항공화물 수요 예측
프로젝트 보고서

2022. 10. 6 - 2022. 10. 31

데이터분석 수련생
박찬성 이재은

목 차

1. 서론

1.1 분석 개요

1.1.1 분석 배경

20세기에 이르러 항공산업의 발전에 따른 항공화물 수요가 증가하고 있으며, 그와 비례하게 항공화물 수요와 관련된 여러 연구도 함께 진행되고 있다. 그러나 지금까지 수행된 연구는 주로 화물이 아닌 여객 중심의 연구였으며, 화물과 관련된 연구 역시 포괄적이고 장기적 관점에서의 연구가 대부분이었다. 또한, 항공화물은 한국공항공사의 주요한 수입원 중 하나로 이에 대한 수요를 정확히 예측하는 것은 현실적으로 매우 필요한 일이다.

1.1.2 분석 목표

이에 본 프로젝트는 김포공항의 일일 화물량을 예측하되, 월 단위로 예측값을 얻은 후 최종 향후 1년의 화물예측값을 도출하는 것을 목표로 설정하였다. 그중 최종 모델인 일일 화물량 예측 모델의 경우, [1]논문을 참고하여 구현하였다.

1.2 분석 프로세스

[그림1] 프로젝트 분석 프로세스

본 프로젝트의 분석 프로세스는 상단의 그림 1처럼, 데이터 수집, 데이터 전처리, 분석 모델링, 결과 도출 순으로 진행되었다. 즉, 필요한 원천 데이터를 수집하고 전처리 과정을 진행한 후, 해당 데이터를 바탕으로 모델링 작업을 수행해 최종 예측값을 얻었다. 이후 분석 결과를 시각화 및 도식화하였다.

[그림2] 분석 모델링 세부 단계

이때 모델링의 경우 총 3단계로 구성하였는데, 첫 번째로 주 단위 화물량을 기반으로 Auto ARIMA를 수행한 뒤 1차 주 단위 예측값을 도출하였다(1단계). 이후 ARIMA

1) https://www.jkst.or.kr/articles/pdf/7jeG/kst-2020-038-03-3.pdf

예측값과 화물, 휴일 정보 등을 활용하여 Auto ML을 수행함으로써 최종 주 단위 예측값을 얻었다(2단계). 그리고 주 단위 예측값과 다른 변수들을 토대로 Auto ML을 다시 한번 수행하여 최종 일 단위 예측값 30일 치를 도출하였다(3단계). 마지막 세 번째 단계의 경우 기간을 바꿔가며 12번 반복 수행함으로써 총 1년 치 화물량을 예측하도록 하였다.

2. 분석 과정

2.1 데이터 수집

수집한 데이터는 총 두 가지로, 항공화물 데이터와 공휴일 정보 데이터이다. 두 개의 데이터를 결합하고 전처리 과정을 거쳐 학습 데이터를 완성하였다.

2.1.1 항공화물 실적 데이터

한국공항공사의 내부 데이터로, 2000년 1월부터 2022년 9월까지 김포공항 내의 일일 항공화물 실적 데이터이다. 해당 데이터는 '운항 일자', '운항 횟수', '유임승객 수', '화물', '수하물' 등 항공운항과 관련된 내용을 담고 있다. 그중 필요한 열인 '일자', '운항 횟수_도착', '운항 횟수_출발', '화물_도착', '화물_출발'만 추출하였고, 해당 데이터를 바탕으로 화물의 총집계량인 '총화물량'과, 해당 일자의 노선 개수를 의미하는 '노선 수'라는 파생 변수를 함께 생성하였다.

2.1.2 공휴일 데이터

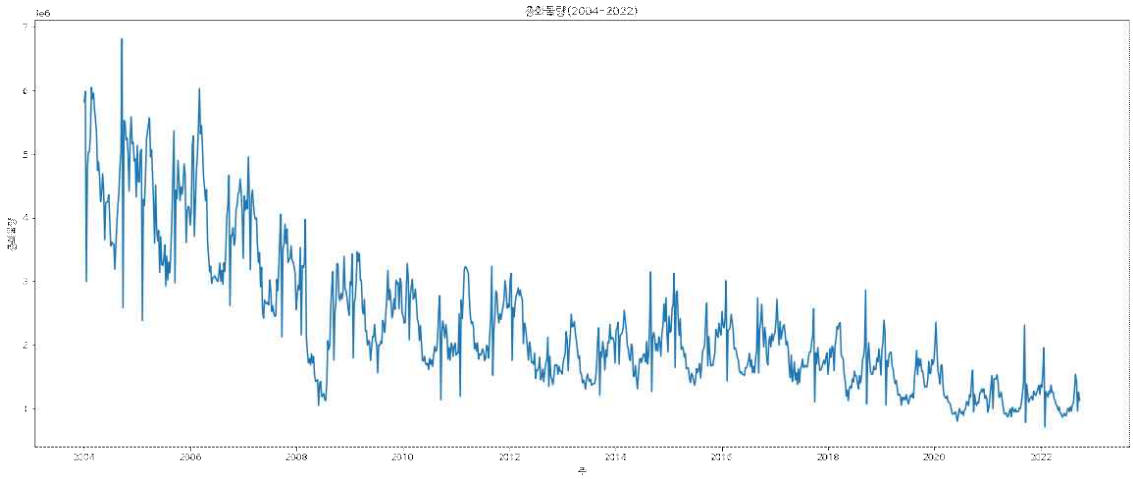
'휴일'을 화물량과 연관이 있는 요인으로 간주하였기에, 이와 관련된 데이터를 공공데이터포털에서 제공되는 [2]한국천문연구원 API를 통해 수집하였다. 해당 API는 날짜별 특일 및 휴일 여부와 명칭을 제공하고 있어, 휴일에 해당하는 일자와 명칭만을 저장하였다. 이때 항공화물 데이터와 동일 기간의 데이터를 얻고자 하였으나 공휴일 정보의 경우 2004년 1월부터 제공되고 있었기에 해당 기간만을 수집하였고, 항공화물 데이터 역시 이와 같은 기간으로 데이터를 수정하였다.

2.2 데이터 전처리

2.2.1 데이터 EDA

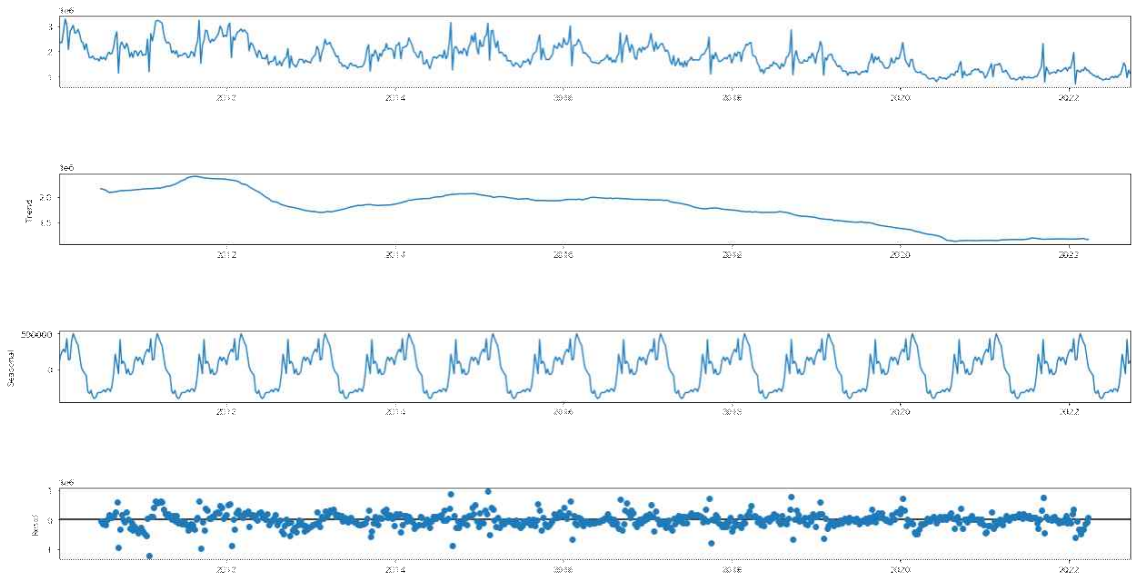
본격적인 데이터분석에 앞서 EDA와 전처리를 진행하였다. 다음 그림은 수집한 데이터 중 '총화물량'의 전체 기간 값을 그린 그래프이다.

2) https://www.data.go.kr/data/15012690/openapi.do



[그림3] 총화물량 그래프 (2004.01 - 2022.09)

육안으로 살펴보았을 때 2008년을 기점으로 전후 기간의 총화물량 차이가 있는 것으로 확인되었기에, 1) 전체 기간(2004.01 ~ 2022.09), 2) 2008년부터 끝까지(2008.01 ~ 2022.09), 3) 2010년부터 끝까지(2010.01 ~ 2022.09) 이렇게 총 세 가지 경우로 나누어 분석을 진행해보기로 하였다. 다음 그림은 세 번째 경우인 2010년 1월부터의 데이터의 시계열 분해 과정 속 그래프이다.



[그림4] 시계열 분해 (2010.01 - 2022.09)

2.2.2 변수 전처리

여러 변수와 관련된 전처리를 수행하였고, 그 과정에서 모델링에 필요한 여러 파생변수 또한 생성하는 작업을 거쳐 학습 데이터를 완성하였다.

◆ 공휴일 이름 통일

공휴일 데이터를 살펴보았을 때, 똑같은 휴일임에도 명칭이 다른 경우가 존재하였다. 다음 그림에서 볼 수 있듯이, 같은 날을 의미하는 '신정'과 '1월 1일' 또는 '석가탄신일'과 '부처님오신날'이 있고, '어린이 날'은 띄어쓰기 하나로 '어린이날'에 포함되지 않기도 하였다. 이러한 날들을 확인한 후, 동일한 명칭으로 수정하였다.

[그림5] 공휴일 명칭 수정 전후 비교

◆ 일자별 요일 추출

어떤 요일인지, 주말인지 아닌지, 그리고 주말에 속한 휴일인지 아니면 주말이 아닌 휴일인지 등을 확인하고자 일자별로 요일을 추출하였다. 월요일부터 일요일까지의 여부를 뽑은 후, '주말', '주말제외휴일', '주말동시휴일' 변수를 생성하였다.

	일자	월요일	화요일	수요일	목요일	금요일	토요일	일요일	주말	주말제외휴일	주말동시휴일
0	20100101	0	0	0	0	1	0	0	0	1	0
1	20100102	0	0	0	0	0	1	0	1	0	0
2	20100103	0	0	0	0	0	0	1	1	0	0
3	20100104	1	0	0	0	0	0	0	0	0	0
4	20100105	0	1	0	0	0	0	0	0	0	0

[그림6] 요일 관련 파생 변수 생성

◆ 일 단위 → 주 단위 변환

첫 번째 모델링의 경우 주 단위 예측값을 도출해야 하므로, 데이터의 일자도 일 단

위에서 주 단위로 변환 과정을 거쳤다. 우선 '일자'의 데이터형을 Object에서 DateTime으로 바꾼 후 인덱스로 설정하였고, DateTimeIndex의 [3]리샘플링을 통해 주 단위 변환을 수행하였다.

일자	총화물량	운항횟수	노선수
2010-01-03	347016.0	298.0	20
2010-01-04	123718.0	78.0	21
2010-01-05	401365.0	278.0	18
2010-01-06	481274.0	280.0	19
2010-01-07	414219.0	224.0	19
...	...	...	...

일자	총화물량	운항횟수	노선수
2010-01-03	2453515.0	1748.0	133
2010-01-10	2338951.0	1822.0	132
2010-01-17	2350311.0	1830.0	133
2010-01-24	2820431.0	1996.0	127
2010-01-31	3280586.0	1969.0	128
...	...	...	...

[그림7] 요일 관련 파생 변수 생성

◆ 공휴일 관련 변수

공휴일별로 해당하는 날인지 아닌지, 그리고 1주 전후인지 아닌지를 확인하고자 관련 변수를 생성하였다. 다음 그림은 '기독탄신일 1주후'에 해당되는 행을 추출한 일부이다. 이처럼 특정 휴일의 당일 혹은 1주 전후에 속한 경우 그 값을 1, 속하지 않는 경우 0으로 라벨링 처리하였다.

	일자	총화물량	운항횟수	노선수	공휴일여부	주말제외휴일	주말동시휴일	개천절	광복절	국회의원선거	...	주말동시휴일1주전	주말동시휴일1주후	개천절1주전	개천절1주후	광복절1주전	광복절1주후	국회의원선거1주전	국회의원선거1주후	기독탄신일1주전	기독탄신일1주후
51	2010-12-26	2023616.0	1890.0	135	1.0	0	1	0	0	0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
104	2012-01-01	2590271.0	1964.0	142	1.0	0	1	0	0	0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
156	2012-12-30	1538186.0	2020.0	146	1.0	1	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
208	2013-12-29	1712237.0	1999.0	150	1.0	1	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
260	2014-12-28	1885667.0	2273.0	124	1.0	1	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

[그림8] 공휴일 변수 생성 후 '기독탄신일 1주후' 행 추출

◆ 주 단위 → 일 단위 변환

주 단위 예측값을 얻은 후 최종 일 단위 예측을 위해 데이터의 단위를 다시 일자로 변환해주고자 하였다. 해당 과정에서는 한 가지 유의할 부분이 있는데, 하단의 그림과 같이 주 단위 데이터의 마지막 행의 경우 일 단위로 전환되면 해당주의 날짜가 모두 자동으로 생성되지 않는다는 점이다. 그래서 주 단위에서는 9월 25일이 9월의 마지막 날인 9월 30일까지를 포함하는 행이었지만, 일 단위로 전환되면 9월 25일 당일만 포

3) Python의 데이터프레임에서 DateTimeIndex를 원하는 주기로 나눠주는 메소드

함하게 된다. 그러므로 주 단위 데이터의 마지막 행의 날짜가 해당 월의 마지막 날인지를 보고 누락된 일자만큼 보충하였다. 이렇게 완성된 일자 데이터를 바탕으로 Auto ML을 수행하여 최종 주 단위 예측값을 도출하였다.

일자	주단위예측값
2010-01-03	1.816007e+06
2010-01-10	2.078754e+06
2010-01-17	2.315227e+06
2010-01-24	2.480183e+06
2010-01-31	3.152892e+06
...	...
2022-08-28	1.673086e+06
2022-09-04	1.712906e+06
2022-09-11	1.216558e+06
2022-09-18	1.404903e+06
2022-09-25	1.356833e+06

일자	주단위예측값
2010-01-03	1.816007e+06
2010-01-04	1.816007e+06
2010-01-05	1.816007e+06
2010-01-06	1.816007e+06
2010-01-07	1.816007e+06
...	...
2022-09-21	1.404903e+06
2022-09-22	1.404903e+06
2022-09-23	1.404903e+06
2022-09-24	1.404903e+06
2022-09-25	1.356833e+06

[그림9] 주 단위 → 일 단위 변환

일자	주단위예측값
2010-01-03	1.816007e+06
2010-01-04	1.816007e+06
2010-01-05	1.816007e+06
2010-01-06	1.816007e+06
2010-01-07	1.816007e+06
...	...
2022-09-21	1.404903e+06
2022-09-22	1.404903e+06
2022-09-23	1.404903e+06
2022-09-24	1.404903e+06
2022-09-25	1.356833e+06

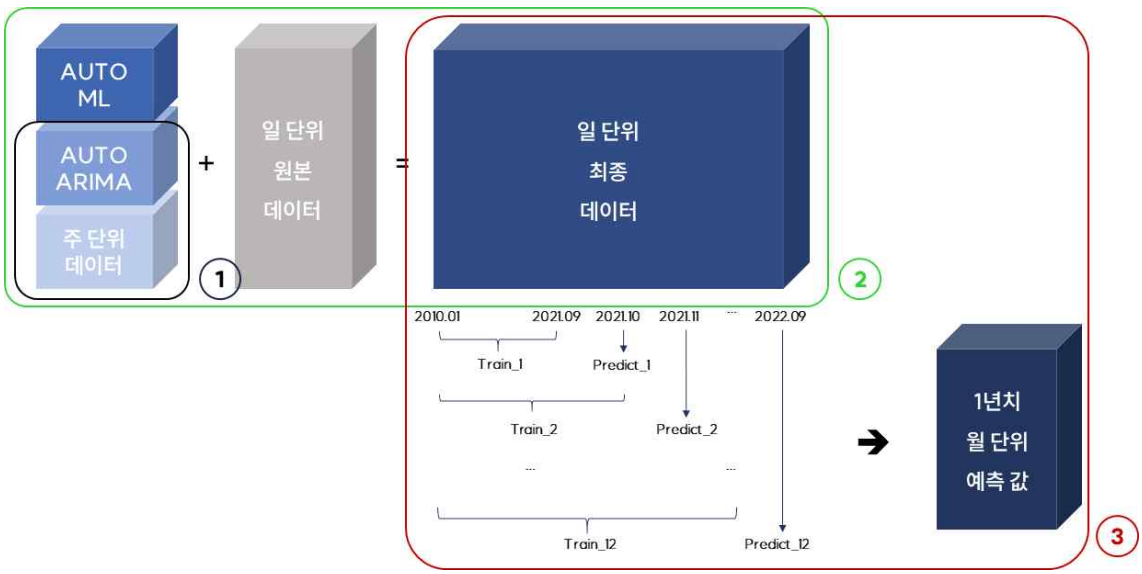
일자	주단위예측값
2010-01-03	1.816007e+06
2010-01-04	1.816007e+06
2010-01-05	1.816007e+06
2010-01-06	1.816007e+06
2010-01-07	1.816007e+06
...	...
2022-09-26	1.356833e+06
2022-09-27	1.356833e+06
2022-09-28	1.356833e+06
2022-09-29	1.356833e+06
2022-09-30	1.356833e+06

[그림10] 누락된 일자 데이터 보충

2.3 분석 모델링

앞서 1.2에서 언급한 바와 같이, 분석 모델링은 총 세 단계로 진행하였다. 주 단위 Auto ARIMA와 Auto ML의 경우, 전체 기간의 데이터를 전처리하면 모델의 반복 학습과 상관없이 필요한 데이터를 모두 얻을 수 있기에 한 번에 학습과 예측을 진행하였다. 학습 데이터는 초반에 계획하였던 대로 우선 2010년 1월부터 2022년 8월까지, 예측 데이터는 2021년 10월부터 22년 9월까지로 설정하여 모델링을 수행하였다(2.2.1에서 언급한 3번째 경우).

그리고 마지막 일 단위 Auto ML의 경우에는 해당 모델 하나가 일 단위로 향후 30일을 예측하도록 설계하였으므로, 총 1년 치 데이터를 예측하기 위해 모델 12개를 사용하였다. 따라서 2021년 10월부터 2022년 9월까지 하나의 모델당 한 달씩 예측하도록 하고, 학습 데이터는 2010년 1월부터 예측 데이터 범위의 한 달 전까지로 설정하여 수행하였다. 그렇게 도출된 12달, 즉 1년 치('21.10 ~ '22.9)의 예측값을 최종 예측값으로 설정하였으며, 전체적인 프로세스는 다음과 같다.



[그림11] 3단계 모델링 프로세스

2.3.1 주 단위 화물량 Auto ARIMA 예측

우선 일일 총화물량 데이터를 기반으로, 주 단위로 데이터를 집계한 후 [4]Auto ARIMA를 수행하였다. 그 결과 추정된 최적의 모형은 ARIMA(0, 1, 1) 모델이었고, 해당 모델로 예측한 주 단위 예측 화물량을 일 단위 Auto ML의 독립변수 중 하나로 활용하였다.

4) 힌드만-칸다카르 알고리즘을 사용하여 ARIMA 모델의 p(AR차수), d(차분수), q(MA차수)를 바꿔가며 계수를 추정하고 데이터에 최적의 결과값을 내는 모형을 추정해주는 모델링 방법

2.3.2 주 단위 화물량 Auto ML 예측

2.3.1에서 얻은 주 단위 예측 화물량과 앞서 만들었던 파생 변수들을 사용하여 [5] Pycaret의 [6]Auto ML 모델링을 수행하였다. 이로써 최종 주 단위 예측값을 도출하였다. 해당 값을 하나의 요인으로 설정하고, 기존에 완성하였던 원본 일 단위 데이터와 결합하여 최종 일 단위 데이터를 생성하였다. 해당 데이터를 바탕으로 마지막 일 단위 Auto ML을 수행하였다.

2.3.3 일 단위 화물량 Auto ML 예측

2.3.2에서 완성된 최종 일 단위 데이터를 학습 데이터로 사용하여, 예측 기간으로 잡았던 2021년 10월부터 2022년 9월까지의 일일 화물량을 Auto ML로 예측하였다. 첫 번째 모델은 2021년 9월까지의 데이터를 학습하고 2021년 10월을 예측하였고, 두 번째 모델은 2021년 10월까지를 학습하고 2021년 11월을 예측하는 식으로, 총 12번 진행하여 최종 1년 치 예측값을 얻었다.

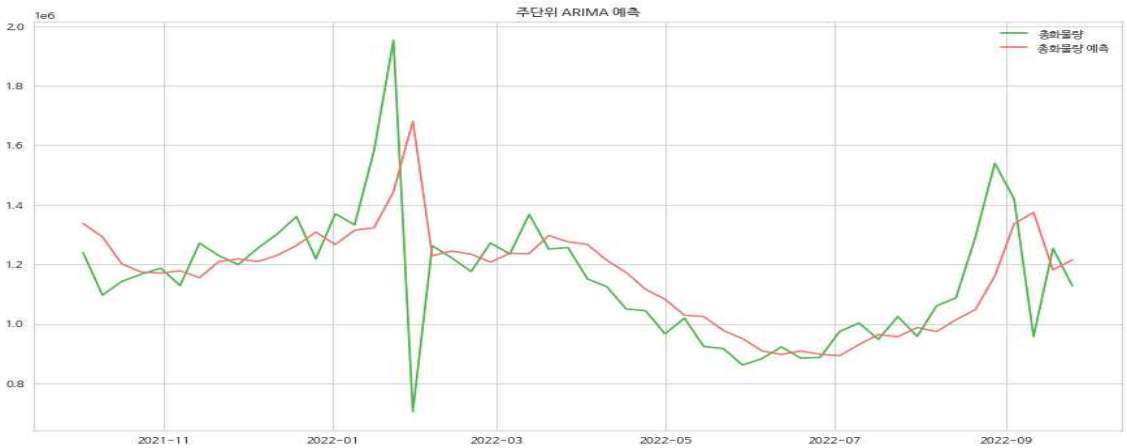
3. 결론

3.1 분석 결과

마지막으로 분석 프로세스의 모델들의 성능을 확인하기 위해 실제 화물량 값과 예측값 간의 차이를 시각화하여 정리하였다.

3.1.1 주 단위 Auto ARIMA

주 단위 Auto ARIMA가 선정한 모델은 ARIMA(0,1,1)이며, 예측 데이터의 주 단위 RMSE는 220829, 전체 기간은 288634이다.



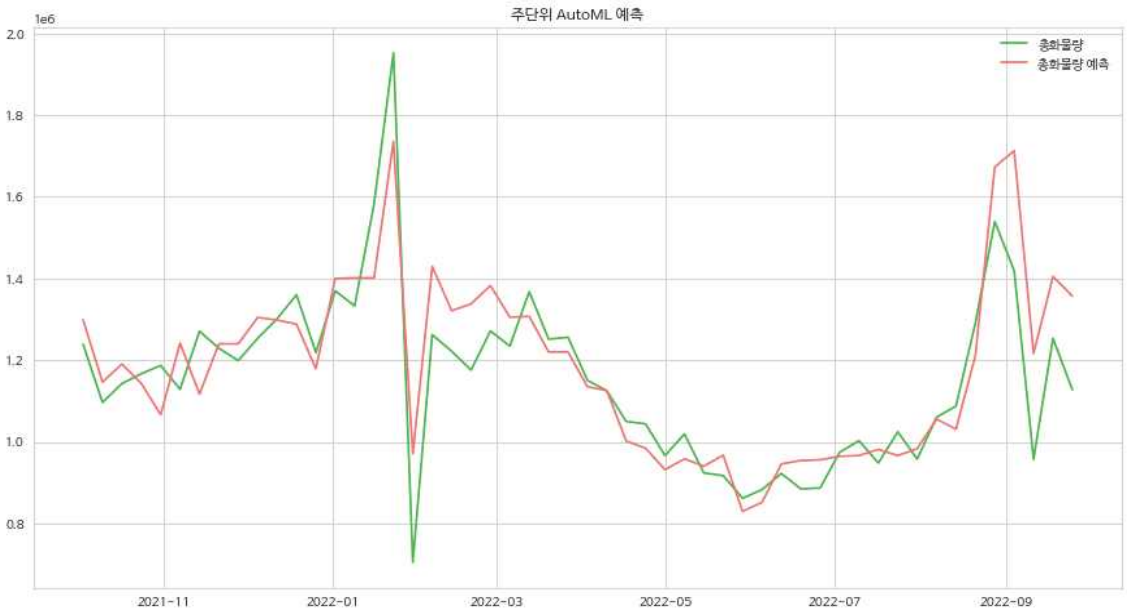
[그림12] 주 단위 Auto ARIMA 예측 그래프 (2021.10 - 2022.09)

5) 여러 모델을 동일한 환경에서 한 줄의 코드로 실행할 수 있도록 만든 자동화 Python 라이브러리, 여러 모델을 한눈에 비교 가능하며 가장 좋은 모델을 뽑아낼 수 있음

6) 시간 소모적이고 반복적인 기계 학습 모델 개발 작업을 자동화하는 프로세스로, 학습 중에 다양한 알고리즘과 매개 변수를 시도하는 많은 파이프라인을 동시에 만듦으로써 최적의 모델을 도출

3.1.2 주 단위 Auto ML

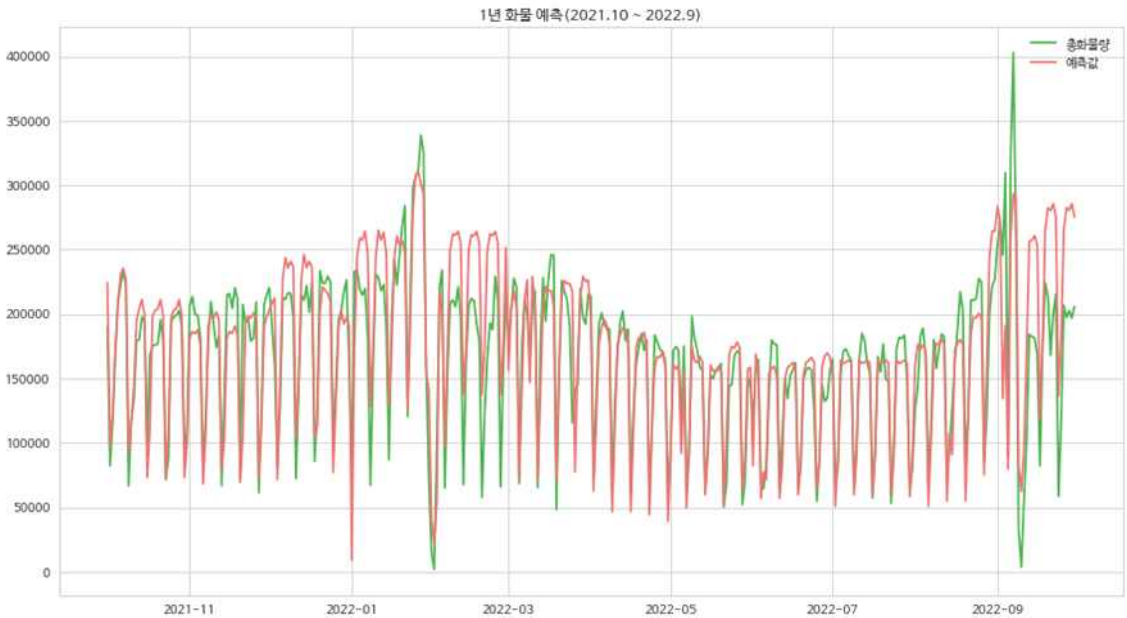
주 단위 Auto ML이 선정한 모델은 CatBoost Regressor이며, 예측 데이터의 주 단위 RMSE는 239294, 전체 기간은 127990이다.



[그림13] 주 단위 Auto ML 예측 그래프 (2021.10 - 2022.09)

3.1.3 최종 모델(일 단위 Auto ML)

최종 예측 모델인 일 단위 Auto ML도 역시 CatBoost Regressor를 선정하여 활용하였으며, 예측 데이터의 일 단위 RMSE는 28217, 전체 기간은 38180이다.

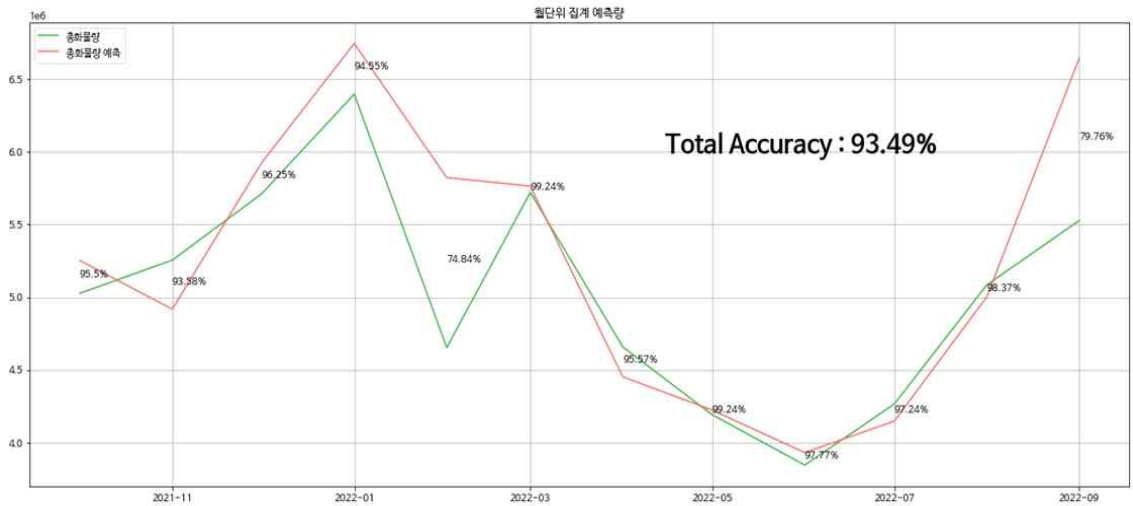


[그림14] 일 단위 Auto ML 1년 치 예측 그래프 (2021.10 - 2022.09)

그리고 2.2.1에서 계획했던 대로, 학습 데이터의 기간을 달리하여 모델링 해본 결과, 1) 전체 기간(2004.01 ~ 2022.09)의 경우, 테스트 데이터와 전체 데이터 모두 예측 성능이 최종 모델(3) 2010.01 ~ 2022.09)보다 낮았고, 2) 2008년부터 끝까지(2008.01 ~ 2022.09)로 설정한 경우 테스트 데이터 예측 성능은 비슷하였으나 전체 데이터 예측도가 현저히 낮음을 확인하였다. 이는 2010년을 기점으로 화물 수요량의 흐름이 크게 변화하여, 2010년 이전의 데이터가 그 이후를 예측하는 데에 실질적인 도움이 되지 않기 때문인 것으로 추정된다. 그러므로 최종 데이터 기간은 처음 계획한 그대로 2010년 1월부터 2022년 9월까지로 잡고 모델링 과정을 완성하였다.

3.1.4 모델 성능

다음은 각각 완성된 최종 모델의 정확도를 시각화한 그림과, 최종 모델과 단일 ARIMA의 예측 정확도 성능을 비교한 표이다(표에서 Month는 2021년 10월을 의미). 모델의 정확도 평균은 약 93.5였고, 명절이 끼어있는 2월과 9월을 제외하면 단일 ARIMA보다 평균적으로 더 높은 정확도를 유지하였다.



[그림15] 최종 예측 모델 1년 치 예측 정확도 (2021.10 – 2022.09)

	3-Part Model	ARIMA
	Accuracy(%)	
Month*	95.50	90.71
Month + 1	93.58	98.53
Month + 2	96.25	97.62
Month + 3	94.55	85.51
Month + 4	74.84	92.64
Month + 5	99.24	96.58
Month + 6	95.57	85.07
Month + 7	99.24	66.14
Month + 8	97.77	58.72
Month + 9	97.24	68.48
Month + 10	98.37	89.54
Month + 11	79.79	99.59
Total	93.49	85.78

[그림16] 최종 모델과 단일 ARIMA 예측 정확도 비교
(2021.10 - 2022.09)

3.2 향후 과제

3.1에서 살펴봤듯 전반적으로 모델의 예측 정확도가 낮은 편은 아니지만, 명절과 같이 화물량의 추이가 급격히 변화하는 특정 기간의 경우 예측 성능이 현저히 떨어지고 있다. 그러므로 해당 부분을 해결할 세 가지 방안을 제시하고자 한다.

3.2.1 예측 가중치 부여

전후 기간에 급격한 변동이 있는 추석, 설과 같은 특정 기간에 가중치를 설정하는 방법을 시도할 수 있다. 물량 수요가 급등하는 명절 전주에는 양의 가중치를, 급락하는 차주에는 음의 가중치를 주는 등 예측 모델이 그 변화를 반영하여 예측할 수 있도록 한다면 현재 예측 정확도보다 더 좋은 성능을 기대할 수 있을 것이다.

3.2.2 특정 기간 예측 모델 개발

또 다른 방안으로는, 예측 성능이 낮았던 특정 기간만을 따로 분리하여 모델링을 시도해볼 수 있을 것이다. 데이터의 양이 적어 학습 과정의 우려가 있지만, 현재의 데이터와는 다소 차이가 크기 때문에 아예 따로 해당 기간만을 학습시킨다면 더 나은 예측값을 얻을 것으로 기대된다.

3.2.3 화물 중심 변수 추가 생성

본 프로젝트에서 활용한 요인의 대다수는, 프로젝트 중반까지의 방향성에 맞게 화물뿐만 아닌 수하물과 우편물까지 합친 '총화물량'을 기준으로 구성된 요인들이었다. 그러다가 후반부에 수하물과 우편물을 제외한 '화물'만 예측하는 것으로 방향을 수정하게 되어, 이에 맞는 요인을 찾는 데에 시간적 한계가 있었다.

그러므로 수하물이나 우편물과 달리 화물에 영향을 주는 다른 요인들을 추가 탐색하여 모델링에 활용한다면 현재 성능보다 더 뛰어난 모델을 개발할 수 있을 것이다.

4. 부록

4.1 데이터 명세

본 프로젝트에서 활용한 데이터의 목록과 데이터별 컬럼 명세는 다음과 같다.

데이터	구분	중요도	제공기관	비고
항공화물 실적	정형/분석	필수	한국공항공사	운항 일자, 화물량
공휴일 정보	반정형/분석	필수	한국천문연구원	일자별 공휴일 종류

[표1] 활용 데이터 목록

4.1.1 항공화물 실적

컬럼명	데이터 타입	예시
공항코드	Categorical	GMP
일자	DateTime	20100101
화물_도착/출발	Numerical	190
수하물_도착/출발	Numerical	7923
우편물_도착/출발	Numerical	0
목적지	Categorical	HIN
운항횟수_도착/출발	Numerical	1
지연횟수_도착/출발	Numerical	0
결항횟수_도착/출발	Numerical	0
노선구분(국내/국제)	Categorical	D
정기/부정기	Categorical	0
여객기/화물기	Categorical	0
공급좌석수_도착/출발	Numerical	162
유임여객_도착/출발	Numerical	39
무임여객_도착/출발	Numerical	1
환승여객_도착/출발	Numerical	0

[표2] 항공화물 실적 데이터 컬럼 명세

4.1.2 공휴일 정보 데이터

컬럼명	데이터 타입	예시
locdate	DateTime	20100101
dateName	Categorical	신정

[표3] 공휴일 정보 데이터 컬럼 명세

4.1.3 전처리 이후 학습 데이터

컬럼명	데이터 타입	예시
시작일자	DateTime	2010-01-03
종료일자	DateTime	2010-01-09
월	Categorical	1월
계절	Categorical	겨울
운항횟수	Numerical	1758
노선수	Numerical	138
arima_pred	Numerical	2967886.796
총화물량	Numerical	3372722
공휴일	Numerical	0
공휴일여부	Categorical	1
주말제외휴일	Categorical	1
주말동시휴일	Categorical	0
휴일1주전/후	Categorical	0

[표4] 전처리 후 주 단위 데이터 컬럼 명세

컬럼명	데이터 타입	예시
일자	DateTime	2010-01-03
계절	Categorical	겨울
요일(월요일~일요일)	Categorical	0
주말	Categorical	1
운항횟수	Numerical	300
노선수	Numerical	21
총화물량	Numerical	347016
주단위예측값	Numerical	1816007
공휴일	Categorical	0
주말제외휴일	Categorical	1
주말동시휴일	Categorical	0
휴일1주전/후	Categorical	0

[표5] 전처리 후 일 단위 데이터 컬럼 명세

4.2 코드 명세

◆ 구글 드라이브 연결 및 경로 설정

원천 데이터의 1차 전처리가 끝난 후 해당 데이터를 구글 드라이브에 업로드하여 Colab으로 작업을 진행하였다. 이를 위해 구글 드라이브에 연결하고, 파일을 저장하고 읽어올 경로를 설정하였다.

```
from google.colab import drive
drive.mount('/content/drive')
# 데이터 접근/저장경로
# path = 'C:/Users/user/Documents/intern_project/oct_project/'   # 로컬
path = '/content/drive/MyDrive/공항공사_인턴/화물수요예측/최종/'   # 구글 드라이브
```

◆ 필요한 라이브러리 설치 및 import

전처리와 모델링 과정에서 필요한 라이브러리를 pip install 명령어로 설치한 후 import하여 사용하였다. 처음 라이브러리를 설치한 후에는 반드시 런타임을 재시작하고 다시 코드를 수행하여야 한다. 이후 시각화했을 때 한글이 제대로 출력되는지 확인하였다.

```
!pip install pycaret        # autoML 모듈 설치
!pip install pmdarima    # autoARIMA 설치
!pip install catboost       # catboost 설치
!pip install xgboost        # xgboost 설치
!pip install numba --upgrade   # numba 버전 업데이트

#한글 폰트 설치
!sudo apt-get install -y fonts-nanum
!sudo fc-cache -fv
!rm ~/.cache/matplotlib -rf
#Runtime Menu => Restart Runtime / command+ctrl=M : 런타임 다시 시작 후 처음부터 다시 진행 필요
```

```
# 필요 라이브러리 불러오기
import pandas as pd
import numpy as np

# 시간 관련
from datetime import datetime, date, timedelta
from dateutil import relativedelta
```

```
# plot 시각화
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# API
import requests
import json
from pprint import pprint

# 모델링
import pycaret
from sklearn.metrics import mean_squared_error
# from sklearn.metrics import mean_absolute_percentage_error
from statsmodels.tsa.arima.model import ARIMA
from pmdarima.arima import ndiffs
import pmdarima as pm
import joblib
from pycaret.regression import *

# plot 한글 확인
plt.rc('font', family='NanumBarunGothic')
plt.text(0.3, 0.3, '한글', size=100)
```

◆ 화물 실적 원천 데이터 가공(로컬에서만 실행할 것)

pd.read_csv로 원천 데이터를 불러서 필요한 컬럼만 추출한 후, 집계를 위해 데이터의 형 변환을 진행하였다.

```
data = pd.read_csv(path + '화물실적(2000.01-2022.09).csv', encoding='cp949')
df = data[['일자', '운항횟수_도착', '운항횟수_출발', '화물_도착', '화물_출발', '수하물_도착', '
수하물_출발', '우편물_도착', '우편물_출발']]  # 필요한 컬럼 추출
df = df.iloc[1:,:]  # 0번째 행(영문명 컬럼) 삭제
df.iloc[:, 1:] = df.iloc[:, 1:].astype(float)  # 총화물량 계산을 위한 데이터형 변환
```

이후 화물량의 합을 집계하여 새로운 파생 변수를 생성하고(원천 데이터를 가공하는 과정에서는 화물, 수하물, 우편물을 일단 모두 저장함), 필요한 부분만 추출하였다.

```
## 파생 변수 생성
df['총화물량'] = df['화물_도착'] + df['화물_출발'] + df['수하물_도착'] + df['수하물_출발'] +
df['우편물_도착'] + df['우편물_출발']
df['화물'] = df['화물_도착'] + df['화물_출발']
df['수하물'] = df['수하물_도착'] + df['수하물_출발']
df['우편물'] = df['우편물_도착'] + df['우편물_출발']
df['운항횟수'] = df['운항횟수_도착'] + df['운항횟수_출발']
df = df[['일자', '총화물량', '화물', '수하물', '우편물', '운항횟수']]  # 필요한 컬럼만 추출
# 추후 연산을 위한 '일자' 컬럼 데이터형 변환  # object -> string
df['일자'] = df['일자'].astype(str)
```

이후 '일자'를 기준으로 집계하여 일자 별 '노선 수'와 '총화물량' 값을 얻었고, 해당 데이터프레임을 저장하였다.

```
dfdf2 = df.groupby('일자').count() # '일자' 기준 몇 개의 노선이 있었는지 총합 계산
df2 = df.groupby(['일자']).sum()  # '일자'를 기준으로 '총화물량' 덧셈 연산
df2['노선수'] = dfdf2['총화물량']

df2.to_csv(path+'총화물량(2000-2022)_정리본.csv', encoding='cp949')  # 데이터프레임 저장
```

◆ 공휴일 데이터 수집

공공데이터포털에서 공휴일 데이터에 대한 API 키를 발급받은 후, 해당 URL로부터 데이터를 받아오는데 이를 다음과 같은 함수로 정의하여 활용하였다.

```
AUTH_KEY = ''    # API Key

def getHolidays(year,month,key):
    url='http://apis.data.go.kr/B090041/openapi/service/SpcdeInfoService/getRestDeInfo'
    params={  # 요청 인자 값
        'solYear':str(year),                # 연도
        'solMonth':str(month).zfill(2),    # 월
        '_type':'json',                     # 반환 형태
        'ServiceKey' : key                    # API key
    }

    res = requests.get(url,params=params)      # 데이터 요청 후 반환
    dic = json.loads(res.text)                   # 반환된 데이터 로드(딕셔너리)
    counts = dic['response']['body']['totalCount']  # 필요한 부분 추출

    if counts < 1 :   # 해당 데이터 없을 경우
        return []      # 빈 리스트 return
```

```
    # 데이터 있을 경우 item에 저장하여 return
    item =  dic['response']['body']['items']['item']
    if counts == 1:
        return [item]
    return item
```

정의된 함수를 활용하여 2010년부터 2022년까지 모든 공휴일을 받아오고(실제로는 2004년부터 데이터가 존재함) 이를 데이터프레임으로 저장하였다.

```
holidays =[]
# 2010 ~ 2021 공휴일
for year in range(2010, 2022):    # 연도
    for month in range(1, 13):   # 월
        holidays.extend(getHolidays(year,month,AUTH_KEY))

# 2022.01 ~ 2022.09 공휴일
for month in range(1,10):
    holidays.extend(getHolidays(2022,month,AUTH_KEY))

# 일자, 공휴일 정보 저장
df_holiday = pd.DataFrame(holidays,columns=['locdate','dateName'])
df_holiday.to_csv(path+"공휴일_2010.csv", index = False)
```

◆ 데이터 전처리 - return_daily_df()

앞서 수집 및 가공된 화물 실적(일 단위) 데이터와 공휴일 데이터의 전처리 과정은 다음과 같이 함수로 정의하면서 활용하였다.

공휴일 데이터(holi)를 받아와서 잘못된 명칭(ex_holi)으로 되어있는 행을 추출한 후 '공휴일' 명칭을 올바른 명칭(fm_holi)로 통일시켰다.

```
def clean_holi(holi, ex_holi, fm_holi):  ## 동일 공휴일 이름 통일
    holi.loc[holi['공휴일'] == ex_holi, '공휴일'] = fm_holi
    return
```

날짜(date_today)를 받아와서 해당 일자가 어떤 요일(weekday())인지 확인한다.

```
def check_day(date_today):  ## 주말(토, 일요일) 여부 확인
    year = round(date_today / 10000)
    mon = round((date_today % 10000) / 100)
    day = date_today % 100
    today_w = datetime(year, mon, day).weekday()  # 일자 입력 -> 요일(숫자) 반환
    return today_w
```

데이터를 받아와서 행(일자)별로 공휴일이 맞는지(6번째 열인 '공휴일'의 값이 1인지) 아닌지를 확인하고, 동시에 공휴일이라면 주말인지(14번째 열인 '주말'의 값이 1인지) 아닌지를 확인한 후 경우에 따라 열 값('주말제외휴일', '주말동시휴일' 값)을 바꿔주었다.

```
def check_holi(tot_df, i):
    if tot_df.iloc[i,6] == 0: # 공휴일이 아닌 경우('공휴일' 컬럼 값: 0)
        pass
    else:  # 공휴일인 경우('공휴일' 컬럼 값: 1)
        if tot_df.iloc[i,14] == 0: # 주말이 아닌 공휴일('주말' 컬럼 값: 0)
            tot_df.loc[i, '주말제외휴일'] = 1
        else:  # 공휴일이면서 주말인 경우
            tot_df.loc[i, '주말동시휴일'] = 1
    return tot_df
```

마찬가지로 데이터를 받아 행(일자)별로 주말제외휴일과 주말동시휴일의 1일 전/후인지의 여부를 판단하기 위해 새로운 열('주말제외휴일1일전', '주말제외휴일1일후', '주말동시휴일1일전', '주말동시휴일1일후')과 그 값을 각각 추가하였다.

```
def prev_next_holi(tot_df, holiday):
    prev_col = holiday + '1일전'  # ex: '주말제외휴일1일전'
    next_col = holiday + '1일후'  # ex: '주말제외휴일1일후'

    # 새로운 컬럼 생성 후 0으로 초기화  # ex: 주말제외휴일1일전' = 0
    tot_df[prev_col] = 0
    tot_df[next_col] = 0

    tot_df.loc[tot_df[holiday]!=0, prev_col] = 1  # 휴일이면, 해당 행의 '~1일전' 값: 1
    tot_df.loc[tot_df[holiday]!=0, next_col] = 1
    tot_df[prev_col] = tot_df[prev_col].shift(-1)  # 1일 전 행의 '~1일전' 컬럼 값: 1
    tot_df[next_col] = tot_df[next_col].shift(1)  # 1일 후 '~1일후' 컬럼 값: 1
    tot_df.fillna(0, inplace=True) # 결측값 0으로 대체
    return
```

데이터의 전처리를 한 번에 처리하기 위해 함수로 정의하였다(return_daily_df()). 우선 공휴일 데이터를 로드한 후(holi) 여러 개의 공휴일이 겹치는 행(일자)을 삭제하였다(drop_duplicates('일자')). 그 후 clean_holi로 공휴일 명칭을 통일시키고, '공휴일여부' 변수를 생성하였다.

```
def return_daily_df():
    holi = pd.read_csv(path+'/데이터_최종/공휴일_2010.csv')     ## 공휴일 데이터 로드
    holi.columns=['일자', '공휴일']  # 필요한 컬럼 추출
    holi = holi.drop_duplicates('일자')  # 여러 휴일이 겹치는 일자는 행 하나만 남김

    # 공휴일 명칭 통일
    clean_holi(holi, '1월1일', '신정')  # '1월1일' -> '신정'
    clean_holi(holi, '어린이 날', '어린이날')
    clean_holi(holi, '부처님오신날', '석가탄신일')
    clean_holi(holi, '제21대 국회의원선거', '국회의원선거')
    clean_holi(holi, '국회의원선거일', '국회의원선거')
    clean_holi(holi, '대통령선거일', '대통령선거')
    clean_holi(holi, '동시지방선거일', '동시지방선거')
    holi['공휴일여부'] = 1  # 공휴일인 경우: 1, 아닌 경우: NaN
```

이후 가공된 화물실적 데이터를 로드하여 '화물량' 합을 '총화물량'으로 설정하고(이전 데이터에서는 수하물과 우편물까지 합한 값이었음) 공휴일 데이터와 결합한 뒤 결측값을 0으로 대체하였다.

```
    df=pd.read_csv(path+'/데이터_최종/총화물량(2000-2022)_정리본.csv', encoding='cp949', low_memory=False)
    df = df[['일자', '화물', '운항횟수', '노선수']]  # '화물량' 기준
    df = df[df['일자'] >= 20100101]  # 공휴일 데이터와 일자 범위 맞추기
    # 컬럼명 변경: 화물 -> 총화물량
    df.rename(columns={'화물': '총화물량'}, inplace=True)

    # 공휴일 데이터프레임, 화물 데이터프레임 결합
    holi_df = pd.merge(df, holi, how='left', on='일자')
    holi_df = holi_df.fillna(0)  # '공휴일여부' 컬럼 - 공휴일인 경우: 1, 아닌 경우: 0
```

그리고 일자별로 무슨 요일인지를 확인하고(check_day) 요일별 컬럼을 생성하여 해당하면 1, 아니면 0으로 바꾸는 가변수화 과정을 거쳤다. 그 후 해당 데이터프레임을 기존 데이터와 결합하고 '주말' 변수를 생성한 후 저장하였다.

```
    holi_df['요일'] = holi_df['일자'].apply(lambda x: check_day(x))
    day_df = pd.get_dummies(holi_df['요일'])     ## 요일 기준 가변수화
    day_df.columns = ['월요일', '화요일', '수요일', '목요일', '금요일', '토요일', '일요일']
    tot_df1 = pd.concat([holi_df, day_df], axis=1)  # 정리된 데이터와 결합
    tot_df1['주말'] = day_df[['토요일', '일요일']].sum(axis=1)  # '주말' 변수 생성
    tot_df1 = tot_df1.drop(columns='요일')     # '요일' 컬럼 삭제
    tot_df = tot_df1.reset_index(drop=False)  # 'index' 컬럼 생성
    tot_df.to_csv(path+'/데이터_최종/일별_공휴일추가_정리본.csv', encoding='cp949')
```

이후 주말제외휴일과 주말동시휴일 여부, 해당 휴일들의 1주 전/후 여부를 판단하여 값을 넣고 저장하였다. 이로써 일 단위 데이터의 1차 전처리를 마쳤다.

```
    # '주말제외휴일', '주말동시휴일' 여부 판단
    tot_df['주말제외휴일'] = 0  # 주말이 아닌 휴일
    tot_df['주말동시휴일'] = 0  # 주말이면서 휴일
    for i in range(len(tot_df)):
        tot_df = check_holi(tot_df, i)
    tot_df = tot_df.drop('index', axis=1)  # 'index' 컬럼 삭제

    ## 휴일 1일전후 여부 판단
    for i in ['주말제외휴일', '주말동시휴일']:
        prev_next_holi(tot_df, i)
    tot_df.to_csv(path+'/데이터_최종/일별_최종_정리본.csv', encoding='cp949')
    return tot_df
```

◆ 데이터 전처리 - return_monthly_df()

다음으로 주 단위 데이터를 생성하는 함수(return_monthly_df())를 정의하였다.

우선 주 단위 데이터로 변환된 이후 활용하는 함수를 정의하였다. 해당 함수는 데이터(dataframe)를 받아 행(주 단위)별로 공휴일(holiday)의 1주 전후에 속하는지를 판단한다. 이때 먼저 공휴일 별로 겹치지 않게 열을 새로 만들고(before_col, after_col) 0으로 초기화한 후,

```
def before_after_holi(dataframe, holiday):
    before_col = holiday + '1주전'  # ex: '기독탄신일1주전'
    after_col = holiday + '1주후'  # ex: '기독탄신일1주후'

    ex_cols = dataframe.columns.tolist()  # 데이터프레임 기존 컬럼 리스트
    if (before_col in ex_cols) or (after_col in ex_cols):  # 이미 만들어진 컬럼인 경우 다시 만들지 않음
        return
    # 새로운 컬럼 생성 후 0으로 초기화
    dataframe[before_col] = 0  # ex: '주말제외휴일1주전' = 0
    dataframe[after_col] = 0  # ex: '주말제외휴일1주후' = 0
```

해당 공휴일이 있는 행들을 뽑아서 '(공휴일)1주전/후' 열값을 1로 설정한 뒤, 전체 '1주전' 값들은 한 칸씩 앞당기고(shift(-1)) '1주후' 값들은 한 칸씩 뒤로 미룸(shift(1))으로써 실제 공휴일의 전주, 후주인 행(주)들의 값이 1로 설정되도록 하였다. 이후 값들을 앞당기고 미루면서 생긴 결측값들을 0으로 대체하였다.

```
    dataframe.loc[dataframe[holiday]!=0, before_col] = 1  # 휴일이면, 해당 로우의 '~1주전' 값을 1로 설정
    dataframe.loc[dataframe[holiday]!=0, after_col] = 1
    dataframe[before_col] = dataframe[before_col].shift(-1)  # 해당 휴일의 1주 전 '~1주전' 컬럼 값: 1
    dataframe[after_col] = dataframe[after_col].shift(1)  # 해당 휴일의 1주 후 '~1주후' 컬럼 값: 1
    dataframe.fillna(0, inplace=True)
    return
```

다음은 데이터(df)를 받아와 DateTime Index를 활용해 월 정보를 추출(df.index.month.values)하고, 그것에 맞게 계절 정보도 추가하는 함수이다.

```
def plus_mon_season(df):
    df['월'] = [f"{df.index.month.values[i]}월" for i in range(df.shape[0])]
    df['계절'] = 0
    df.loc[(df.월 == '12월') | (df.월 == '1월') | (df.월 == '2월'),"계절"] = "겨울"
    df.loc[(df.월 == '3월') | (df.월 == '4월') | (df.월 == '5월'),"계절"] = "봄"
    df.loc[(df.월 == '6월') | (df.월 == '7월') | (df.월 == '8월'),"계절"] = "여름"
    df.loc[(df.월 == '9월') | (df.월 == '10월') | (df.월 == '11월'),"계절"] = "가을"
    return df
```

일 단위 데이터를 받아와(return_daily_df()) 먼저 공휴일별 컬럼을 생성하고 결합하였다. 이후 '일자' 컬럼을 DateTimeIndex로 설정한 후 2010년부터 추출하였다.

```
def return_monthly_df():
    # 각 공휴일별 여부 컬럼 추가
    df = return_daily_df()  # 일단위 기본 데이터 생성
    p = pd.get_dummies(df['공휴일'])  # '공휴일' 기준 가변수화
    p = p.drop(0, axis=1)
    df_w = pd.concat([df, p], axis=1)  # 일자별 기본데이터에 결합
    # 불필요한 컬럼 삭제
    df_w = df_w.drop(['월요일', '화요일', '수요일', '목요일', '금요일', '토요일', '일요일', '주말'], axis=1)
    df_w = df_w.drop(['주말동시휴일1일전', '주말동시휴일1일후',  '주말제외휴일1일전', '주말제외휴일1일후'], axis=1)
    df_w.reset_index(drop=True)  # 인덱스 reset
    # '일자' 컬럼 형 변환 후 인덱스로 설정
    df_w['일자']=df_w['일자'].apply(lambda x: pd.to_datetime(str(x), format='%Y%m%d'))
    df_w.set_index(df_w['일자'], inplace=True)  # '일자'를 인덱스로 설정
    df_w = df_w.loc['20100103':, :]
```

이후 일 단위 데이터를 '일요일 - 토요일' 기준의 주 단위 데이터로 변환하고(resample) 월과 계절 정보를 추가하였다(plus_mon_season). 이후 공휴일별로 1주 전후 여부를 판단한 뒤(before_after_holi) 최종 주단위 컬럼 데이터를 완성하였다.

```
    week_df = df_w.resample('W-Sat').sum()  # '공휴일' 컬럼 삭제됨
    week_df = week_df.set_index(week_df.index - pd.Timedelta(days=6))  # 시작일
    week_df = plus_mon_season(week_df)
    week_df.to_csv(path+'/데이터_최종/week_df_정리본.csv', encoding='cp949')  # 데이터
프레임 저장

    for i in week_df.columns[4:24]:  # '주말제외휴일' ~ '현충일'
        before_after_holi(week_df, i)
    week_df.to_csv(path+'/데이터_최종/주단위_컬럼정리_정리본.csv', encoding='cp949')
    return week_df
```

◆ Auto ARIMA 모델링 - 주 단위 예측

완성된 주 단위 데이터를 바탕으로 화물량 값을 가지고 Auto ARIMA를 수행하였다. 여러 번 모델을 fitting하는 작업을 거치고(forecast_one_step), 최적의 차분값들을 도출시켜 최적의 모델을 얻는다.

```
def forecast_one_step(model): ## 모델 기간별 예측값 도출
  fc, conf_int = model.predict(n_periods=1 # 한 스텝씩!
      , return_conf_int=True)                # 신뢰구간 출력
  return (
      fc.tolist()[0],
      np.asarray(conf_int).tolist()[0]
  )

def part_autoARIMA(full_data, train, test, path, name):
  print('part_autoARIMA 시작')
  # train, test 데이터
  y_train = train
  y_test = test

  kpss_diffs = ndiffs(y_train, alpha=0.05, test='kpss', max_d=6)  # kpss test 최적차분값
  adf_diffs = ndiffs(y_train, alpha=0.05, test='adf', max_d=6) # adf test 최적차분값
  n_diffs = max(adf_diffs, kpss_diffs) # 추정 최적차분값 중 높은 차분값 사용
```

```
model = pm.auto_arima(y = y_train,  # 학습 데이터
                      d = n_diffs,  # 차분 차수, ndiffs 결과!
                      start_p = 1,  # 최소 p값 설정
                      max_p = 5,    # 최대 p값 설정
                      start_q = 1,  # 최소 q값 설정
                      max_q = 5,    # 최대 q값 설정
                      m = 1,  # 계절적 차분이 필요할 때 쓸 수 있는 모수로 m=4이면 분기별, m=12면 월별, m=1이면 계절적 특징을 띠지 않는 데이터
                      seasonal = False, # 계절성 ARIMA가 아니라면 False
                      stepwise = True,  # 최적의 모수를 찾기 위해 쓰는 힌드만 - 칸다카르 알고리즘을 사용할지의 여부
                      trace=True,  # stepwise로 모델을 적합할 때마다 결과를 프린트하고 싶을 때 사용
                      )
```

이후 ARIMA의 예측값을 예측 데이터와 전체 데이터 각각 얻고, RMSE 값을 기준으로 비교하고자 하였다.

```
# test data 예측값 산출
forecasts = []
y_pred = []
pred_upper = []
pred_lower = []
for new_ob in y_test.values:
    fc, conf = forecast_one_step(model)
    y_pred.append(fc)
    pred_upper.append(conf[1])
    pred_lower.append(conf[0])
    model.update(new_ob)  # 모델 업데이트

# 전체 데이터 예측값 산출
forecasts2 = []
y_pred2 = []
pred_upper2 = []
pred_lower2 = []
for new_ob in full_data.values:
    fc, conf = forecast_one_step(model)
    y_pred2.append(fc)
    pred_upper2.append(conf[1])
    pred_lower2.append(conf[0])
    model.update(new_ob)  # 모델 업데이트
```

```
  print("test RMSE : ",round(np.sqrt(mean_squared_error(y_test.values,y_pred)),2))
  print("total RMSE : ",round(np.sqrt(mean_squared_error(full_data.values,y_pred2)),2))

  y_predict = pd.DataFrame(y_pred, index = y_test.index, columns=['Prediction'])
  y_predict2 = pd.DataFrame(y_pred2, index = full_data.index, columns=['Prediction'])

  # 그래프
  fig, axes = plt.subplots(1, 1, figsize=(15, 8))
  plt.plot(y_test, label='Test')                # 테스트 데이터
  plt.plot(y_predict, label='Prediction')    # 예측 데이터
  plt.legend()
  plt.show()

  y_predict2.to_csv(f"{path}데이터_최종/{name}.csv", encoding = "cp949")  # 전체 데이터 예측값 저장
  print('part_autoARIMA 완료')
  return y_pred, y_pred2, y_predict2
```

위에 정의된 함수들을 호출(return_monthly_df(), part_autoARIMA)하고 이를 토대로 실제 주 단위 예측을 주어진 기한대로(학습 기간 상한: train_upper, 예측 기간 하한: test_lower, 예측 기간 상한: test_upper) 수행하고 그 예측값을 'arima_pred' 변수로 저장하는 함수이다.

```
def return_pred_arima():
  print('return_pred_arima 시작')
  data = return_monthly_df()
  df = data[['총화물량']]   # 일자(인덱스), 총화물량
  y_train = df[df.index < train_upper]
  y_test = df[(df.index >= test_lower) & (df.index <= test_upper)]
  print('train:',len(y_train), 'test:',len(y_test))

  # AutoARIMA 수행
  y_pred, y_pred2, y_predict2 = part_autoARIMA(full_data = df, train = y_train, test = y_test, path = path, name = f"주별_arima_pred(화물)_{test_lower}")

  data['arima_pred'] = y_predict2["Prediction"].values
  print('return_final_daily 완료')
  return data
```

◆ Auto ML 모델링 - 주 단위, 일 단위

주 단위, 일 단위로 예측하는 Auto ML의 모델링 함수는 다음과 같다. 우선 데이터를 받고 모델의 세팅을 설정한 다음, RMSE(loss) 값을 기준으로 최적의 모델을 도출한다. 이후 해당 모델로 테스트, 전체 데이터에 대한 예측값을 얻는다.

```
def part_autoML(num, train, test, target, train_size, cat_features, num_features, loss, path, name):
  df = pd.concat([train,test])    # train, test 데이터 합치기
  reg_test = setup(data = train,                # 학습 데이터
                   target = target,             # 종속변수 설정
                   train_size = train_size,   # 학습 데이터 크기 설정
                   fold = 5,                    # 반복 횟수
                   categorical_features = cat_features,   # 범주형 변수 지정
                   numeric_features = num_features,       # 수치형 변수 지정
                   fold_shuffle = True,        # Shuffle 여부
                   session_id = 42)            # random_state 값
  best = compare_models(sort = loss)
  best_tune = tune_model(best)  # 최적 모델로 모델링

  # test 데이터와 전체 데이터의 예측값 도출
  test_pred=predict_model(best_tune, data=test.drop(columns=[target],axis=1))["Label"]
  total_pred=predict_model(best_tune, data=df.drop(columns=[target],axis=1))["Label"]
  print("test RMSE : ",round(np.sqrt(mean_squared_error(y_test.values,y_pred)),2))
  print("total RMSE : ",round(np.sqrt(mean_squared_error(full_data.values,y_pred2)),2))

  # test data 예측 그래프
  fig, axes = plt.subplots(1, 1, figsize=(15, 8))
  plt.plot(test[target], label='총화물량')                        # 모든 데이터
  plt.plot(test_pred, label='총화물량 예측',color = "red")  # 예측 데이터
  plt.title(f'{name}:{test_upper}')
  plt.legend()
  plt.show()

  if num == 1:
    label = '주단위'
  else:
    label = '일단위'

  y_predict = pd.DataFrame(total_pred).rename(columns = {"Label":f"{label}예측값"})
  y_predict.to_csv(f"{path}/데이터_최종/{name}.csv", encoding = "cp949")  # 예측값 저장
  return test_pred, total_pred, y_predict, best_tune
```

앞서 정의된 Auto ML 모델링 함수(part_autoML)를 실제로 수행하는 함수는 다음과 같다(pred_autoML). 우선 Auto ML의 경우 주 단위와 일 단위 총 두 번 실행되기 때문에 이를 구분할 변수(num)를 설정해주었다.

이때 주 단위 예측을 한다면 주 단위 ARIMA 예측값까지 얻은 데이터(return_pred_arima())를 완성하고, 일 단위 예측을 한다면 기존에 완성된 학습 데이터(최종 일 단위 데이터)가 있는지 판단한 후 없다면(first == 1) 만들고(make_final_daily() - 추후 정의), 있다면 해당 데이터를 로드하여 Auto ML을 수행하고 최종 일일 예측값을 반환한다.

```
def pred_autoML(num=0, first=0):
  if num == 1:     ## 2단계 주 단위 autoML
    data = return_pred_arima()   # arima 예측값 포함 주 단위 데이터 생성
  else:            ## 3단계 일 단위 autoML
    if first == 1:  # 일 단위 데이터 완성하고 return
      print("기존 데이터 없음! 일 단위 데이터 새로 생성 시작")
      make_final_daily()  # 최종 일 단위 데이터 생성
      print("최종 일 단위 데이터 생성 완료")
      return
    else:
      data=pd.read_csv(path + '/데이터_최종/최종일단위데이터_정리본.csv', encoding = 'cp949').rename(columns = {'Unnamed: 0':'일자'})
      data['일자'] = data['일자'].apply(lambda x: pd.to_datetime(str(x), format = '%Y-%m-%d'))  # '일자' 형 변환
      data = data.set_index('일자', drop=True)
      print('기존에 완성된 일 단위 데이터 로드')

  train = data[data.index < train_upper]
  test = data[(data.index >= test_lower) & (data.index <= test_upper)]
  target = "총화물량"
  train_size = 0.999
  loss = "rmse"  # RMSE 기준으로 최적 모델 선정
  cat_features = cat_features = [i for i in data.columns if (data[i].nunique() < 3) | (data[i].dtype == "object")]  # 범주형 변수 정의
  num_features = [i for i in data.columns if (data[i].nunique() >= 3) & ((data[i].dtype == "int64")|(data[i].dtype == "float64"))]  # 수치형 변수 정의
  num_features.remove("총화물량")

  if num == 1:  ## 2단계 주단위 autoML
    name = f"최종주단위예측량_{test_lower}"
  else: ## 3단계 일단위 autoML
    name = f"최종일단위예측량_{test_lower}"
```

```
  # Auto ML 수행
  test_pred, total_pred, y_predict, best_tune = part_autoML(num = num, train = train, test = test, target = target, train_size = train_size, cat_features = cat_features, num_features = num_features,  loss = loss, path = path, name = name)
  evaluate_model(best_tune)

  if (num != 1) & (first != 1):  ## 3단계 일단위 autoML
    joblib.dump(best_tune, f"{path}일일화물량예측모델_{test_lower}.pkl")  # 모델 저장
  return y_predict
```

◆ 일 단위 학습 데이터 완성 - make_final_daily()

일 단위 최종 예측을 위한 학습 데이터를 완성시키는 함수로, 먼저 주 단위 예측값을 얻은 후(pred_autoML(num=1)) 해당 데이터를 다시 일 단위로 변환한다(resample). 이때 마지막 행의 날짜가 해당 월의 마지막 날인지를 판단한 후, 다운샘플링을 하면서 없어진 데이터도 함께 채움으로써 일자별 주단위 예측값을 얻었다.

```
def make_final_daily():
    print('make_final_daily 시작')
    df = pred_autoML(num=1)     ## 최종 주단위 예측값 데이터프레임
    print('2단계 pred autoML 완료')

    df = df.resample(rule='D').first().ffill()  # 주단위 -> 일단위 다운샘플링
    last_time = df.index.tolist()[-1]  # 마지막 행
    this_month_fir = last_time.replace(day=1)  # 해당 월 첫날
    last_month_last = this_month_fir - timedelta(days=1) # 지난달 마지막날
    last_date = last_month_last+relativedelta.relativedelta(months=1) #해당 월 마지막날

    n = (last_date - last_time).days # 마지막 행 일자와 당월 마지막 일자 간 차이
    if n == 0:  # 마지막 행의 일자가 당월의 마지막 날인 경우
      print("마지막 날짜까지 이미 존재함")
      pass
    else:  # 마지막 행의 날짜 이후에도 당월의 일수가 남은 경우
      # (당일 날짜 이후 ~ 당월 날짜)까지 datetime 생성
      add_date = pd.date_range(last_time, periods=n+1, freq='D')[1:]
      add_df = pd.DataFrame(index = add_date)
      # 마지막 행의 데이터로 복사(같은 1주에 속하므로 값이 동일함)
      for x in df.columns.tolist():
        add_df[x] = df.loc[last_time, x]
      df = pd.concat([df, add_df])  # 주단위 데이터 일자 완성
      print("부족한 날짜 행 채움")
```

기존의 가공된 일 단위 데이터(df2)를 불러와 일자별 주 단위 예측값 데이터(df)와 결합함으로써 최종 일 단위 학습용 데이터를 완성하였다.

```
    df2 = pd.read_csv(path+'/데이터_최종/일별_최종_정리본.csv', encoding='cp949')
    df2['일자'] = df2['일자'].apply(lambda x: pd.to_datetime(str(x), format='%Y-%m-%d'))
    df2 = df2.set_index('일자')  # '일자' DateTime Index로 설정
    df2 = df2[(df2.index >= '2010-01-03') & (df2.index <= test_upper)]  # 기간 설정
    df2 = df2.drop('Unnamed: 0', axis=1)

    # 기존 일단위 데이터프레임 + 주단위예측값, 월, 계절 컬럼
    total_df = pd.concat([df2, df], axis=1)
    total_df = plus_mon_season(total_df)
    total_df.to_csv(path+'/데이터_최종/최종일단위데이터_정리본.csv', encoding='cp949')
    print('make_final_daily 완료: 최종일단위데이터 생성 완료')
    return
```

◆ 주 단위 / 일 단위 학습 데이터 생성

정의된 함수들을 바탕으로, 실제 기간에 맞춰 학습 데이터를 한 번에 생성하였다. 예측하는 기간을 설정하고(start_test_month로부터 1년치 목록인 month_li 도출, 그에 맞는 train_upper와 test_lower, test_upper 설정), 데이터를 생성하였다(pred_autoML(first=1)).

```
start_test_month = '2021-10-01'   # 21년 10월부터 예측 시작 -> 22년 9월까지
month_li = pd.date_range(start_test_month, periods=12, freq='MS')

last_month = month_li[-1]                                  # 2022-09-01
train_upper = last_month - timedelta(days=1)    # train: 2010-01-04 ~ 2022-08-31
test_lower = last_month                                    # test: 2022-09-01 ~ 2022-09-30
test_upper = last_month + relativedelta.relativedelta(months=1) - timedelta(days=1)  # 예측 종료일

train_upper = str(train_upper)[:10]
test_lower = str(test_lower)[:10]
test_upper = str(test_upper)[:10]
print(train_upper, test_lower, test_upper)

pred_autoML(first=1)
```

◆ 일 단위 예측 후 최종 1년치 예측값 도출

생성된 학습 데이터를 가지고 일 단위 최종 예측을 수행하였는데, 월 단위로 수행하기 위해 데이터 상한/하한이 계속 변화하도록 한 후 그에 맞춰 일 단위 예측(pred_autoML())을 한 달씩 하도록 하였다. 하나의 모델이 한 달씩 예측값을 도출하면 하나의 데이터프레임(year_pred_df)에 값을 추가로 저장하고 최종 데이터를 저장하였다.

```
year_pred_df = pd.DataFrame(columns=['일단위예측값'])  # 값 저장용 데이터프레임

for i in month_li:
  train_upper = i - timedelta(days=1)   # 2010-01-04 ~ 예측시작 전달 말일까지 train
  test_lower = i    # 예측 시작일
  test_upper = i + relativedelta.relativedelta(months=1) - timedelta(days=1)  # 예측 종료일
  train_upper = str(train_upper)[:10]
  test_lower = str(test_lower)[:10]
  test_upper = str(test_upper)[:10]
  print("train: 2010-01-04 ~", train_upper)
  print("test:", test_lower, "~", test_upper)

  y_predict_df = pred_autoML()
  y_predict_df = y_predict_df[(y_predict_df.index >= test_lower) & (y_predict_df.index <= test_upper)]
  year_pred_df = pd.concat([year_pred_df, y_predict_df])

  print(len(y_predict_df))
  print(test_lower,":",test_upper)
  print(len(year_pred_df))
  print(i, "예측 완료!")

year_pred_df.to_csv(path+"일년치_최종예측값.csv", encoding='cp949') # 최종 예측값 저장
```