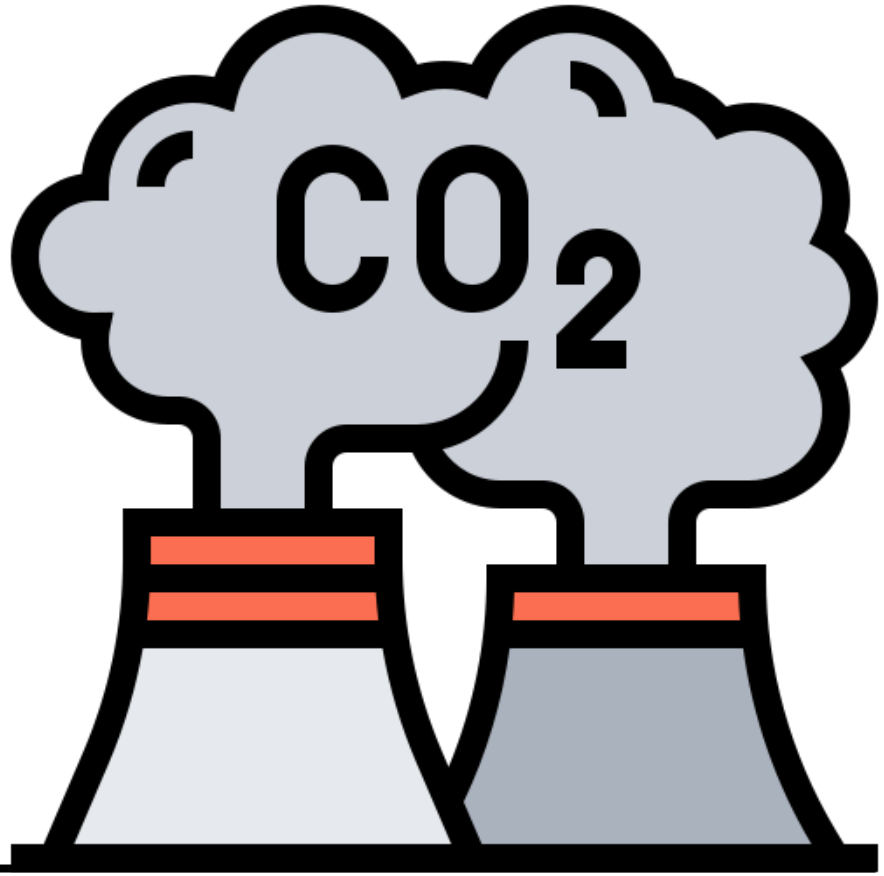
전공종합설계 및 논문(가)

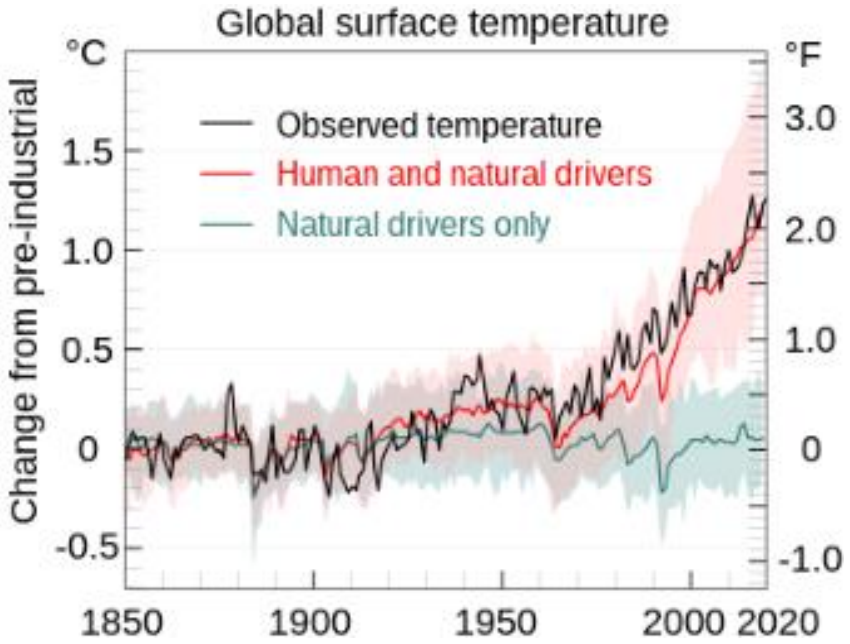
거시경제 지수를 활용한 **탄소배출권** 가격 예측

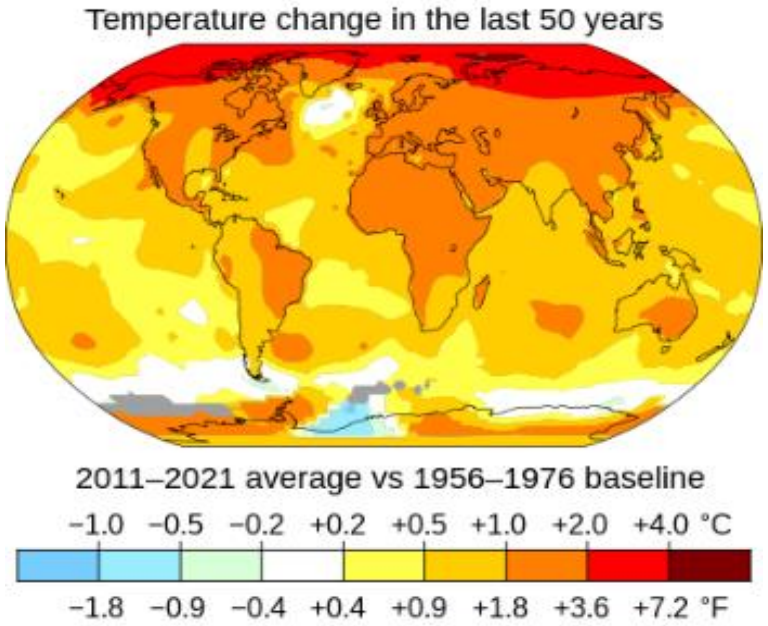
20192246 이유진
20192237 윤우석
20192263 정영훈
20192270 최수정



Contents

01 연구 배경: 탄소배출권 가격 예측 필요성





기후 위기(climate crisis): **지구 온난화**처럼 지구의 평균 기온이 점진적으로 상승하면서 전지구적 기후 패턴이 급격하게 변화하는 현상

↓

온실가스 배출을 줄이기 위해 **탄소배출권** 거래 제도 탄생

01 연구 배경: 탄소배출권 가격 예측 필요성

사업장간 자유로운 거래를 통하여 업체의 온실가스 감축활동을
유도하고 기후변화에 대응하는 "온실가스 배출권거래제"

온실가스 배출권거래제란 정부가 온실가스를 배출하는 사업장을 대상으로 연단위 배출권 할당하여 할당 범위 내에서 배출행위를 할 수 있도록 하고, 할당된 사업장의 실질적 온실가스 배출량을 평가하여 여분 또는 부족분의 배출권에 대하여는 사업장간 거래를 허용하는 제도입니다.

온실가스 배출권거래제(GHG Emissions Trading Scheme):

정부가 **온실가스**를 배출하는 사업장을 대상으로 **연단위 배출권**을 할당하는 제도

적절한 **배출권 가격 예측**을 통해 기업은 비용 효율적인 배출 감소 전략을 수립
탄소배출 감소 기술 개발 및 도입을 계획하고 투자 결정에 활용

01 연구 배경: 탄소배출권 가격 예측 필요성

탄소배출권 정책 시행은 기업들에게 '불로소득'의 기회가 되었다.
이로 인해 **탄소배출권 가격의 예측**은 주식과 마찬가지로
기업들에게 중요한 미션이 될 것이다.

본 연구는 **탄소배출권**에 어떤 거시적 요인이 영향을 주는 지 알아내어
탄소배출권 가격을 예측하는 **예측 모델**을 만들기로 하였다.

02 연구 방법

다양한 거시경제 data를 활용하여 **상관분석**을 진행하고
의미 있는 변수를 선택한다 .

여러 가지 **시계열 모델**과 **머신러닝 모델 및 딥러닝 모델**을 이용해
두 달 뒤 탄소배출권 가격을 예측한다 .

02 연구 방법

Korean Allowance Units, KAU: 대한민국에서 시행 중인 탄소배출권 거래제도에서 사용되는 단위,
탄소배출권 거래제도는 기업이 일정량의 온실가스를 배출할 때 그에 상응하는 탄소배출권을 보유해야 함을 의미한다.

KAU는 탄소배출권의 **국내 시장**에서 사용되는 단위로, **한국 내**에서 탄소배출량에 비례하여 발급되고 거래된다.
기업은 정부로부터 일정량의 KAU를 할당받아야 하며,
배출량이 할당량을 초과할 경우 추가로 KAU를 구매하여 부족한 배출량을 보완해야 한다.
반대로 배출량이 할당량보다 적을 경우, 기업은 잉여 KAU를 판매할 수 있다.

따라서 기업 간 계약을 체결할 때, **가격 예측**이 **필요!**

02 연구 방법

에코아이: **탄소배출권 거래시스템**을 운영하고,
기업들에게 탄소배출량 측정 및 보고,
온실가스 감축 방안 제안 등의 서비스를 제공하는 기업,

탄소배출권 시장 동향 분석, 환경 규제 및 정책에 대한 **컨설팅**도 수행하여
기업들이 환경 관리에 대한 전략적인 결정을 내릴 수 있도록 지원한다.

몇 달 뒤 가격을 예측하는 것이 효과적인지 자문을 구했다.

가격 예측을 할 때,
물론 기업마다 상이하지만
보통 **한 달**에서 **두 달** 정도
미래를 예측

by. 에코아이 최민아 팀장

02 연구 방법: feature 선정 이유

김수이, 『배출권거래 가격결정요인 분석과 전망』, 에너지경제연구원, 2007

정수관, 『에너지가격이 탄소배출권가격에 미치는 영향에 대한 분석』, 산업연구원, 2018

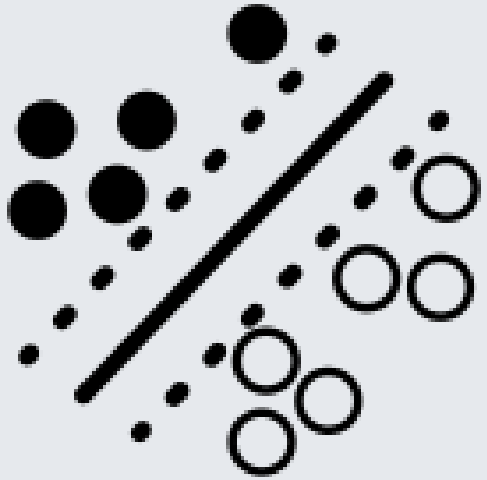
박순철 외 1명, 『한국 탄소시장에서의 배출권 가격 결정요인 분석』, 한국환경경제학, 2018

우주희 외 1명, 『국내 탄소배출권거래시장 수요측면의 가격결정요인분석』, 한국자료분석학회, 2022

등의 논문을 참고하여 feature를 결정했다.

탄소배출권 거래시장에서는 기업의 탄소배출량을 기준으로 거래되기 때문에
다양한 요소들이 가격 결정에 영향을 미친다.

02 연구 방법: 이론적 배경



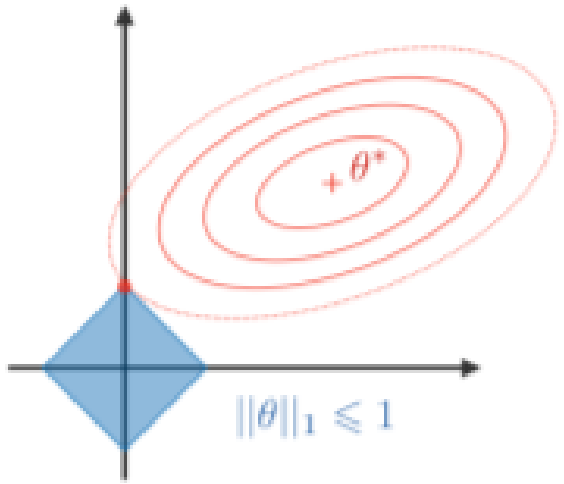
SVM(Support Vector Machines):

지도 학습 알고리즘 중 하나로,
데이터를 분류하거나 회귀하는 데 사용

SVM은 데이터를 고차원 공간으로 매핑하여
최적의 결정 경계(Decision Boundary)를
찾는 방식으로 작동

마진은 **클래스 간의 거리**를 **최대화**하는
결정 경계를 찾는 것을 목표로 한다 .

02 연구 방법: 이론적 배경



라쏘(lasso) **회귀:**

선형 회귀의 한 유형,
변수 선택과 모델 규제를 동시에 수행하여
과적합을 방지하고
모델의 **일반화 성능**을 **향상**시키는 방법

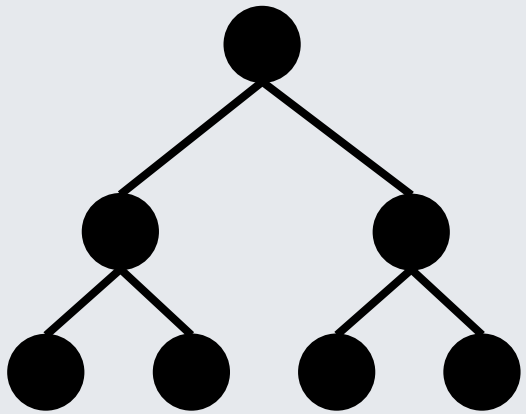
L1 규제를 사용하여 모델의 **가중치**를 **제한**하는데,
이로써 중요하지 않은 변수들의
가중치를 0으로 만들어
변수 선택의 **효과**를 얻을 수 있다.

02 연구 방법: 이론적 배경

목적 함수: $J(\beta) = RSS(y, X\beta) + \alpha * \Sigma|\beta|$

RSS: 잔차의 제곱합 (Residual Sum of Squares),
예측값과 실제값의 차이의 제곱을 모두 더한 값
y: 종속 변수
X: 설명 변수(독립 변수)를 나타내는 행렬
β: 회귀 계수
α: 규제 강도를 나타내는 하이퍼파라미터

02 연구 방법: 이론적 배경



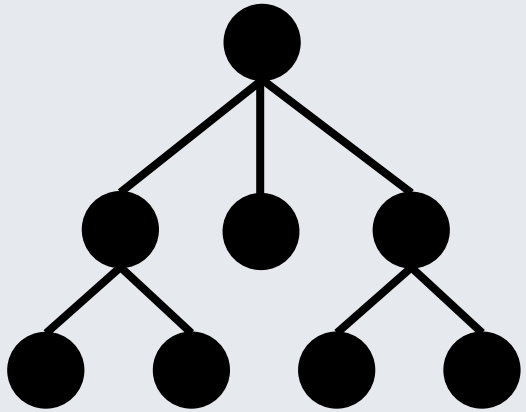
XGBoost(eXtreme Gradient Boosting):

부스팅(Boosting) 알고리즘 중 하나,
부스팅은 약한 학습기(weak learner)를
순차적으로 학습시켜
강력한 **앙상블 모델**을 만드는 알고리즘

약한 학습기인 **결정 트리**(Decision Tree)를
사용하여 순차적으로 학습한다.

그래디언트 부스팅 방법을 사용하여
오차를 줄이는 방향으로 모델을 학습시킨다.

02 연구 방법: 이론적 배경

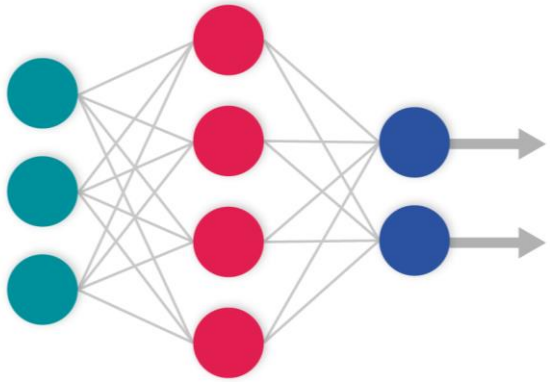


랜덤 포레스트(Random Forest):

앙상블 학습(Ensemble Learning)의 일종,
여러 개의 **의사 결정 트리**(Decision Tree)를
결합하여 예측을 수행하는 알고리즘

각각의 의사 결정 트리는
독립적으로 학습하고 예측을 수행하며,
최종 예측은 모든 트리의 예측 결과를
종합하여 결정된다.

02 연구 방법: 이론적 배경



다층 퍼셉트론(Multi-Layer Perceptron, MLP):
인공 신경망의 한 종류로,
여러 개의 **은닉층**(hidden layer)을 포함하는 **신경망** 구조
MLP는 입력층(input layer), 은닉층(hidden layer),
출력층(output layer)으로 구성되어 있다.

다층 퍼셉트론은 입력층에서
데이터를 받아 은닉층을 거쳐
출력층으로 전달하는 과정을 반복하여
데이터의 **비선형적**인 관계를 학습하고 예측한다.

02 연구 방법: 이론적 배경

ARIMA(Autoregressive Integrated Moving Average) 모델 :
시계열 데이터 분석에 사용되는 통계적 모델

ARIMA 모델은 **시계열 데이터**의 자기 상관성을 고려하여
예측 모델을 구축하는데 사용

ARIMA 모델은 일반적으로 **ARIMA(p, d, q)**로 표현된다.
p는 AR 모델의 차수, d는 차분의 차수, q는 MA 모델의 차수를 의미

02 연구 방법: 이론적 배경

ARIMA 모델은 **세 가지** 주요 구성 요소로 이루어져 있다 .

①**AR**(Autoregressive) 모델: 이전 시간 단계의 관측치에 의존하여 현재 값을 예측하는 모델.
현재 값과 이전 p개의 관측치 사이의 선형 관계를 나타낸다 .

②**MA**(Moving Average) 모델: 이전 예측 오차에 의존하여 현재 값을 예측하는 모델.
현재 값을 이전 q개의 예측 오차와 관련시킨다.

③**차분**(Difference): 시계열 데이터가 정상성을 만족하지 않을 경우, 차분을 통해 정상성을 확보.
현재 값과 이전 시간 단계의 값을 빼는 것을 의미.
시계열 데이터의 추세나 계절성을 제거하는데 도움을 준다.

02 연구 방법: 이론적 배경

PROPHET : **Facebook**에서 **개발**한 시계열 데이터를 예측하기 위한 오픈 소스 라이브러리

계절성, 휴일 효과, 비선형 추세 등의 다양한 패턴을 자동으로 감지하고 모델링할 수 있다.
다른 변수들과의 상호작용도 고려할 수 있어
시계열 데이터의 복잡한 패턴을 효과적으로 모델링할 수 있다.

주어진 시계열 데이터에서 계절성, 연간, 주간, 일간 패턴을 자동으로 추정할 수 있다.
이를 위해 Fourier 시리즈를 사용하여 계절성을 모델링하고,
가법 모델 또는 로그 모델을 선택하여 성분을 추가할 수 있다.

03 데이터 전처리

```
import pandas as pd
import numpy as np
import datetime as dt
from datetime import datetime, timedelta, date

# 탄소배출권 시세 데이터 업로드
df1 = pd.read_csv('배출권_시세_데이터(2015).csv', low_memory=False, thousands = ',')
df2 = pd.read_csv('배출권_시세_데이터(2016).csv', low_memory=False, thousands = ',')
df3 = pd.read_csv('배출권_시세_데이터(2017).csv', low_memory=False, thousands = ',')
df4 = pd.read_csv('배출권_시세_데이터(2018).csv', low_memory=False, thousands = ',')
df5 = pd.read_csv('배출권_시세_데이터(2019).csv', low_memory=False, thousands = ',')
df6 = pd.read_csv('배출권_시세_데이터(2020).csv', low_memory=False, thousands = ',')
df7 = pd.read_csv('배출권_시세_데이터(2021).csv', low_memory=False, thousands = ',')
df8 = pd.read_csv('배출권_시세_데이터(2022).csv', low_memory=False, thousands = ',')
# 연도별로 나눠져있는 것들 열방향으로 합침
df_list = [df1, df2, df3, df4, df5, df6, df7, df8]
df_all = pd.concat(df_list, ignore_index = True)

# 일자에 맞춰서 정렬
df_all = df_all.sort_values('일자')

# KAU 종목만 추출
df_all4 = df_all[df_all['종목명'].str.contains('KAU')]

# 해당연도와 KAU 연도가 동일한 종목만 추출
df_all5 = df_all4[df_all4.종목명.apply(lambda x : int(x[-2:])) == df_all4.일자.apply(lambda x : int(str(x).split('-')[0][-2:]))]

# 연도별 데이터와 월별 데이터를 조인을 위한 연도 칼럼과 월 칼럼 생성
df_all5['연도'] = pd.DatetimeIndex(df_all5['일자']).year
df_all5['월'] = pd.to_datetime(df_all5['일자']).dt.strftime('%Y-%m')
```

03 데이터 전처리

날씨 데이터 추가

```
# 일별 서울 날씨 데이터 업로드
df101 = pd.read_csv('일별 서울 날씨 데이터.csv', encoding = 'cp949', low_memory=False, thousands = ',')

# 일자에 맞추어 inner join 수행
df1001 = pd.merge(df_all5, df101, left_on='일자', right_on='날짜', how='inner')
df1001 = df1001.drop(['날짜', '지점'], axis=1)
df1001
```

	일자	종목명	종가	대비	등락률	시가	고가	저가	거래량	거래대금	가중평균	연도	월	평균기온(℃)	최저기온(℃)	최고기온(℃)
0	2015-01-12	KAU15	8640	780	9.92	7860	8640	7860	1190	9740400	8185	2015	2015-01	-2.7	-7.3	3.7
1	2015-01-13	KAU15	9500	860	9.95	9500	9500	9500	50	475000	9500	2015	2015-01	0.2	-5.1	5.8
2	2015-01-14	KAU15	9510	10	0.11	9510	9510	9510	100	951000	9510	2015	2015-01	2.5	-0.3	6.1
3	2015-01-15	KAU15	9580	70	0.74	0	0	0	0	0	0	2015	2015-01	2.8	-1.1	9.0
4	2015-01-16	KAU15	9610	30	0.31	9610	9610	9610	40	384400	9610	2015	2015-01	0.8	-2.5	3.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1958	2022-12-23	KAU22	14000	1200	9.38	12800	14000	12800	29700	402095000	13539	2022	2022-12	-11.8	-13.7	-8.6
1959	2022-12-26	KAU22	14000	0	0.00	14000	14900	14000	85210	1195774000	14033	2022	2022-12	-3.9	-7.9	1.7
1960	2022-12-27	KAU22	15000	1000	7.14	14100	15400	14100	80466	1207767900	15010	2022	2022-12	-2.6	-7.3	3.3
1961	2022-12-28	KAU22	16400	1400	9.33	15500	16450	15500	136926	2205312100	16106	2022	2022-12	-3.3	-6.0	0.1
1962	2022-12-29	KAU22	16000	-400	-2.44	16700	16900	16000	235308	3889410600	16529	2022	2022-12	-2.9	-7.8	2.1

1963 rows × 16 columns

03 데이터 전처리

유가 데이터 추가

```
# 일별 유가 가격 데이터 업로드
df102 = pd.read_csv('일별 유가 가격(주유소_평균판매가격_제품별).csv', encoding = 'cp949', low_memory=False, thousands = ',')
df102["구분"] = df102["구분"].str.replace('년', '-')
df102["구분"] = df102["구분"].str.replace('월', '-')
df102["구분"] = df102["구분"].str.replace('일', '')

# 일자에 맞추어 inner join 수행
df1002 = pd.merge(df1001, df102, left_on='일자', right_on='구분', how='inner')
df1002 = df1002.drop(['구분'], axis=1)
df1002
```

	일자	종목명	종가	대비	등락률	시가	고가	저가	거래량	거래대금	가중평균	연도	월	평균기온(℃)	최저기온(℃)	최고기온(℃)	고급휘발유	보통휘발유	자동차용경유	실내등유
0	2015-01-12	KAU15	8640	780	9.92	7860	8640	7860	1190	9740400	8185	2015	2015-01	-2.7	-7.3	3.7	1926.31	1540.37	1359.51	1065.47
1	2015-01-13	KAU15	9500	860	9.95	9500	9500	9500	50	475000	9500	2015	2015-01	0.2	-5.1	5.8	1914.07	1532.39	1352.35	1058.78
2	2015-01-14	KAU15	9510	10	0.11	9510	9510	9510	100	951000	9510	2015	2015-01	2.5	-0.3	6.1	1905.75	1524.05	1345.01	1053.16
3	2015-01-15	KAU15	9580	70	0.74	0	0	0	0	0	0	2015	2015-01	2.8	-1.1	9.0	1896.81	1514.91	1337.26	1046.88
4	2015-01-16	KAU15	9610	30	0.31	9610	9610	9610	40	384400	9610	2015	2015-01	0.8	-2.5	3.4	1890.56	1504.89	1328.80	1040.20
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1958	2022-12-23	KAU22	14000	1200	9.38	12800	14000	12800	29700	402095000	13539	2022	2022-12	-11.8	-13.7	-8.6	1832.20	1530.29	1738.09	1527.65
1959	2022-12-26	KAU22	14000	0	0.00	14000	14900	14000	85210	1195774000	14033	2022	2022-12	-3.9	-7.9	1.7	1829.38	1527.32	1731.44	1522.79
1960	2022-12-27	KAU22	15000	1000	7.14	14100	15400	14100	80466	1207767900	15010	2022	2022-12	-2.6	-7.3	3.3	1828.79	1526.31	1728.14	1520.45
1961	2022-12-28	KAU22	16400	1400	9.33	15500	16450	15500	136926	2205312100	16106	2022	2022-12	-3.3	-6.0	0.1	1827.85	1526.05	1725.84	1518.78
1962	2022-12-29	KAU22	16000	-400	-2.44	16700	16900	16000	235308	3889410600	16529	2022	2022-12	-2.9	-7.8	2.1	1827.17	1526.34	1724.28	1517.69

1963 rows × 20 columns

03 데이터 전처리

전산업 경기종합 기업경기실사 데이터 추가

```
# 월별 전산업 경기종합 기업경기실사 데이터 업로드
df103 = pd.read_csv('전산업생산_경기종합__기업경기실사_데이터 총편집본.csv', encoding = 'cp949', low_memory=False, thousands = ',')

# 월 컬럼에 맞추어 inner join 수행
df1003 = pd.merge(df1002, df103, left_on='월', right_on='시점', how='inner')
df1003 = df1003.drop(['시점'], axis=1)
df1003
```

	일자	종목명	종가	대비	등락률	시가	고가	저가	거래량	거래대금	...	건설업	서비스업	공공행정	업황실적	매출실적	수출실적	내수판매실적	생산실적	동행지수순환변동치	선행지수순환변동치
0	2015-01-12	KAU15	8640	780	9.92	7860	8640	7860	1190	9740400	...	82.3	92.1	86.9	216	256	93	88	96	99.6	100.1
1	2015-01-13	KAU15	9500	860	9.95	9500	9500	9500	50	475000	...	82.3	92.1	86.9	216	256	93	88	96	99.6	100.1
2	2015-01-14	KAU15	9510	10	0.11	9510	9510	9510	100	951000	...	82.3	92.1	86.9	216	256	93	88	96	99.6	100.1
3	2015-01-15	KAU15	9580	70	0.74	0	0	0	0	0	...	82.3	92.1	86.9	216	256	93	88	96	99.6	100.1
4	2015-01-16	KAU15	9610	30	0.31	9610	9610	9610	40	384400	...	82.3	92.1	86.9	216	256	93	88	96	99.6	100.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1958	2022-12-23	KAU22	14000	1200	9.38	12800	14000	12800	29700	402095000	...	98.3	114.6	108.4	211	246	78	75	89	99.8	98.8
1959	2022-12-26	KAU22	14000	0	0.00	14000	14900	14000	85210	1195774000	...	98.3	114.6	108.4	211	246	78	75	89	99.8	98.8
1960	2022-12-27	KAU22	15000	1000	7.14	14100	15400	14100	80466	1207767900	...	98.3	114.6	108.4	211	246	78	75	89	99.8	98.8
1961	2022-12-28	KAU22	16400	1400	9.33	15500	16450	15500	136926	2205312100	...	98.3	114.6	108.4	211	246	78	75	89	99.8	98.8
1962	2022-12-29	KAU22	16000	-400	-2.44	16700	16900	16000	235308	3889410600	...	98.3	114.6	108.4	211	246	78	75	89	99.8	98.8

1963 rows × 32 columns

03 데이터 전처리

강수량, smp, 천연가스가격, 코스피지수, 환율 데이터 추가

```
# 강수량, smp, 천연가스가격, 코스피지수, 환율 데이터 업로드
df104 = pd.read_csv('강수량_smp_천연가스가격_코스피지수_환율_데이터.csv', encoding = 'UTF8', low_memory=False,thousands = ',')

# 일자 컬럼에 맞추어 inner join 수행
df1004 = pd.merge(df1003, df104, left_on='일자', right_on='Date', how='left')
df1004 = df1004.drop(['Date', '석탄가격'], axis=1)

# 강수량 결측치는 비가 오지 않은 날이기에 0으로 대체
df1004["강수량"] = df1004["강수량"].fillna(0)
df1004
```

수관매실적	생산실적	동행지수순환변동치	선행지수순환변동치	강수량	smp	천연가스 가격	코스피지수	환율
88	96	99.6	100.1	0.0	NaN	2.795	1920.95	1082.72
88	96	99.6	100.1	0.0	NaN	2.943	1917.14	1078.78
88	96	99.6	100.1	0.0	NaN	3.233	1913.66	1082.85
88	96	99.6	100.1	0.0	NaN	3.158	1914.14	1075.69
88	96	99.6	100.1	0.3	NaN	3.127	1888.13	1079.07
...	...	...	...	...	...	...	...	...
75	89	99.8	98.8	0.0	279.42	4.980	2313.69	1279.55
75	89	99.8	98.8	0.0	269.25	5.144	2317.14	1273.91
75	89	99.8	98.8	0.0	275.74	5.282	2332.79	1271.79
75	89	99.8	98.8	0.1	276.34	4.709	2280.45	1274.37
75	89	99.8	98.8	0.0	277.34	4.559	2236.40	1260.85

```
# smp 데이터 결측치 값은 통계 데이터를 찾아 삽입
df1004.loc[df1004['월'] == '2015-01', 'smp'] = 140.54
df1004.loc[df1004['월'] == '2015-02', 'smp'] = 121.16
df1004.loc[df1004['월'] == '2015-03', 'smp'] = 118.36
df1004.loc[df1004['월'] == '2015-04', 'smp'] = 103.45
df1004.loc[df1004['월'] == '2015-05', 'smp'] = 96.31
df1004.loc[df1004['월'] == '2015-06', 'smp'] = 84.13
df1004.loc[df1004['월'] == '2015-07', 'smp'] = 81.53
df1004.loc[df1004['월'] == '2015-08', 'smp'] = 88.15
df1004.loc[df1004['월'] == '2015-09', 'smp'] = 90.73
df1004.loc[df1004['월'] == '2015-10', 'smp'] = 98.29
df1004.loc[df1004['월'] == '2015-11', 'smp'] = 94.89
```

03 데이터 전처리

유연탄 가격 데이터 추가

```
# 주별 유연탄 가격 데이터 업로드
df105 = pd.read_csv('주별 유연탄 가격(2015-2022).csv', encoding = 'UTF8', low_memory=False, thousands = ',')
df105 = df105.drop(['Unnamed: 2','Unnamed: 3', 'Unnamed: 4', 'Unnamed: 5'], axis=1)
df105 = df105.dropna(axis=0)

# 동일 월에 해당하는 주는 평균을 내서 월별 데이터로 사용
df105['Date'] = pd.to_datetime(df105['Date']).dt.strftime('%Y-%m')
df105 = df105.groupby(['Date'], as_index=False).mean()
df105

# 월 컬럼에 맞추어 inner join 수행
df1005 = pd.merge(df1004, df105, left_on='월', right_on='Date', how='inner')
df1005 = df1005.drop(['Date'], axis=1)
df1005
```

수순환변동치	선행지수순환변동치	강수량	smp	천연가스 가격	코스피지수	환율	석탄가격
99.6	100.1	0.0	140.54	2.795	1920.95	1082.72	58.814
99.6	100.1	0.0	140.54	2.943	1917.14	1078.78	58.814
99.6	100.1	0.0	140.54	3.233	1913.66	1082.85	58.814
99.6	100.1	0.0	140.54	3.158	1914.14	1075.69	58.814
99.6	100.1	0.3	140.54	3.127	1888.13	1079.07	58.814
...	...	...	...	...	...	...	...
99.8	98.8	0.0	279.42	4.980	2313.69	1279.55	159.556
99.8	98.8	0.0	269.25	5.144	2317.14	1273.91	159.556
99.8	98.8	0.0	275.74	5.282	2332.79	1271.79	159.556
99.8	98.8	0.1	276.34	4.709	2280.45	1274.37	159.556
99.8	98.8	0.0	277.34	4.559	2236.40	1260.85	159.556

03 데이터 전처리

배출권 거래 제도 칼럼 추가

```
# 배출권 거래 제도 칼럼을 기한에 맞게 데이터 할당
df1005.loc[df1005['연도'] <= 2017, '배출권 거래 제도'] = 1
df1005.loc[(df1005['연도'] >= 2018) & (df1005['연도'] <= 2020), '배출권 거래 제도'] = 2
df1005.loc[df1005['연도'] >= 2021 , '배출권 거래 제도'] = 3

# 배출권 거래 제도 칼럼을 범주형 변수로 지정
df1005['배출권 거래 제도'] = df1005['배출권 거래 제도'].astype('category')
df1005
```

	일자	종목명	종가	대비	등락률	시가	고가	저가	거래량	거래대금	...	생산실적	동행지수순환변동치	선행지수순환변동치	강수량	smp	천연가스 가격	코스피지수	환율	석탄가격	배출권 거래 제도
0	2015-01-12	KAU15	8640	780	9.92	7860	8640	7860	1190	9740400	...	96	99.6	100.1	0.0	140.54	2.795	1920.95	1082.72	58.814	1.0
1	2015-01-13	KAU15	9500	860	9.95	9500	9500	9500	50	475000	...	96	99.6	100.1	0.0	140.54	2.943	1917.14	1078.78	58.814	1.0
2	2015-01-14	KAU15	9510	10	0.11	9510	9510	9510	100	951000	...	96	99.6	100.1	0.0	140.54	3.233	1913.66	1082.85	58.814	1.0
3	2015-01-15	KAU15	9580	70	0.74	0	0	0	0	0	...	96	99.6	100.1	0.0	140.54	3.158	1914.14	1075.69	58.814	1.0
4	2015-01-16	KAU15	9610	30	0.31	9610	9610	9610	40	384400	...	96	99.6	100.1	0.3	140.54	3.127	1888.13	1079.07	58.814	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1958	2022-12-23	KAU22	14000	1200	9.38	12800	14000	12800	29700	402095000	...	89	99.8	98.8	0.0	279.42	4.980	2313.69	1279.55	159.556	3.0
1959	2022-12-26	KAU22	14000	0	0.00	14000	14900	14000	85210	1195774000	...	89	99.8	98.8	0.0	269.25	5.144	2317.14	1273.91	159.556	3.0
1960	2022-12-27	KAU22	15000	1000	7.14	14100	15400	14100	80466	1207767900	...	89	99.8	98.8	0.0	275.74	5.282	2332.79	1271.79	159.556	3.0
1961	2022-12-28	KAU22	16400	1400	9.33	15500	16450	15500	136926	2205312100	...	89	99.8	98.8	0.1	276.34	4.709	2280.45	1274.37	159.556	3.0
1962	2022-12-29	KAU22	16000	-400	-2.44	16700	16900	16000	235308	3889410600	...	89	99.8	98.8	0.0	277.34	4.559	2236.40	1260.85	159.556	3.0

1963 rows × 39 columns

03 데이터 전처리

코로나 확진자 수, 사회적 거리두기 단계 데이터 추가

```
# 코로나 확진자수 데이터 업로드
df106 = pd.read_csv('서울시 코로나19 확진자 발생동향.csv', low_memory=False, thousands = ',', encoding = "cp949")
df106.rename(columns={'전국 기준일' : '일자', '전국 추가 확진' : '전국 코로나 확진자 수'}, inplace=True)

# 일자에 맞추서 left join 수행
df1006 = df1005.merge(df106, on='일자', how='left')

# Nan 값인 경우는 확진자 수가 없었다는 의미이기에 0으로 대체
df1006['전국 코로나 확진자 수'].fillna(0, inplace=True)

# 코로나 사회적 거리두기 단계 칼럼 추가
df1006['일자'] = pd.DatetimeIndex(df1006['일자']).strftime('%Y-%m-%d')
df1006['사회적 거리두기 단계'] = 0
df1006.loc[(df1006['일자'] >= '2020-05-06') & (df1006['일자'] <= '2020-08-15'), '사회적 거리두기 단계'] = 1
df1006.loc[(df1006['일자'] >= '2020-08-16') & (df1006['일자'] <= '2020-08-29'), '사회적 거리두기 단계'] = 3
df1006.loc[(df1006['일자'] >= '2020-08-30') & (df1006['일자'] <= '2020-09-13'), '사회적 거리두기 단계'] = 4
df1006.loc[(df1006['일자'] >= '2020-09-14') & (df1006['일자'] <= '2020-10-11'), '사회적 거리두기 단계'] = 3
df1006.loc[(df1006['일자'] >= '2020-10-12') & (df1006['일자'] <= '2020-11-18'), '사회적 거리두기 단계'] = 1
df1006.loc[(df1006['일자'] >= '2020-11-19') & (df1006['일자'] <= '2020-11-23'), '사회적 거리두기 단계'] = 2
df1006.loc[(df1006['일자'] >= '2020-11-24') & (df1006['일자'] <= '2020-12-07'), '사회적 거리두기 단계'] = 3
df1006.loc[(df1006['일자'] >= '2020-12-08') & (df1006['일자'] <= '2021-02-14'), '사회적 거리두기 단계'] = 4
df1006.loc[(df1006['일자'] >= '2021-02-15') & (df1006['일자'] <= '2021-06-30'), '사회적 거리두기 단계'] = 3
df1006.loc[(df1006['일자'] >= '2021-07-01') & (df1006['일자'] <= '2021-07-24'), '사회적 거리두기 단계'] = 1
df1006.loc[(df1006['일자'] >= '2021-07-25') & (df1006['일자'] <= '2021-10-31'), '사회적 거리두기 단계'] = 5
df1006.loc[(df1006['일자'] >= '2021-11-01') & (df1006['일자'] <= '2022-04-17'), '사회적 거리두기 단계'] = 1

# 사회적 거리두기 단계 칼럼을 범주형 변수로 지정
df1006['사회적 거리두기 단계'] = df1006['사회적 거리두기 단계'].astype('category')
```

category	사회적 거리두기 단계
0	거리두기 시행 X
1	1단계
2	1.5단계
3	2단계
4	2.5단계
5	4단계

03 데이터 전처리

종속변수 칼럼 추가

```
# 종속변수로 8주 뒤 배출권 가격(종가) 칼럼 데이터 추가
df_8wks = df1006.copy().iloc[:, [0,2]]
df_8wks.일자 = df_8wks.일자.apply(lambda x : datetime.strptime(str(x),'%Y-%m-%d') - timedelta(days=28+2)).apply(lambda x: int(''.join(str(x).split('-'))[:8]))
df_8wks.rename(columns={'종가':'8주뒤배출권가격(종가)'},inplace=True)
df_8wks['일자'] = df_8wks['일자'].astype(str)
df_8wks['일자'] = pd.to_datetime(df_8wks['일자'], format='%Y%m%d').dt.strftime('%Y-%m-%d')

# 일자 컬럼에 맞추어 inner join 수행
df1007 = pd.merge(df1006, df_8wks, left_on='일자', right_on='일자', how='inner')
df1007
```

	일자	종목명	종가	대비	등락률	시가	고가	저가	거래량	거래대금	...	강수량	smp	천연가스 가격	코스피지수	환율	석탄가격	배출권 거래 제도	전국 코로나 확진자 수	사회적 거리두기 단계	8주뒤배출권가격(종가)
0	2015-01-12	KAU15	8640	780	9.92	7860	8640	7860	1190	9740400	...	0.0	140.54	2.795	1920.95	1082.72	58.8140	1.0	0.0	0	10100
1	2015-01-13	KAU15	9500	860	9.95	9500	9500	9500	50	475000	...	0.0	140.54	2.943	1917.14	1078.78	58.8140	1.0	0.0	0	10100
2	2015-01-14	KAU15	9510	10	0.11	9510	9510	9510	100	951000	...	0.0	140.54	3.233	1913.66	1082.85	58.8140	1.0	0.0	0	10100
3	2015-01-15	KAU15	9580	70	0.74	0	0	0	0	0	...	0.0	140.54	3.158	1914.14	1075.69	58.8140	1.0	0.0	0	10100
4	2015-01-16	KAU15	9610	30	0.31	9610	9610	9610	40	384400	...	0.3	140.54	3.127	1888.13	1079.07	58.8140	1.0	0.0	0	10100
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1808	2022-10-28	KAU22	21450	200	0.94	22850	22900	21200	14020	300694500	...	0.0	253.30	5.684	2268.40	1421.66	190.1175	3.0	35924.0	0	14000
1809	2022-10-31	KAU22	21100	-350	-1.63	21800	21800	19450	17650	365209250	...	0.0	245.31	6.355	2293.61	1425.83	190.1175	3.0	18510.0	0	14000
1810	2022-11-01	KAU22	21300	200	0.95	21450	21450	21300	4950	105910500	...	0.0	253.90	5.714	2335.22	1416.73	175.3375	3.0	58379.0	0	15000
1811	2022-11-02	KAU22	21100	-200	-0.94	21200	21250	21100	20280	429076000	...	0.0	255.02	6.268	2336.87	1412.45	175.3375	3.0	54766.0	0	16400
1812	2022-11-03	KAU22	20800	-300	-1.42	21000	21050	20800	21970	458524500	...	0.0	258.66	5.975	2329.17	1423.70	175.3375	3.0	46896.0	0	16000

1813 rows × 42 columns

03 데이터 전처리

```
# feature로 사용하지 않을 열 제거
df1007 = df1007.drop(['종목명', '시가', '고가', '저가', '가중평균','연도', '월', '대비', '등락률', '거래량', '거래대금'], axis=1)
df1007
```

	일자	종가	평균기온(℃)	최저기온(℃)	최고기온(℃)	고급휘발유	보통휘발유	자동차용경유	실내등유	전산업생산지수	...	강수량	smp	천연가스 가격	코스피지수	환율	석탄가격	배출권 거래 제도	전국 코로나 확진자 수	사회적 거리두기 단계	8주뒤배출권가격(종가)
0	2015-01-12	8640	-2.7	-7.3	3.7	1926.31	1540.37	1359.51	1065.47	91.5	...	0.0	140.54	2.795	1920.95	1082.72	58.8140	1.0	0.0	0	10100
1	2015-01-13	9500	0.2	-5.1	5.8	1914.07	1532.39	1352.35	1058.78	91.5	...	0.0	140.54	2.943	1917.14	1078.78	58.8140	1.0	0.0	0	10100
2	2015-01-14	9510	2.5	-0.3	6.1	1905.75	1524.05	1345.01	1053.16	91.5	...	0.0	140.54	3.233	1913.66	1082.85	58.8140	1.0	0.0	0	10100
3	2015-01-15	9580	2.8	-1.1	9.0	1896.81	1514.91	1337.26	1046.88	91.5	...	0.0	140.54	3.158	1914.14	1075.69	58.8140	1.0	0.0	0	10100
4	2015-01-16	9610	0.8	-2.5	3.4	1890.56	1504.89	1328.80	1040.20	91.5	...	0.3	140.54	3.127	1888.13	1079.07	58.8140	1.0	0.0	0	10100
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1808	2022-10-28	21450	13.6	8.2	20.6	1947.93	1661.71	1864.82	1600.90	109.4	...	0.0	253.30	5.684	2268.40	1421.66	190.1175	3.0	35924.0	0	14000
1809	2022-10-31	21100	14.9	11.2	21.1	1947.01	1660.70	1869.29	1601.52	109.4	...	0.0	245.31	6.355	2293.61	1425.83	190.1175	3.0	18510.0	0	14000
1810	2022-11-01	21300	13.1	9.4	17.8	1945.99	1659.88	1872.05	1601.86	108.9	...	0.0	253.90	5.714	2335.22	1416.73	175.3375	3.0	58379.0	0	15000
1811	2022-11-02	21100	11.1	5.5	17.9	1944.18	1659.38	1874.01	1602.02	108.9	...	0.0	255.02	6.268	2336.87	1412.45	175.3375	3.0	54766.0	0	16400
1812	2022-11-03	20800	9.3	5.6	13.4	1943.64	1658.83	1875.90	1602.00	108.9	...	0.0	258.66	5.975	2329.17	1423.70	175.3375	3.0	46896.0	0	16000

1813 rows × 31 columns

03 데이터 전처리

```
종가                  0.848514
평균기온(℃)           0.013662
최저기온(℃)           0.021107
최고기온(℃)          -0.000722
고급휘발유            0.037018
보통휘발유            0.196085
자동차용경유          0.218713
실내등유              0.147774
전산업생산지수        0.516696
광공업                0.425812
건설업                0.418495
서비스업              0.477404
공공행정              0.507946
업황실적              0.020250
매출실적             -0.024261
수출실적              0.059088
내수판매실적         -0.027593
생산실적             -0.051718
동행지수순환변동치    -0.049331
선행지수순환변동치    -0.278964
강수량                0.036714
smp                -0.021889
천연가스 가격         0.095377
코스피지수            0.215585
환율                 0.144690
석탄가격              0.208626
전국 코로나 확진자 수  -0.003195
8주뒤배출권가격(종가)   1.000000
Name: 8주뒤배출권가격(종가), dtype: float64
```

종속변수와 **상관계수**가 0.2 **이하**인 변수 제거

	일자	종가	자동차용경유	전산업생산지수	광공업	건설업	서비스업	공공행정	선행지수순환변동치	코스피지수	석탄가격	배출권 거래 제도	사회적 거리두기 단계	8주뒤배출권가격(종가)
0	2015-01-12	8640	1359.51	91.5	93.0	82.3	92.1	86.9	100.1	1920.95	58.8140	1.0	0	10100
1	2015-01-13	9500	1352.35	91.5	93.0	82.3	92.1	86.9	100.1	1917.14	58.8140	1.0	0	10100
2	2015-01-14	9510	1345.01	91.5	93.0	82.3	92.1	86.9	100.1	1913.66	58.8140	1.0	0	10100
3	2015-01-15	9580	1337.26	91.5	93.0	82.3	92.1	86.9	100.1	1914.14	58.8140	1.0	0	10100
4	2015-01-16	9610	1328.80	91.5	93.0	82.3	92.1	86.9	100.1	1888.13	58.8140	1.0	0	10100
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1808	2022-10-28	21450	1864.82	109.4	103.7	99.8	113.7	105.7	99.5	2268.40	190.1175	3.0	0	14000
1809	2022-10-31	21100	1869.29	109.4	103.7	99.8	113.7	105.7	99.5	2293.61	190.1175	3.0	0	14000
1810	2022-11-01	21300	1872.05	108.9	103.3	99.1	112.8	108.8	99.3	2335.22	175.3375	3.0	0	15000
1811	2022-11-02	21100	1874.01	108.9	103.3	99.1	112.8	108.8	99.3	2336.87	175.3375	3.0	0	16400
1812	2022-11-03	20800	1875.90	108.9	103.3	99.1	112.8	108.8	99.3	2329.17	175.3375	3.0	0	16000

1813 rows × 14 columns

종속변수와 각 독립변수의 상관계수

03 데이터 전처리

```
# 사회적 거리두기 단계별 종속 변수와 상관계수
df1010 = pd.get_dummies(df1008['사회적 거리두기 단계'])
df1010.corrwith(df1008['8주뒤배출권가격(종가)'])

# 배출권 거래 제도별 종속 변수와 상관계수
df1011 = pd.get_dummies(df1008['배출권 거래 제도'])
df1011.corrwith(df1008['8주뒤배출권가격(종가)'])
```

One-hot encoding 진행

```
0   -0.148948
1    0.156663
2   -0.024696
3   -0.070690
4   -0.043054
5    0.224398
dtype: float64
```

배출권 거래 제도

```
1.0   -0.646025
2.0    0.524824
3.0    0.137272
dtype: float64
```

거리두기 단계

독립변수에 대한 카테고리 별 **상관계수**

배출권 거래 제도	사회적 거리두기 단계
1.0	0
1.0	0
1.0	0
1.0	0
1.0	0
...	...
3.0	0
3.0	0
3.0	0
3.0	0
3.0	0

사회적 거리두기 4단계	1차 계획기간	2차 계획기간
0	1	0
0	1	0
0	1	0
0	1	0
0	1	0
...	...	...
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

03 데이터 전처리

독립변수 간에 상관계수가 높은 상태(**다중 공선성**이 높음)

	종가	자동차용경유	전산업생산지수	광공업	건설업	서비스업	공공행정	선행지수순환변동치	코스피지수	석탄가격	8주뒤배출권가격(종가)	사회적 거리두기 4단계	1차 계획기간	2차 계획기간
종가	1.000000	0.204289	0.537140	0.431354	0.427620	0.498169	0.559293	-0.353070	0.169968	0.250589	0.848514	0.137549	-0.679339	0.537975
자동차용경유	0.204289	1.000000	0.744937	0.663316	-0.122603	0.797780	0.585080	0.138838	0.312664	0.885157	0.218713	0.093107	-0.383968	-0.191111
전산업생산지수	0.537140	0.744937	1.000000	0.938390	0.141402	0.977187	0.891375	0.131149	0.651048	0.825845	0.516696	0.196487	-0.710014	0.010545
광공업	0.431354	0.663316	0.938390	1.000000	0.012599	0.860317	0.864537	0.276620	0.737364	0.774444	0.425812	0.237097	-0.632738	-0.099007
건설업	0.427620	-0.122603	0.141402	0.012599	1.000000	0.088669	-0.078130	-0.104917	-0.050623	-0.065936	0.418495	-0.141533	-0.137072	0.402698
서비스업	0.498169	0.797780	0.977187	0.860317	0.088669	1.000000	0.851810	0.065981	0.572974	0.843008	0.477404	0.168455	-0.689556	0.006280
공공행정	0.559293	0.585080	0.891375	0.864537	-0.078130	0.851810	1.000000	0.066655	0.658749	0.699043	0.507946	0.293683	-0.741700	0.035072
선행지수순환변동치	-0.353070	0.138838	0.131149	0.276620	-0.104917	0.065981	0.066655	1.000000	0.665533	0.275204	-0.278964	0.269756	0.212055	-0.591425
코스피지수	0.169968	0.312664	0.651048	0.737364	-0.050623	0.572974	0.658749	0.665533	1.000000	0.518005	0.215585	0.382222	-0.458943	-0.226673
석탄가격	0.250589	0.885157	0.825845	0.774444	-0.065936	0.843008	0.699043	0.275204	0.518005	1.000000	0.208626	0.287736	-0.385209	-0.321139
8주뒤배출권가격(종가)	0.848514	0.218713	0.516696	0.425812	0.418495	0.477404	0.507946	-0.278964	0.215585	0.208626	1.000000	0.224398	-0.646025	0.524824
사회적 거리두기 4단계	0.137549	0.093107	0.196487	0.237097	-0.141533	0.168455	0.293683	0.269756	0.382222	0.287736	0.224398	1.000000	-0.144678	-0.145357
1차 계획기간	-0.679339	-0.383968	-0.710014	-0.632738	-0.137072	-0.689556	-0.741700	0.212055	-0.458943	-0.385209	-0.646025	-0.144678	1.000000	-0.614424
2차 계획기간	0.537975	-0.191111	0.010545	-0.099007	0.402698	0.006280	0.035072	-0.591425	-0.226673	-0.321139	0.524824	-0.145357	-0.614424	1.000000

03 데이터 전처리

변수 제거를 통해 독립변수 간에 상관성 감소

	종가	건설업	공공행정	선행지수순환변동치	석탄가격	사회적 거리두기 4단계	1차 계획기간	2차 계획기간
종가	1.000000	0.427620	0.559293	-0.353070	0.250589	0.137549	-0.679339	0.537975
건설업	0.427620	1.000000	-0.078130	-0.104917	-0.065936	-0.141533	-0.137072	0.402698
공공행정	0.559293	-0.078130	1.000000	0.066655	0.699043	0.293683	-0.741700	0.035072
선행지수순환변동치	-0.353070	-0.104917	0.066655	1.000000	0.275204	0.269756	0.212055	-0.591425
석탄가격	0.250589	-0.065936	0.699043	0.275204	1.000000	0.287736	-0.385209	-0.321139
사회적 거리두기 4단계	0.137549	-0.141533	0.293683	0.269756	0.287736	1.000000	-0.144678	-0.145357
1차 계획기간	-0.679339	-0.137072	-0.741700	0.212055	-0.385209	-0.144678	1.000000	-0.614424
2차 계획기간	0.537975	0.402698	0.035072	-0.591425	-0.321139	-0.145357	-0.614424	1.000000

03 데이터 전처리

최종 전처리 데이터 셋

```
# feature 간의 상관관계가 0.75 이상인 feature 제거
df2004 = df2003.drop(['자동차용경유', '광공업', '서비스업', '코스피지수','전산업생산지수', '배출권 거래 제도', '사회적 거리두기 단계'], axis=1)
```

	일자	종가	건설업	공공행정	선행지수순환변동치	석탄가격	사회적 거리두기 4단계	1차 계획기간	2차 계획기간	8주뒤배출권가격(종가)
0	2015-01-12	8640	82.3	86.9	100.1	58.8140	0	1	0	10100
1	2015-01-13	9500	82.3	86.9	100.1	58.8140	0	1	0	10100
2	2015-01-14	9510	82.3	86.9	100.1	58.8140	0	1	0	10100
3	2015-01-15	9580	82.3	86.9	100.1	58.8140	0	1	0	10100
4	2015-01-16	9610	82.3	86.9	100.1	58.8140	0	1	0	10100
...	...	...	...	...	...	...	...	...	...	...
1808	2022-10-28	21450	99.8	105.7	99.5	190.1175	0	0	0	14000
1809	2022-10-31	21100	99.8	105.7	99.5	190.1175	0	0	0	14000
1810	2022-11-01	21300	99.1	108.8	99.3	175.3375	0	0	0	15000
1811	2022-11-02	21100	99.1	108.8	99.3	175.3375	0	0	0	16400
1812	2022-11-03	20800	99.1	108.8	99.3	175.3375	0	0	0	16000

1813 rows × 10 columns

03 데이터 전처리: 데이터셋 구성

속성	데이터 예시
일자	2015-01-12
종가	8640
건설업	82.3
공공행정	86.9
선행지수순환변동치	100.1

속성	데이터 예시
석탄가격	58.8140
사회적 거리두기 4단계	0
1차 계획기간	1
2차 계획기간	0
8주뒤배출권가격(종가)	10,100

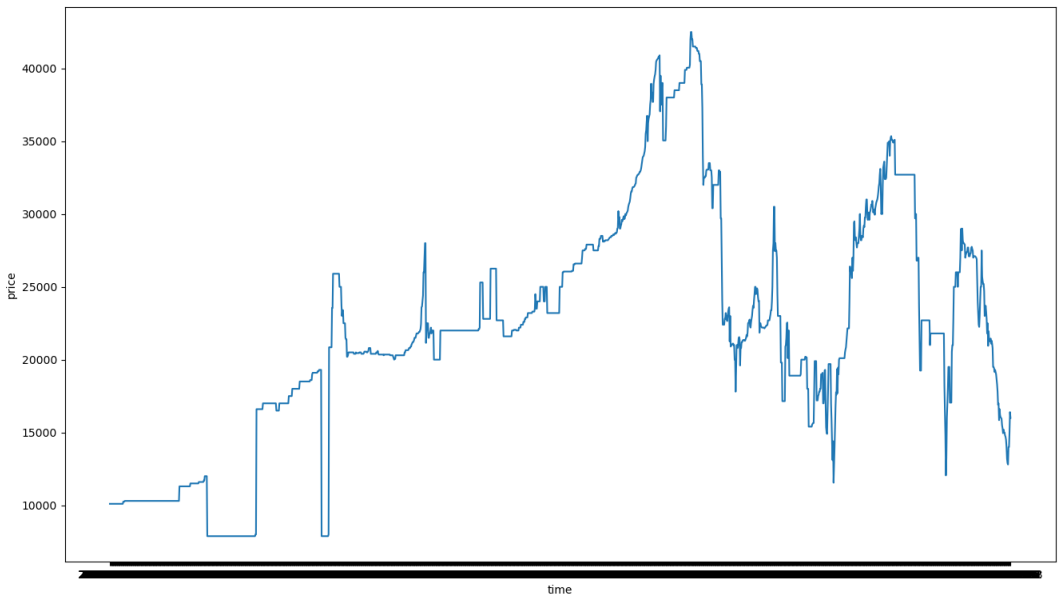
04 모델링: SVM

파일 열기

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import math

import pandas as pd

# 파일 경로
file_path = "C:/Users/JeongFM/Desktop/2022_3학년_1학기/전종설/Carbon_price_일자 포함 데이터.csv"

# 인코딩 방식 리스트
encodings = ['utf-8', 'euc-kr', 'latin1']

# 각 인코딩 방식으로 파일 열기 시도
df = None
for encoding in encodings:
    try:
        df = pd.read_csv(file_path, encoding=encoding)
        break
    except UnicodeDecodeError:
        continue

# 파일 열기 성공 여부 확인
if df is None:
    print("파일 열기에 실패했습니다.")
else:
    print("파일 열기 성공!")
```

일자와 가격 간의 그래프 그리기 → 데이터의 주기성 확인

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(16, 9))
sns.lineplot(y=df['8주뒤배출권가격(종가)'], x=df['일자'])
plt.xlabel('time')
plt.ylabel('price')
```



04 모델링: SVM

일자를 Index로 설정 → Visualization

```
df['일자'] = pd.to_datetime(df['일자'], format='%Y-%m-%d')
df['일자'] = pd.to_datetime(df['일자'], unit='s')
df.set_index('일자', inplace=True)
df
```

일자	종가	건설업	공공행정	선행지수순환변동치	석탄가격	사회적 거리두기 4단계	1차 계획기간	2차 계획기간	8주뒤배출권가격(종가)
2015-01-12	8640	82.3	86.9	100.1	58.8140	0	1	0	10100
2015-01-13	9500	82.3	86.9	100.1	58.8140	0	1	0	10100
2015-01-14	9510	82.3	86.9	100.1	58.8140	0	1	0	10100
2015-01-15	9580	82.3	86.9	100.1	58.8140	0	1	0	10100
2015-01-16	9610	82.3	86.9	100.1	58.8140	0	1	0	10100
...	...	...	...	...	...	...	...	...	...
2022-10-28	21450	99.8	105.7	99.5	190.1175	0	0	0	14000
2022-10-31	21100	99.8	105.7	99.5	190.1175	0	0	0	14000
2022-11-01	21300	99.1	108.8	99.3	175.3375	0	0	0	15000
2022-11-02	21100	99.1	108.8	99.3	175.3375	0	0	0	16400
2022-11-03	20800	99.1	108.8	99.3	175.3375	0	0	0	16000

1813 rows × 9 columns

04 모델링: SVM

정규화 수행(MinMax 사용)

```
from sklearn.preprocessing import MinMaxScaler

# '8주뒤배출권가격(종가)'를 제외한 독립변수 선택
columns_to_normalize = df.drop('8주뒤배출권가격(종가)', axis=1).columns

# 정규화 수행
scaler = MinMaxScaler()
df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])
df.dropna(inplace=True)
df.isnull().sum()
```

```
종가                0
건설업               0
공공행정              0
선행지수순환변동치         0
석탄가격              0
사회적 거리두기 4단계      0
1차 계획기간           0
2차 계획기간           0
8주뒤배출권가격(종가)      0
dtype: int64
```

SVM 모델 생성

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import math

# 독립 변수(X)와 종속 변수(y) 설정
X = df.drop("8주뒤배출권가격(종가)", axis=1)
y = df["8주뒤배출권가격(종가)"]

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# SVM 회귀 모델 생성
svm_model = SVR(C=2.9, epsilon=0.2, kernel='linear')
svm_model.fit(X_train, y_train)

# 예측
y_pred = svm_model.predict(X_test)

# RMSE 계산
rmse = math.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE:", rmse)

r2_score = svm_model.score(X_test, y_test)
print("R2 Score:", r2_score)


kernel_list = ['rbf', 'poly', 'sigmoid']

for kernel in kernel_list:
    reg = SVR(C=10, epsilon=0.2, kernel=kernel) # SVR 생성
    reg.fit(X_train, y_train)

    # 예측
    y_pred = reg.predict(X_test)

    # RMSE 계산
    rmse = math.sqrt(mean_squared_error(y_test, y_pred))
    print("RMSE:", rmse)

    r2_score = svm_model.score(X_test, y_test)
    print("R2 Score:", r2_score)
```

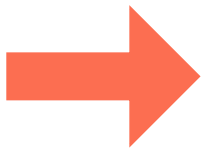
```
RMSE: 7072.0553895536705
R2 Score: 0.16098425836184194
RMSE: 6619.176619384001
R2 Score: 0.16098425836184194
RMSE: 4376.989293802981
R2 Score: 0.16098425836184194
RMSE: 6840.811691866695
R2 Score: 0.16098425836184194
```

04 모델링: SVM

```python
import numpy as np
from sklearn.model_selection import GridSearchCV

## Grid Search 적용 파라미터
kernel_list = ['linear', 'rbf', 'poly']
C_list = np.arange(1, 15, 1)
## parameter 키는 svr.get_params().keys()를 통해 확인
print(svm_model.get_params().keys())
parameters = {'kernel':kernel_list, 'C':C_list}

## Grid Search 수행
reg = GridSearchCV(svm_model, parameters, cv=5)
reg.fit(X_train, y_train)
```

```python
## 결과 확인
print('최적 파라미터 값 : ', reg.best_params_)
print('점수 : ', reg.best_score_)
```

```
최적 파라미터 값 :  {'C': 14, 'kernel': 'poly'}
점수 :  0.6941154267767283
```



최적 파라미터로 SVM 모델 생성

```python
# SVM 회귀 모델 생성
svm_model = SVR(C=14, epsilon=0.2, kernel='poly')
svm_model.fit(X_train, y_train)

# 예측
y_pred = svm_model.predict(X_test)

# RMSE 계산
rmse = math.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE:", rmse)

r2_score = svm_model.score(X_test, y_test)
print("R2 Score:", r2_score)
```

```
RMSE: 4096.389049199393
R2 Score: 0.7184981877253649
use kernel rbf // RMSE: 6619.176619384001 // R2 Score: 0.26500101184609803
use kernel poly // RMSE: 4376.989293802981 // R2 Score: 0.6786119157869899
use kernel sigmoid // RMSE: 6840.811691866695 // R2 Score: 0.21495586464202943
```

04 모델링: SVM

```python
import math
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score, mean_squared_error

# K-fold Cross Validation
kfold = KFold(n_splits=5, shuffle=True)
fold = 1

for train_index, test_index in kfold.split(X_train):
    X_train_fold, X_test_fold = X_train.iloc[train_index], X_train.iloc[test_index]
    y_train_fold, y_test_fold = y_train.iloc[train_index], y_train.iloc[test_index]

    # 모델 학습
    svm_model.fit(X_train_fold, y_train_fold)

    # 예측
    y_pred_fold = svm_model.predict(X_test_fold)

    # R-squared 값 계산
    r2 = r2_score(y_test_fold, y_pred_fold)
    print(f"Fold {fold}: R-squared = {r2}")

    # RMSE 값 계산
    rmse = math.sqrt(mean_squared_error(y_test_fold, y_pred_fold))
    print(f"Fold {fold}: RMSE = {rmse}")

    fold += 1
```

K-Fold Cross Validation 시행 → 모델의 일반화 검증

```
Fold 1: R-squared = 0.7076489016150782
Fold 1: RMSE = 4441.256004901415
Fold 2: R-squared = 0.7041296779612828
Fold 2: RMSE = 4422.055690521156
Fold 3: R-squared = 0.6999159522792382
Fold 3: RMSE = 4245.0784265135135
Fold 4: R-squared = 0.6883559676148416
Fold 4: RMSE = 4572.940242344417
Fold 5: R-squared = 0.6808091349687757
Fold 5: RMSE = 4605.185153578386
```

04 모델링: 라쏘 회귀

```
from sklearn.model_selection import train_test_split
train , test = train_test_split(df2005, test_size = 0.2)

x_train = train.drop(['8주뒤배출권가격(종가)'], axis=1)
y_train = train['8주뒤배출권가격(종가)']

x_test = test.drop(['8주뒤배출권가격(종가)'], axis = 1)
y_test = test['8주뒤배출권가격(종가)']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)

x_test_scaled = scaler.fit_transform(x_test)
x_test = pd.DataFrame(x_test_scaled)
```

```
Mean Squared Error: 17674445.578374073
RMSE: 4204.098664205453
R²  Score: 0.7083987969124752
```

Lasso 모델 생성

```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import numpy as np

# Lasso 회귀 모델 생성
lasso = Lasso(alpha=0.01)  # alpha는 정규화 강도를 조절하는 매개변수입니다.

# 모델 훈련
lasso.fit(x_train, y_train)

# 훈련된 모델을 사용하여 예측
y_pred = lasso.predict(x_test)

# 평균 제곱 오차 계산
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

print("RMSE:", np.sqrt(mse))

r2 = r2_score(y_test, y_pred)
print("R²  Score:", r2)
```

04 모델링: XGBoost

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import math

import pandas as pd
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV


df.dropna(inplace=True)
df.isnull().sum()

from sklearn.preprocessing import MinMaxScaler

# '배출권 가격(2달)'를 제외한 독립변수 선택
columns_to_normalize = df.drop('8주뒤배출권가격(종가)', axis=1).columns

# 정규화 수행
scaler = MinMaxScaler()
df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])


# 독립변수(X)와 종속변수(y)로 나누기
X = df.drop("8주뒤배출권가격(종가)", axis=1)
y = df["8주뒤배출권가격(종가)"]

# 훈련 데이터와 테스트 데이터로 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# XGBoost 모델 생성 및 학습
xgb_model = xgb.XGBRegressor()
xgb_model.fit(X_train, y_train)

param_grid = {
    'n_estimators': [100, 200, 300],  # 트리의 개수
    'max_depth': [3, 4, 5],  # 트리의 최대 깊이
    'learning_rate': [0.1, 0.01, 0.001]  # 학습률
}
```

XGBoost모델 생성 및 평가

```
# 그리드 서치 수행
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)


# 최적의 파라미터와 모델 출력
print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_

# 테스트 데이터로 예측
y_pred = xgb_model.predict(X_test)

# 모델 평가
y_pred = xgb_model.predict(X_test)
rmse = math.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE:", rmse)


# 모델 평가 (R2 스코어)
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 300}
RMSE: 1644.8469364401594
R2 Score: 0.9546131771256503
```

04 모델링: XGBoost

Best Parameters 로 모델 돌리기

```python
# XGBoost 모델 생성 및 학습
xgb_model = xgb.XGBRegressor(learning_rate= 0.1, max_depth= 5, n_estimators= 300)
xgb_model.fit(X_train, y_train)

# 테스트 데이터로 예측
y_pred = xgb_model.predict(X_test)

# 모델 평가
y_pred = xgb_model.predict(X_test)
rmse = math.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE:", rmse)


# 모델 평가 (R2 스코어)
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
RMSE: 1634.3646106008653
R2 Score: 0.9551898186223888
```

```
Fold 1: R-squared = 0.9806713908156182
Fold 1: RMSE = 1160.4957783510617
Fold 2: R-squared = 0.962744626704706
Fold 2: RMSE = 1527.202221977913
Fold 3: R-squared = 0.9678275074893193
Fold 3: RMSE = 1432.503819066544
Fold 4: R-squared = 0.9736498318463106
Fold 4: RMSE = 1272.8313106570936
Fold 5: R-squared = 0.9737888434313342
Fold 5: RMSE = 1350.777538706034
```

K-Fold Cross Validation 시행 → 모델의 **일반화** 검증

```python
import math
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score, mean_squared_error

# K-fold Cross Validation
kfold = KFold(n_splits=5, shuffle=True)
fold = 1

for train_index, test_index in kfold.split(X_train):
    X_train_fold, X_test_fold = X_train.iloc[train_index], X_train.iloc[test_index]
    y_train_fold, y_test_fold = y_train.iloc[train_index], y_train.iloc[test_index]

    # 모델 학습
    xgb_model.fit(X_train_fold, y_train_fold)

    # 예측
    y_pred_fold = xgb_model.predict(X_test_fold)

    # R-squared 값 계산
    r2 = r2_score(y_test_fold, y_pred_fold)
    print(f"Fold {fold}: R-squared = {r2}")

    # RMSE 값 계산
    rmse = math.sqrt(mean_squared_error(y_test_fold, y_pred_fold))
    print(f"Fold {fold}: RMSE = {rmse}")

    fold += 1
```

04 모델링: 랜덤 포레스트

Random forest 모델 구현 및 평가

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import numpy as np


# '배출권 가격(2달)'를 제외한 독립변수 선택
columns_to_normalize = df.drop('8주뒤배출권가격(종가)', axis=1).columns

# 정규화 수행
scaler = MinMaxScaler()
df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])


# 독립변수(X)와 종속변수(y)로 나누기
X = df.drop("8주뒤배출권가격(종가)", axis=1)
y = df["8주뒤배출권가격(종가)"]

# 훈련 데이터와 테스트 데이터로 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# 모델 생성
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# 모델 훈련
rf_model.fit(X_train, y_train)

# 예측 수행
y_pred = rf_model.predict(X_test)

# 모델 평가
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("RMSE:", np.sqrt(mse))
print("R2 Score:", r2)
```

```
Mean Squared Error: 2827855.217026939
RMSE: 1681.622792729374
R2 Score: 0.952560951191452
```

04 모델링: 랜덤 포레스트

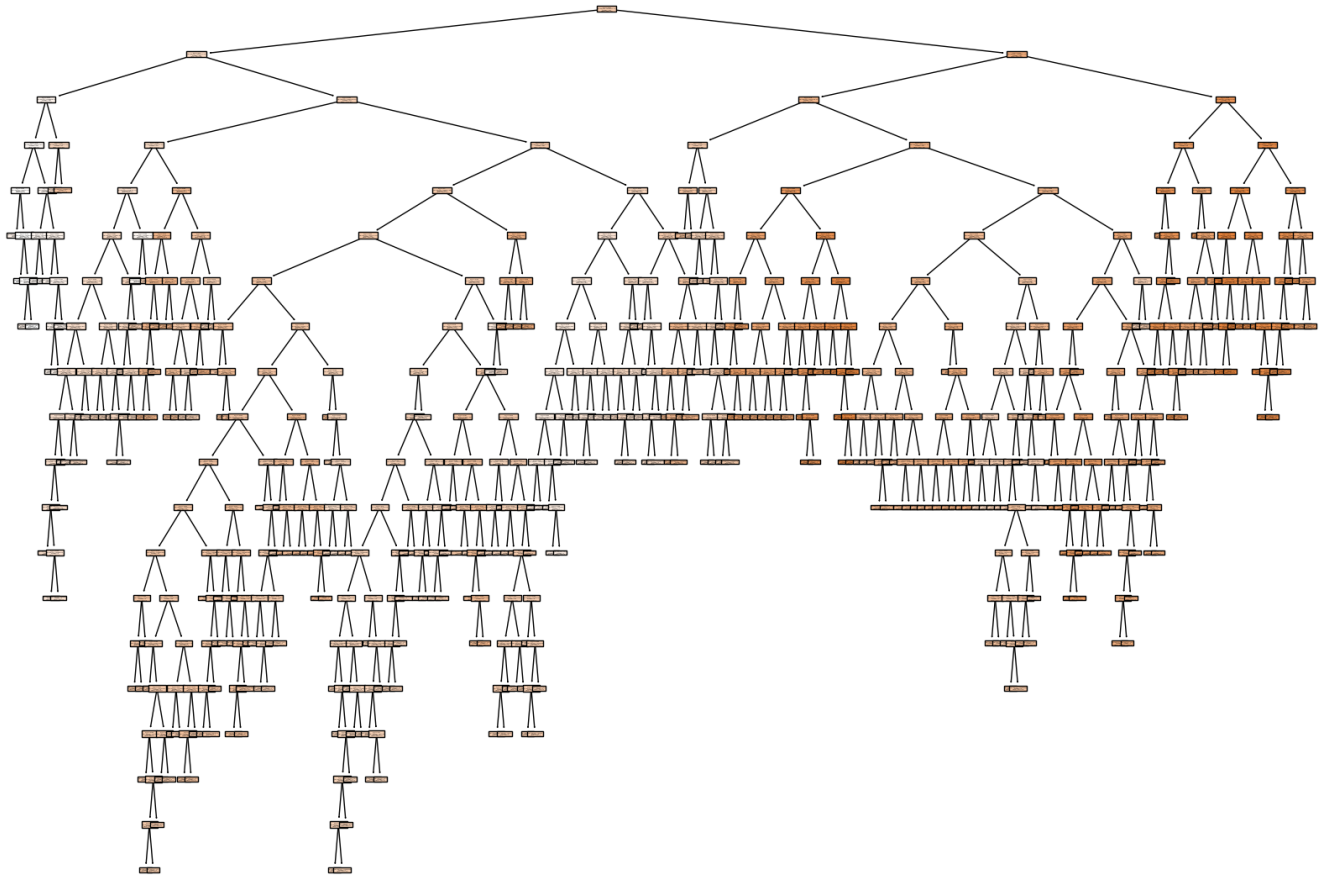
```python
import math
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score, mean_squared_error

# K-fold Cross Validation
kfold = KFold(n_splits=5, shuffle=True)
fold = 1

for train_index, test_index in kfold.split(X_train):
    X_train_fold, X_test_fold = X_train.iloc[train_index], X_train.iloc[test_index]
    y_train_fold, y_test_fold = y_train.iloc[train_index], y_train.iloc[test_index]

    # 모델 학습
    rf_model.fit(X_train_fold, y_train_fold)

    # 예측
    y_pred_fold = rf_model.predict(X_test_fold)

    # R-squared 값 계산
    r2 = r2_score(y_test_fold, y_pred_fold)
    print(f"Fold {fold}: R-squared = {r2}")

    # RMSE 값 계산
    rmse = math.sqrt(mean_squared_error(y_test_fold, y_pred_fold))
    print(f"Fold {fold}: RMSE = {rmse}")

    fold += 1
```

K-Fold Cross Validation 시행
→ 모델의 **일반화** 검증

```
Fold 1: R-squared = 0.9623826843338883
Fold 1: RMSE = 1489.0344854460395
Fold 2: R-squared = 0.9723164189165578
Fold 2: RMSE = 1360.09746074357
Fold 3: R-squared = 0.9742858448863609
Fold 3: RMSE = 1257.6667650354545
Fold 4: R-squared = 0.9765982421893781
Fold 4: RMSE = 1318.1705250403977
Fold 5: R-squared = 0.9775169240618844
Fold 5: RMSE = 1205.4414422413793
```

04 모델링: 랜덤 포레스트

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn import tree
import matplotlib.pyplot as plt

# 첫 번째 결정 트리 추출
tree_model = rf_model.estimators_[0]

# 결정 트리 시각화
plt.figure(figsize=(20, 14))
tree.plot_tree(tree_model, filled=True)
plt.show()
```



04 모델링: 다층퍼셉트론

ML 모델에서 더 나아가 **딥러닝 모델** 사용

여러 개의 **은닉층** 통해 **비선형 문제** 해결

```
R2 Score: 0.7024063069935089
```

```python
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler

# 독립변수와 종속변수 분리
X = df.drop("8주뒤배출권가격(종가)", axis=1)
y = df["8주뒤배출권가격(종가)"]

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 입력 변수 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# 탐색 대상 파라미터 범위 설정
# param_space = {
#     'hidden_layer_sizes': [(50, 50), (100, 100), (200, 200)],
#     'activation': ['relu', 'tanh'],
#     'solver': ['adam', 'lbfgs'],
#     'alpha': (1e-4, 1e-2),
#     'learning_rate': ['constant', 'adaptive']
# }
# MLP 모델 구성
mlp_model = MLPRegressor(hidden_layer_sizes=(100, 100), activation="relu", solver="lbfgs",
                         learning_rate="adaptive",random_state=42)

# 모델 학습
mlp_model.fit(X_train, y_train)

# 모델 예측
y_pred = mlp_model.predict(X_test)

# 모델 평가
r2score = r2_score(y_test, y_pred)
print("R2 Score:", r2score)
```

04 모델링: 다층퍼셉트론

```python
import math
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score, mean_squared_error

# K-fold Cross Validation
kfold = KFold(n_splits=5, shuffle=True)
fold = 1

for train_index, test_index in kfold.split(X_train):
    X_train_fold, X_test_fold = X_train.iloc[train_index], X_train.iloc[test_index]
    y_train_fold, y_test_fold = y_train.iloc[train_index], y_train.iloc[test_index]

    # 모델 학습
    mlp_model.fit(X_train_fold, y_train_fold)

    # 예측
    y_pred_fold = mlp_model.predict(X_test_fold)

    # R-squared 값 계산
    r2 = r2_score(y_test_fold, y_pred_fold)
    print(f"Fold {fold}: R-squared = {r2}")

    # RMSE 값 계산
    rmse = math.sqrt(mean_squared_error(y_test_fold, y_pred_fold))
    print(f"Fold {fold}: RMSE = {rmse}")

    fold += 1
```

K-Fold Cross Validation 시행
→ 모델의 **일반화** 검증

```
Fold 1: R-squared = 0.7014157723963176
Fold 1: RMSE = 4354.950144810078
Fold 2: R-squared = 0.7288211114724887
Fold 2: RMSE = 4221.049938912648
Fold 3: R-squared = 0.7694782749268786
Fold 3: RMSE = 3911.4007057796016
Fold 4: R-squared = 0.6490540327824907
Fold 4: RMSE = 4496.317321354828
Fold 5: R-squared = 0.7489478043525578
Fold 5: RMSE = 4266.898466994502
```

04 모델링: ARIMA 모델

탄소배출권 가격은 시간에 따라 타겟값이 달라지는 **시계열적 특성**을 가진 데이터

시계열적 특성을 반영하기 위해 **시계열 모델** 적용

6개월, 12개월의 **계절성**을 반영한다고 가정

```python
import math
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import cpi
from datetime import date
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller, kpss
```

```python
# KPSS 검정을 수행할 함수 정의
def kpss_test(x, h0_type='c'):
    '''
    Function for performing the Kwiatkowski-Phillips-Schmidt-Shin test for stationarity

    Null Hypothesis: time series is stationary
    Alternate Hypothesis: time series is not stationary

    Parameters
    ----------
    x: pd.Series / np.array
        The time series to be checked for stationarity
    h0_type: str{'c', 'ct'}
        Indicates the null hypothesis of the KPSS test:
            * 'c': The data is stationary around a constant(default)
            * 'ct': The data is stationary around a trend

    Returns
    -------
    results: pd.DataFrame
        A DataFrame with the KPSS test's results
    '''

    indices = ['Test Statistic', 'p-value', '# of Lags']

    kpss_test = kpss(x, regression=h0_type)
    results = pd.Series(kpss_test[0:3], index=indices)

    for key, value in kpss_test[3].items():
        results[f'Critical Value ({key})'] = value

    return results
```

04 모델링: ARIMA 모델

```python
# ADF 테스트를 수행할 함수 정의
def adf_test(x):
    '''
    Function for performing the Augmented Dickey-Fuller test for stationarity

    Null Hypothesis: time series is not stationary
    Alternate Hypothesis: time series is stationary

    Parameters
    ----------
    x : pd.Series / np.array
        The time series to be checked for stationarity

    Returns
    -------
    results: pd.DataFrame
        A DataFrame with the ADF test's results
    '''

    indices = ['Test Statistic', 'p-value',
               '# of Lags Used', '# of Observations Used']

    adf_test = adfuller(x, autolag='AIC')
    results = pd.Series(adf_test[0:4], index=indices)

    for key, value in adf_test[4].items():
        results[f'Critical Value ({key})'] = value

    return results
```

```python
def test_autocorrelation(x, n_lags=40, alpha=0.05, h0_type='c'):
    '''
    Function for testing the stationarity of a series by using:
    * the ADF test
    * the KPSS test
    * ACF/PACF plots
    Parameters
    ----------
    x: pd.Series / np.array
        The time series to be checked for stationarity
    n_lags : int
        The number of lags for the ACF/PACF plots
    alpha : float
        Significance level for the ACF/PACF plots
    h0_type: str{'c', 'ct'}
        Indicates the null hypothesis of the KPSS test:
            * 'c': The data is stationary around a constant(default)
            * 'ct': The data is stationary around a trend
    Returns
    -------
    fig : matplotlib.figure.Figure
        Figure containing the ACF/PACF plot
    '''

    adf_results = adf_test(x)
    kpss_results = kpss_test(x, h0_type=h0_type)

    print('ADF test statistic: {:.2f} (p-val: {:.2f})'.format(adf_results['Test Statistic'],
                                                              adf_results['p-value']))
    print('KPSS test statistic: {:.2f} (p-val: {:.2f})'.format(kpss_results['Test Statistic'],
                                                               kpss_results['p-value']))

    fig, ax = plt.subplots(2, figsize=(16, 8))
    plot_acf(x, ax=ax[0], lags=n_lags, alpha=alpha)
    plot_pacf(x, ax=ax[1], lags=n_lags, alpha=alpha)

    return fig
```

04 모델링: ARIMA 모델

```python
# 파일 경로
file_path = "/content/drive/MyDrive/🐹전종설🐹/코드/유진티비/Time Series Modeling/df_time_series.csv"

# 인코딩 방식 리스트
encodings = ['utf-8', 'euc-kr', 'latin1']

# 각 인코딩 방식으로 파일 열기 시도
df = None
for encoding in encodings:
    try:
        df = pd.read_csv(file_path, encoding=encoding)
        break
    except UnicodeDecodeError:
        continue

# 파일 열기 성공 여부 확인
if df is None:
    print("파일 열기에 실패했습니다.")
else:
    print("파일 열기 성공!")

df.head(5)
```

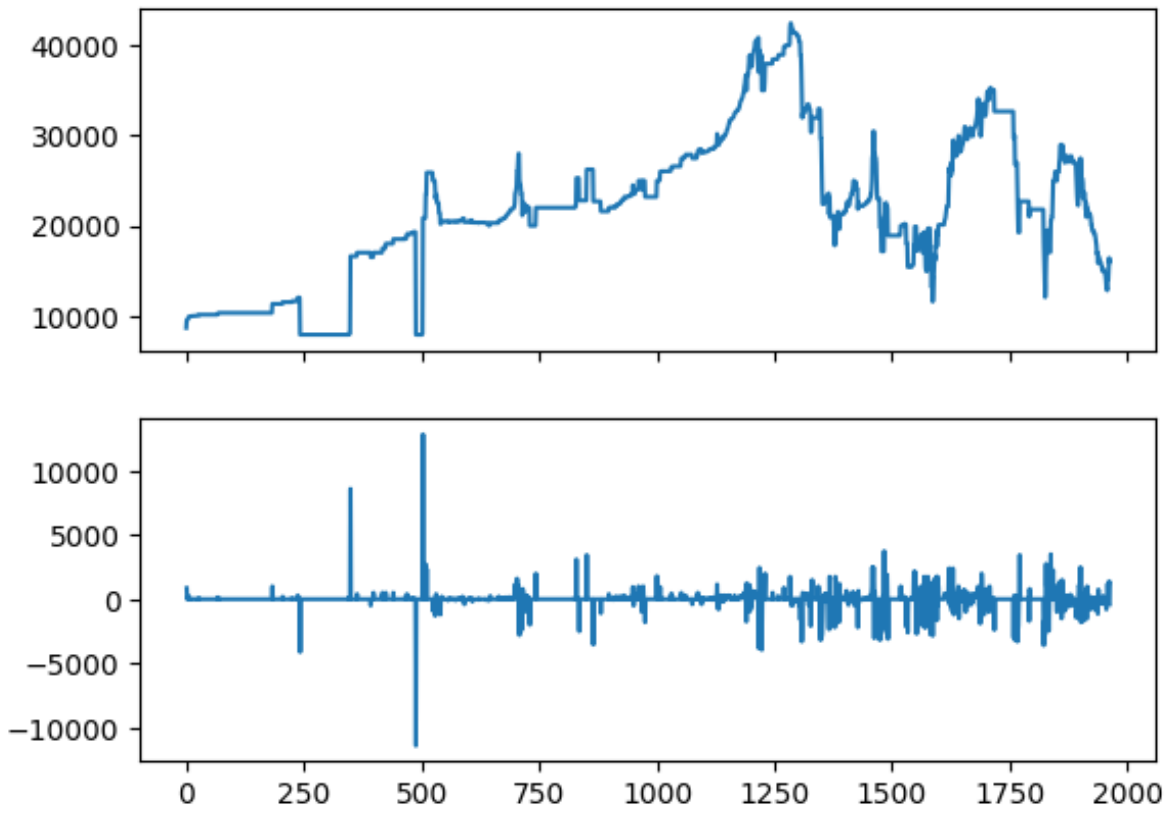
\# 설명 변수를 제외한 타겟 데이터 (탄소배출권 가격)만을 가지고 **시계열 모델링** 진행

파일 열기 성공!

	일자	배출권 가격(종가)
0	2015-01-12	8640
1	2015-01-13	9500
2	2015-01-14	9510
3	2015-01-15	9580
4	2015-01-16	9610

04 모델링: ARIMA 모델

```
# from pmdarima.model_selection import train_test_split

# train/validation 데이터 나누기
df_train = df[df.일자 < '2021-1-1']
df_test = df[df.일자 >= '2021-1-1']
df_train.set_index('일자', inplace=True)
df_test.set_index('일자', inplace=True)

diff = df['배출권 가격(종가)'].diff().dropna()

fig, ax = plt.subplots(2, sharex=True)
df['배출권 가격(종가)'].plot(ax=ax[0])
diff.plot(ax=ax[1])
```



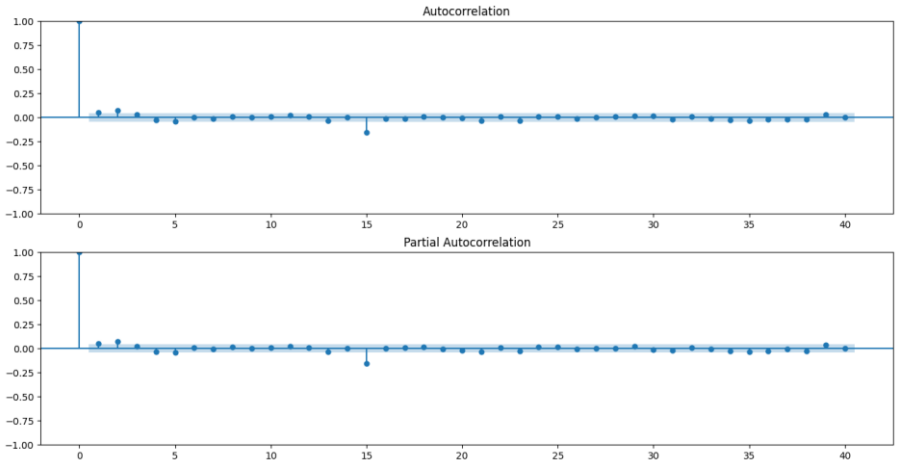
배출권 가격 데이터와 1차 차분한 시도표 시각화

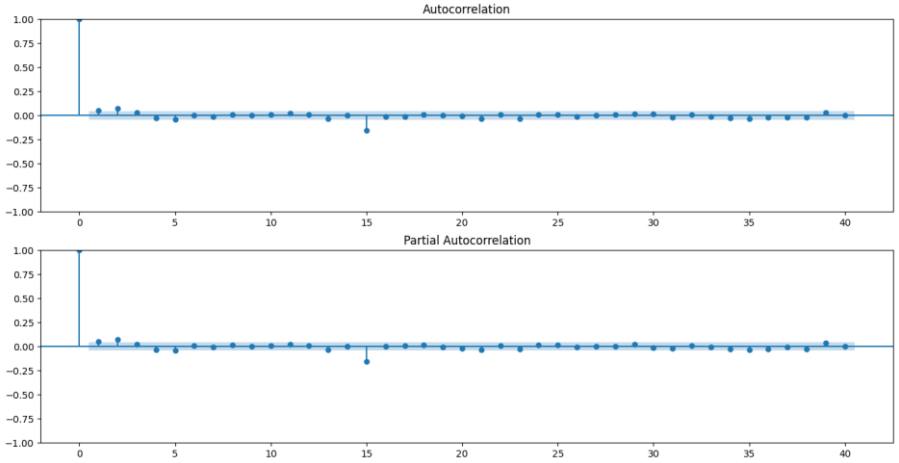
1차 차분한 시도표는 **추세**가 없어지고 수평한 형태

04 모델링: ARIMA 모델

```
# 정상성 검사
test_autocorrelation(diff)
```

```
ADF test statistic: -13.19 (p-val: 0.00)
KPSS test statistic: 0.12 (p-val: 0.10)
```





ADF 검정의 귀무가설 : 시계열이 **정상성**이 아니다.
KPSS 검정의 귀무가설 : 시계열이 **정상성**이다.

위 검정 결과에서 ADF의 **귀무가설 기각**
위 검정 결과에서 KPSS 검정 **귀무가설 채택**

즉, 차분된 주기는 **정상성**임을 나타낸다.

04 모델링: ARIMA 모델

```
# ARIMA 시계열 모델에 가장 적합한 모델을 자동으로 적합
model2 = pm.auto_arima(df_train,
                       error_action='ignore',
                       suppress_warnings=True,
                       seasonal=False, # 계절적 계절성 제외
                        stepwise=False,
                        approximation=False,
                        n_jobs=-1,
                        m=1)

model2.summary()
```

적합한 모델

ARIMA

ARIMA(2,1,2)(0,0,0)[0] intercept

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	1654
Model:	SARIMAX(2, 1, 2)	**Log Likelihood**	-13221.259
Date:	Tue, 30 May 2023	**AIC**	26454.519
Time:	13:04:04	**BIC**	26486.981
Sample:	0	**HQIC**	26466.553
	- 1654		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	11.5751	25.674	0.451	0.652	-38.744	61.895
ar.L1	0.6725	0.059	11.323	0.000	0.556	0.789
ar.L2	-0.8771	0.057	-15.274	0.000	-0.990	-0.765
ma.L1	-0.6619	0.053	-12.591	0.000	-0.765	-0.559
ma.L2	0.9147	0.048	19.028	0.000	0.821	1.009
sigma2	5.226e+05	3359.374	155.556	0.000	5.16e+05	5.29e+05

Ljung-Box (L1) (Q):	0.04	**Jarque-Bera (JB):**	844422.59
Prob(Q):	0.84	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.91	**Skew:**	1.11
Prob(H) (two-sided):	0.27	**Kurtosis:**	113.70

04 모델링: ARIMA 모델

```
model2 = pm.auto_arima(df_train,
                       error_action='ignore',
                       suppress_warnings=True,
                       seasonal=False, # 계절적 계절성 제외
                        stepwise=False,
                        approximation=False,
                        n_jobs=-1,
                        m=1)

model2.summary()
```

pmdarima 모듈을 이용한
최적의 arima 모델 search를 수행한 결과,

SARIMAX(2, 1, 2) 가 best model (AIC ⬇)

SARIMAX Results

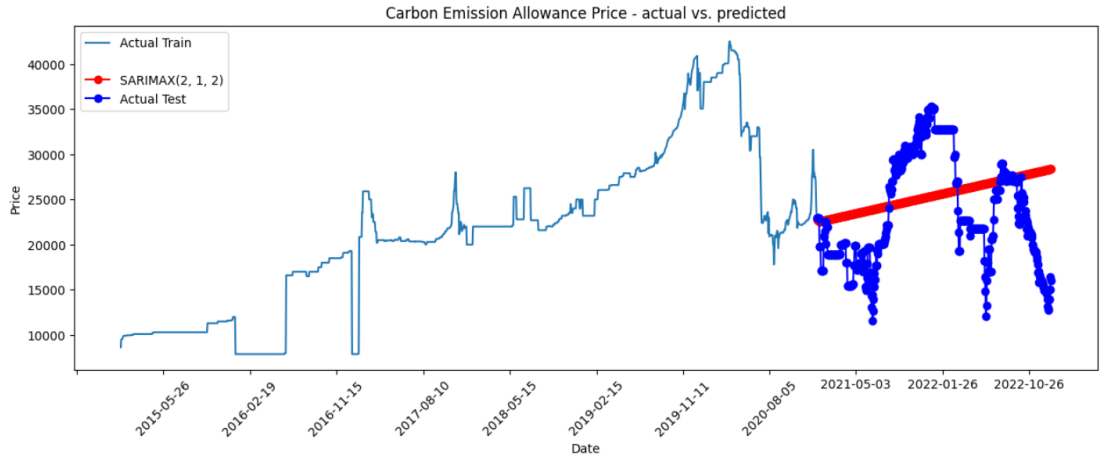
Dep. Variable:	y	**No. Observations:**	1654
Model:	SARIMAX(2, 1, 2)	**Log Likelihood**	-13221.259
Date:	Tue, 30 May 2023	**AIC**	26454.519
Time:	13:04:34	**BIC**	26486.981
Sample:	0	**HQIC**	26466.553
	- 1654		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	11.5751	25.674	0.451	0.652	-38.744	61.895
ar.L1	0.6725	0.059	11.323	0.000	0.556	0.789
ar.L2	-0.8771	0.057	-15.274	0.000	-0.990	-0.765
ma.L1	-0.6619	0.053	-12.591	0.000	-0.765	-0.559
ma.L2	0.9147	0.048	19.028	0.000	0.821	1.009
sigma2	5.226e+05	3359.374	155.556	0.000	5.16e+05	5.29e+05

Ljung-Box (L1) (Q):	0.04	**Jarque-Bera (JB):**	844422.59
Prob(Q):	0.84	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.91	**Skew:**	1.11
Prob(H) (two-sided):	0.27	**Kurtosis:**	113.70

04 모델링: ARIMA 모델

```python
import seaborn as sns
# 모델 예측
fcast = model2.predict(n_periods = len(df_test))
fcast.index = df_test.index
# 예측값 시각화
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

fig, ax = plt.subplots(figsize=(15,5))

ax = sns.lineplot(data=df_train, label='Actual Train')
ax.xaxis.set_major_locator(mdates.MonthLocator(interval=6))
plt.xticks(rotation=45)
ax.plot(fcast, color='red', marker='o', label='SARIMAX(2, 1, 2)')
ax.plot(df_test, color='blue', marker='o', label='Actual Test')
ax.set(title="Carbon Emission Allowance Price - actual vs. predicted",
            xlabel='Date',
            ylabel='Price')
ax.legend(loc='upper left')

plt.show()
```



기존 ml 모델에 비해 **설명변수**를 **제외**한
반응변수의 상태만을 가지고 모델링하였기 때문에
시계열적 특성을 **제외**한 다른 부분에서
예측 성능의 **한계**가 존재

```python
# 성능 평가 (R squared 값은 시계열 모델에서 잘 사용하지 x)

from sklearn.metrics import mean_squared_error
print('The RMSE SARIMAX(2, 1, 2) is :', mean_squared_error(df_test.values, fcast.values)**0.5)
```

```
The RMSE SARIMAX(2, 1, 2) is :
6298.482174655584
```

04 모델링: Prophet

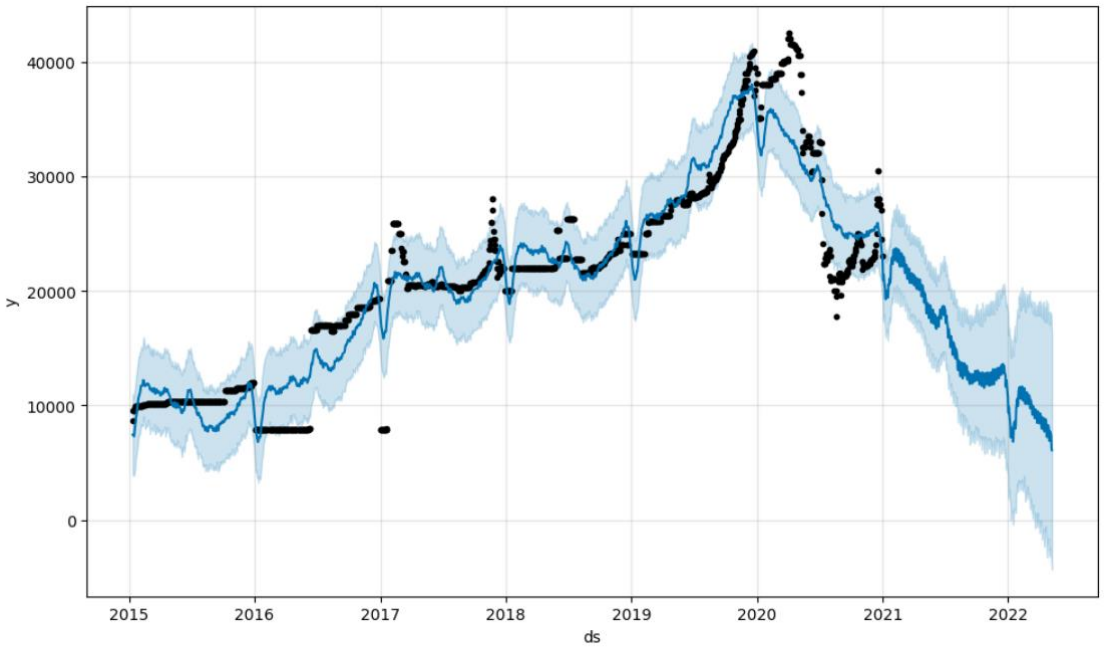
```python
# 라이브러리 불러오기
import pandas as pd
import seaborn as sns
from prophet import Prophet
import matplotlib.pyplot as plt
# 파일 경로
file_path = "/content/drive/MyDrive/🐹전종설🐹/코드/유진티비/Time Series Modeling/df_time_series.csv"

# 인코딩 방식 리스트
encodings = ['utf-8', 'euc-kr', 'latin1']

# 각 인코딩 방식으로 파일 열기 시도
df = None
for encoding in encodings:
    try:
        df = pd.read_csv(file_path, encoding=encoding)
        break
    except UnicodeDecodeError:
        continue

# 파일 열기 성공 여부 확인
if df is None:
    print("파일 열기에 실패했습니다.")
else:
    print("파일 열기 성공!")

df.head(5)
```

파일 열기 성공!

	일자	배출권 가격(종가)
0	2015-01-12	8640
1	2015-01-13	9500
2	2015-01-14	9510
3	2015-01-15	9580
4	2015-01-16	9610

04 모델링: Prophet

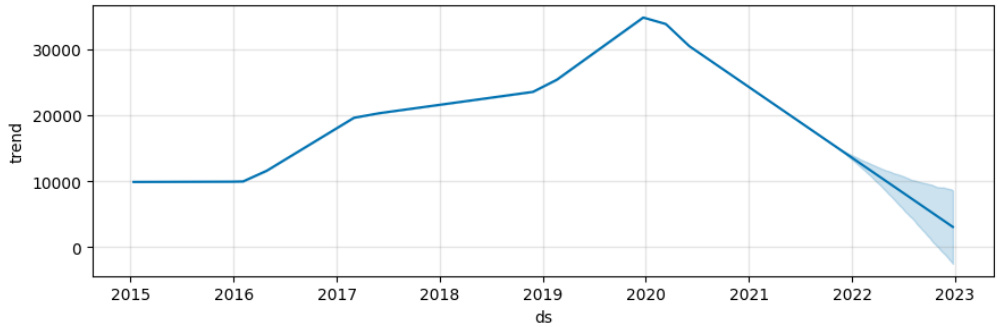
```python
# train/validation 데이터 나누기
df_train = df[df.일자 < '2021-1-1']
df_test = df[df.일자 >= '2021-1-1']

df_train = df_train
df_train.columns
df_train.rename(columns={'일자': 'ds', '배출권 가격(종가)': 'y'}, inplace=True)

df_test = df_test
df_test.columns
df_test.rename(columns={'일자': 'ds', '배출권 가격(종가)': 'y'}, inplace=True)

# 모델의 인스턴스를 만들고 데이터 적합
model_prophet = Prophet(seasonality_mode='additive')
model_prophet.add_country_holidays(country_name='KR')
model_prophet.add_seasonality(name='monthly', period=30.5, fourier_order=5)
model_prophet.fit(df_train)

# 가격을 예측하고 결과를 도식화
df_future = model_prophet.make_future_dataframe(periods=450)
df_pred = model_prophet.predict(df_future)
model_prophet.plot(df_pred)

plt.tight_layout()
plt.show()
```

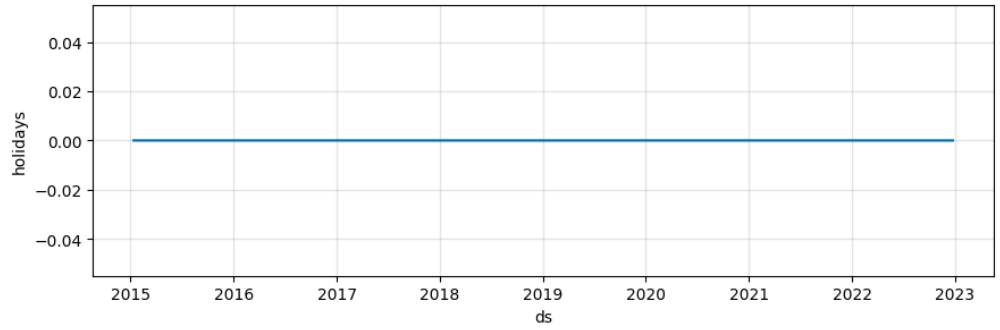


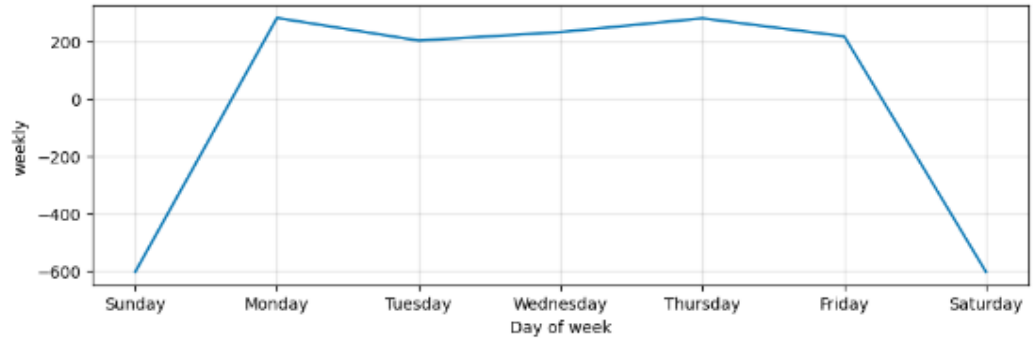
검은 점: 실제 관측값, 파란선: 적합화 (관측값과 정확히 일치x)
모델이 데이터의 노이즈를 **평활화**해서 **제거**했기 때문
Prophet은 **불확실성**을 **정량화** -> 적합선 주위의 파란색 간격

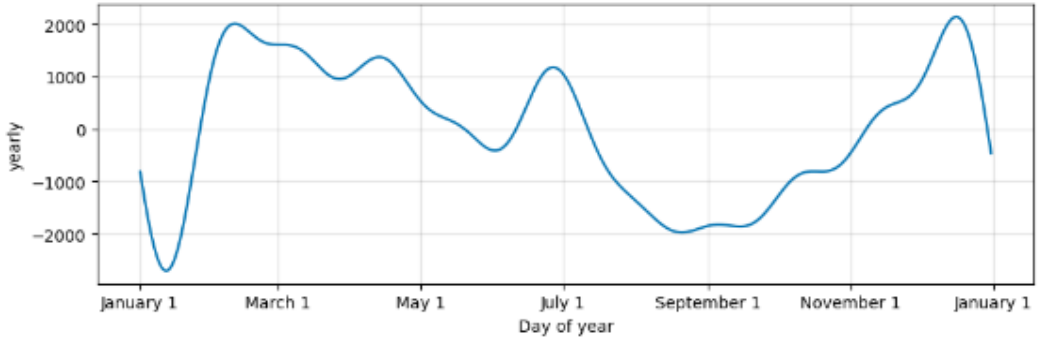
04 모델링: Prophet

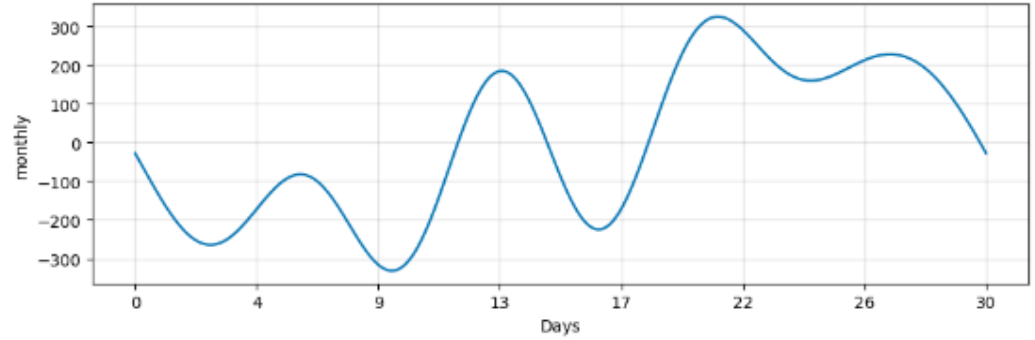
```
# 시계열 분해 검사
model_prophet.plot_components(df_pred)

plt.tight_layout()
plt.show()
```











04 모델링: Prophet

```python
# 성능 평가

from sklearn.metrics import mean_squared_error
print('The RMSE Prophet is :', mean_squared_error(df_test.y.values, df_pred[1469:]['yhat'].values)**0.5)
```

ds	y	yhat_lower	yhat_upper	yhat
2021-01-04	23000	17994.131683	24814.278956	21405.551423
2021-01-05	23000	17497.195630	24281.623999	20932.511567
2021-01-06	23000	17232.750471	24093.404784	20626.446583
2021-01-07	23000	17011.078526	23575.322369	20437.604573
2021-01-08	23000	16876.224300	23511.268489	20241.938895

```
The RMSE Prophet is
12235.5666556122
```

05 결론: 모델 성능 비교

종류	라쏘 회귀	SVM	XGBoost	**랜덤포레스트**	다층 퍼셉트론	ARIMA	Prophet
RMSE	4204.099	4096.389	1634.365	1681.623	4354.950	6298.482	12235.567
R2 score	0.708	0.718	0.955	0.963	0.701		

XGBoost와 **랜덤 포레스트**는 비슷한 **RMSE**를 보여주지만, **R2 score**값에서는 **랜덤 포레스트**가 약간 더 우수하다.

SVM은 기본적으로 **선형 분류**를 가정하는데 주어진 데이터는 **비선형**적인 패턴을 가지고 있어
성능이 좋지 않았을 것으로 생각된다.

ARIMA와 **Prophet**은 시계열 데이터의 시간적 특성을 고려하는 모델로서,
설명 변수와의 선형 관계를 고려하지 않으므로 R2 score 대신 RMSE를 사용하여 성능을 평가하였다.

최종 모델은 **랜덤포레스트** 모델로 선정하였다.

05 결론: 모델 성능 검증

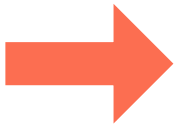
가장 성능이 좋다고 판단되는 Random Forest 모델을 이용하여 **탄소배출권 가격**을 예측

데이터의 **마지막 행** 확인

일자	종가	건설업	공공행정	선행지수순환변동치	석탄가격	사회적 거리두기 4단계	1차 계획기간	2차 계획기간
2022-11-03	0.373195	0.551813	0.833962	0.317073	0.751887	0.0	0.0	0.0

Random Forest가 데이터의 **마지막 행**으로 **예측한 가격**

```
X# X의 마지막 행을 x에 넣고 예측값을 얻기
x = X[-1:]
y_pred = rf_model.predict(x)

# 예측값 출력(2023년 1월 3일의 탄소배출권 가격)
print(y_pred)
```



```
[15857.]
```

05 결론: 모델 성능 검증

실제 2023-01-03의 **탄소배출권 가격** 확인

일자	종목명	종가
2023-01-03	KAU22	15,650

Random Forest가 데이터의 **마지막 행**으로 **예측한 가격**

[15857.]

(15857-15650) / 15857= 0.013054

즉, 학습시킨 **Random Foerst 모델**로
2023-01-03의 탄소배출권 가격을 예측하면
약 1.31%의 오차로 **예측 가능!**

05 결론: 한계점과 기대효과

한계점

연구에 사용된 **데이터셋**의 **양**이 **제한적**일 수 있다. 더 많은 데이터를 수집하고 활용한다면 모델의 성능과 예측 정확도를 향상시킬 수 있다.

연구에서 사용된 **독립 변수**들이 탄소배출권 가격에 영향을 미치는 모든 요인을 **포함**하지 않을 수 있다. 더 다양한 변수들을 고려하고 변수 선택 방법을 개선한다면 모델의 설명력을 높일 수 있다.

연구에서 적용된 **모델**은 **주관적인 선택**에 의해 결정된 것일 수 있다. 다른 모델이나 알고리즘을 추가로 검토하고 비교함으로써 모델 선택의 주관성을 줄일 수 있다.

기대효과

정확한 탄소배출권 가격 예측은 기업이 탄소배출을 줄이기 위한 **효율적인 자원 할당**을 할 수 있도록 도와준다. 이는 **기후 변화 대응** 및 **탄소 저감 목표 달성**에 기여할 수 있다.

연구 결과는 탄소배출권 가격에 영향을 미치는 요인을 이해하는 데 도움을 주며, 이를 토대로 **정부**와 **기관**들이 탄소배출권 시장에 대한 **정책**을 **개발**하고 **개선**하는 데 활용될 수 있다.

예측 가능한 가격 정보를 통해 기업은 탄소배출에 따른 비용을 예측하고 관리할 수 있으며, 이는 **기업**의 **경쟁력 강화**와 **비용 절감**에 도움이 될 수 있다.

본 예측 모델은 두 달 뒤의 탄소배출권 가격을 예측함으로써 기업의 **의사결정과정을 지원**하고, 향후 **예산 계획을 수립**할 때 신뢰할 수 있는 도구로 활용될 수 있다.

05 결론: 참고문헌

김수이, 『배출권거래 가격결정요인 분석과 전망』, 에너지경제연구원, 2007

김영민 외 1명, 『탄소배출권 거래시장 특성을 반영한 배출권가격 예측모델 개발』, 엘지씨엔에스, 2016

심성희, 『배출권거래시장 연계의 최신 논의 동향과 시사』, 에너지경제연구원, 2018

정수관, 『에너지가격이 탄소배출권가격에 미치는 영향에 대한 분석』, 산업연구원, 2018

박순철 외 1명, 『한국 탄소시장에서의 배출권 가격 결정요인 분석』, 한국환경경제학, 2018

임기성 외 4명, 『빅데이터 분석기법을 활용한 탄소배출권 가격 예측』, 한국건축시공학회, 2019

신동현, 『탄소효율성과 기업성과 간 관계 분석 : 국내 기업의 사례 분석』, 에너지경제연구원, 2019

우주희 외 1명, 『국내 탄소배출권거래시장 수요측면의 가격결정요인분석』, 한국자료분석학회, 2022

05 결론: 기여도

이름	이유진	윤우석	정영훈	최수정
기여도	100%	100%	100%	100%

Thank you

20192246 이유진
20192237 윤우석
20192263 정영훈
20192270 최수정