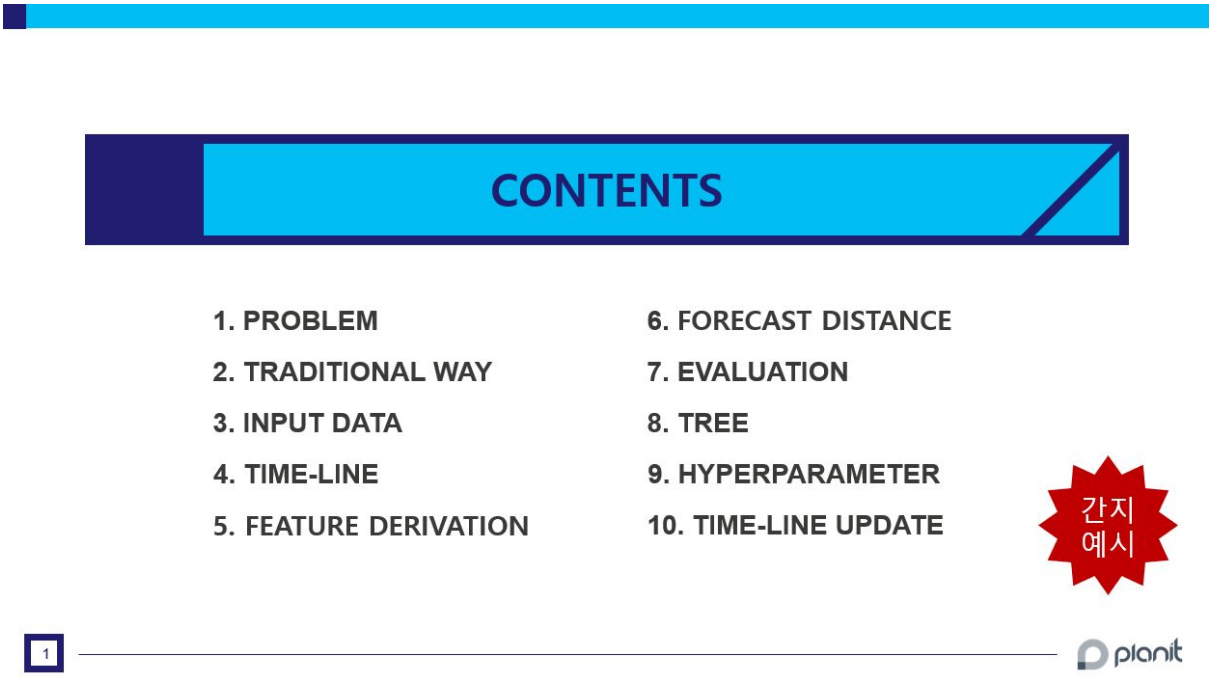
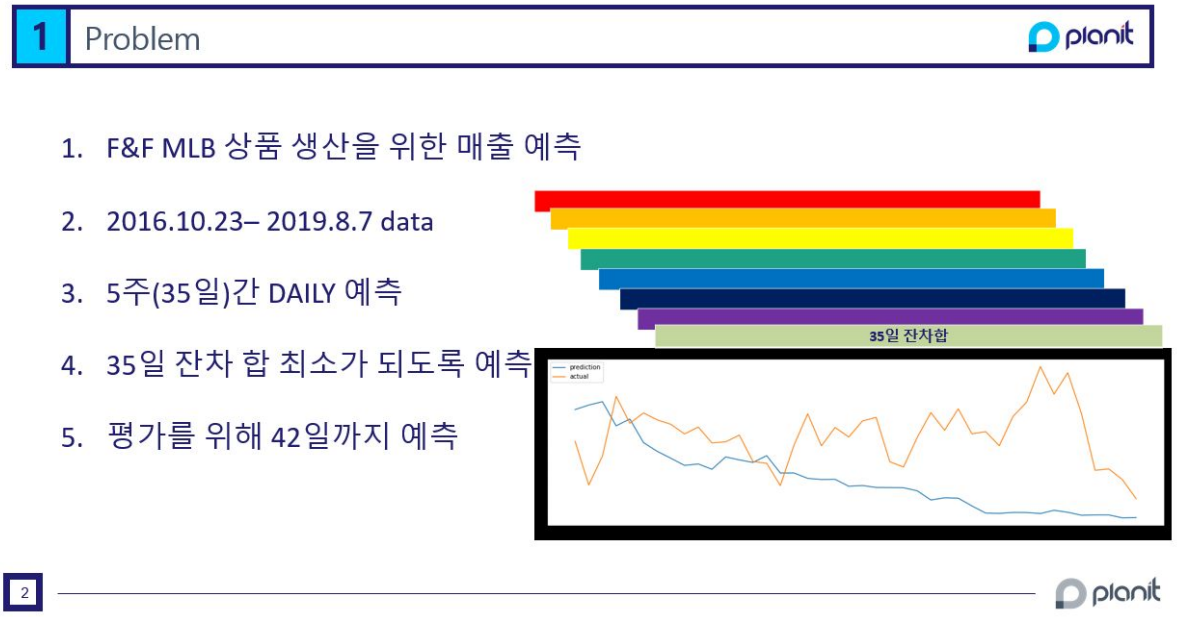
이 게시글은 머신러닝 알고리즘 중 하나인 **LGBM(Light Gradient Boosting Machine)**을 이용해 **매출을 예측**하는 모델을 소개하는 글입니다. 이 방식의 시초는 모르겠지만 kaggle 대회 grocery store sales forecasting 우승작을 참고했습니다.



먼저 목차입니다. 플랜잇 ppt 폼을 이용했는데 간지예시가 넘나 간지가 나서 그대로 살려두었습니다.
번호를 보시고 궁금하신 내용으로 바로 가셔도 되겠습니다.
목차에 대한 부가적인 설명을 드리자면,

1. 현재 해결하고 싶은 business problem이 무엇인지 정의
2. 머신러닝 알고리즘 사용 이전에는 어떤 방식으로 시계열 예측을 했나
3. 시계열 예측에서 기본 인풋 데이터 모양
4. 시계열 예측을 할 때 타임라인(?)이 어떻게 구성되는가
5. 모델을 학습시켜야 하는데 어떤 피쳐들을 구성하는가
6. Forecast Distance 란?
7. 모델이 좋은지 어떻게 평가할까
8. 트리 모양이 어떻게 되나
9. hyperparameter를 어떻게 수정할 것인가
10. bayesian optimization이 첨가된 새로운 타임라인



의류 제조, 유통업체의 생산을 위해 향후 **5주간의 매출을 예측**하고 싶습니다. 시범적으로 최근 주목하고 있는 10가지 내외의 제품을 상권, 색상별로 나누어 총 **360가지 품목**의 매출을 예측해 보기로 했습니다. 상권은 4가지이고 색상은 제품별로 다릅니다.

data 기간은 **2016.10.23 – 2019.8.7** 입니다.

하지만 모든 품목의 매출 기록 양이 다 같은 건 아닙니다. 어떤 품목은 240일밖에 없기도 하고 어떤 품목은 360일치만 있습니다. 최장 기간이 채 3년이 안 되지만, 매출이 특정 품목에 몰려 있어서 상위 품목만 고려하면 1년 6개월 정도의 기간을 채우기가 어려운 데이터입니다.

Daily, Weekly, Monthly 버전 모두 가능하겠지만 현재 일별로 해도 데이터가 적은 상태이기 때문에 **Daily 매출을 학습하고 예측**하기로 합니다.

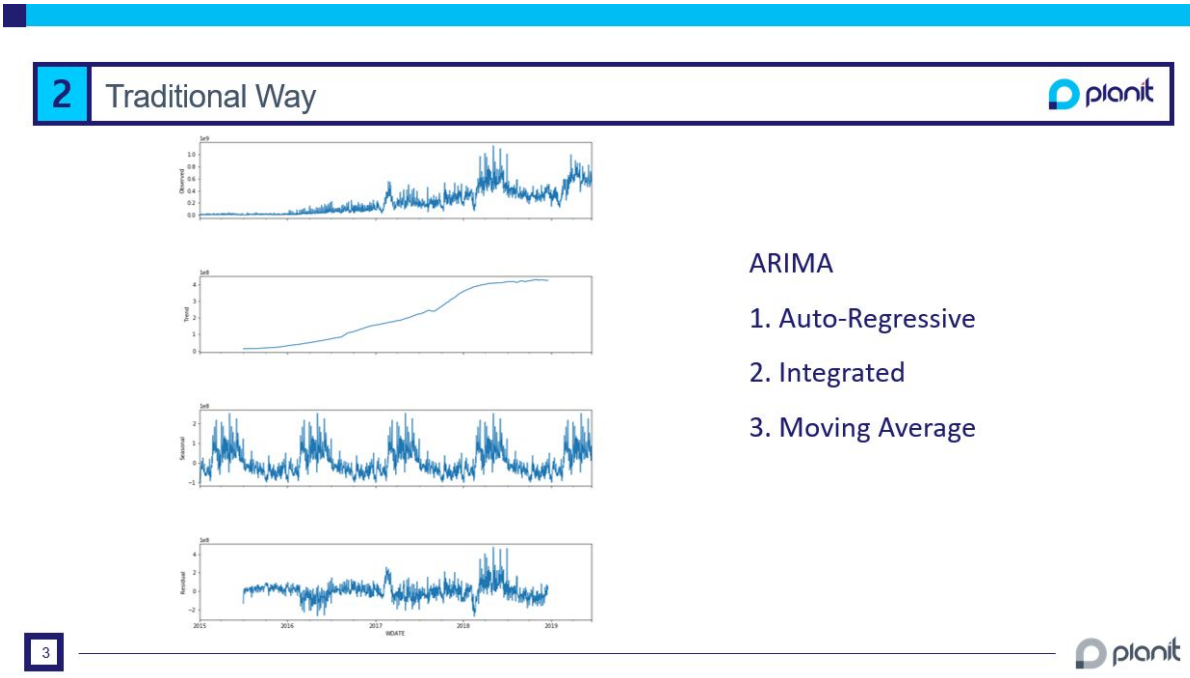
Daily 예측이 하루하루 다 정확하면 좋겠지만, 그게 어렵다면 5주간 매출을 예측하고 **잔차(실제값 – 예측값)의 합이 0에 가까우면** 그것도 현재 기업에서 요구하는 문제 해결에는 좋은 방법일 것입니다.

그래서 35일(5주)의 잔차 합도 지표로 삼을 텐데 35일을 예측하고 잔차 합을 하나만 보면 평가가 어렵습니다. **42일을 예측**하고 1-35일, 2-36일, 3-37일, 이런 식으로 총 **8개의 잔차 합**이 나오도록 예측해 봅니다. 그림으로 표현돼 있는 것이 바로 이것입니다.

색상은 예쁘게 무지개 빨주노초파남보쑥색으로 했습니다. 5-7살 때 3년간 새부산미술학원이라는 **디자인스쿨**을 다녔을 때 키웠던 저의 색감입니다.

마지막 그래프는 제가 자주 본 시계열 예측의 실패 형상이라 넣어봤습니다. 파란색이 prediction, 주황색이 actual 값입니다. 초반에 맞는 듯하다가 갈수록 예측력이 떨어지는 것이

상식에 부합하는 듯합니다. 이제 어떻게 파란색 그래프를 주황색에 가깝게 맞춰갈지 알아보겠습니다.



하지만 그 전에 머신러닝을 이용하지 않는, **과거의 시계열 분석과 예측**을 잠깐 알려드리겠습니다. 시계열 데이터에서 **"어떤 정보를 만들어내서"** 분석하고 예측했는지 보시면 앞으로의 이해도를 높일 수 있겠습니다.

왼쪽 그림은 **time-series decomposition**입니다. 예측에 쓰이진 않고 분석에 쓰이는 방법인데요. 원래 시계열 관측 값을 **추세+주기성+나머지** 로 분해한 것입니다. 영어로는 Observed = Trend + Seasonality + Residual 이네요.

Trend를 보고 **상승 또는 하강**하는지 파악할 수 있겠고요.
Seasonality를 보고 **시간에 dependant한 주기**의 기간과 변동량을 살펴볼 수 있겠습니다.
Residual은 그 외 **나머지**가 어떤 형상을 띄는지, 얼마나 많은지, 어느 기간에 특이한 지점이 있는지 정도를 볼 수 있겠습니다.

오른쪽에 '**ARIMA**'라는 과거의 시계열 예측 방법 중 하나입니다. time-series decomposition과는 다른 내용입니다.

AR(auto-regressive)는 **자기상관성**이라고 합니다. **자기 자신의 과거값**을 변수로 회귀식을 추정한다는 말인데요. 아주 rough 하게 표현하면

$$X_t = aX_{t-1} + \epsilon_t$$

요런 느낌입니다. **t 시점의 값이 t-1 시점의 값의 일차식(선형조합)**으로 추정할 수 있겠다는 겁니다.

I(integrated)는 X_t 대신 $\mathbf{X_t - X_{t-1}}$을 보겠다는 말입니다. 모든 시점의 값들을 **차분(differencing)**된 값으로 대체합니다. 이렇게 하는 이유는 정상성(stationarity)을 가진 시계열 자료로 만들고 싶기 때문입니다. 조금 더 구체적으로 아시고 싶으신 분은 https://otexts.com/fppkr/stationarity.html#fn15 를 참고하시기 바랍니다.

MA(moving average)는 **이동평균**인데요. 오해의 소지가 있습니다. 보통 **두 가지 의미**로 사용되고 있더라고요. 하나는 직관적으로 와닿는 것인데요. 일정 기간, 예를 들어 5주라고 하면, 1-5주, 2-6주, 3-7주의 평균이 계속 같도록 이후의 수치를 예측하는 방식입니다. ARIMA의 MA는 이것과 다릅니다. ARIMA의 MA는 회귀식에서 오류가 과거 여러 시간에 발생한 오류의 선형 조합이라고 가정하고 식을 만드는 겁니다.

그러면 AR과 MA가 비슷하다고 느끼실 수 있겠습니다. AR은 과거 값의 선형 조합으로 미래 값을 추정할 수 있다고 하고, MA는 오류가 과거의 오류의 선형조합이라고 하니까요. 실제 종합해서 같은 말로 알고 계시면 되겠습니다.
결론적으로 **t 시점의 값은 이전 시점 값들의 선형조합 회귀식으로 추정할 수 있다**는 사실입니다.

회귀식이라는 단어가 생소해서 **회귀분석의 어원**을 덧붙이자면, 영국의 갈톤이라는 학자가 **부모와 자식들 간의 키**의 상관관계를 분석해 재미있는 관계를 찾아냈습니다. 키가 매우 큰 부모의 자식들은 대게 크긴 하되 부모들보다는 대부분 작았고, 키가 매우 작은 부모들의 자식들은 대게 작긴 하되 부모들보다는 대부분 크다는 사실입니다. 이러한 경향은 사람들의 **키가 평균키로 회귀하려는 경향이 있음**을 의미한다고 해석했는데, 바로 이 연구에서부터 회귀분석이라는 용어가 사용되었습니다. 학자들은 평균을 매우 좋아하는 것 같습니다.

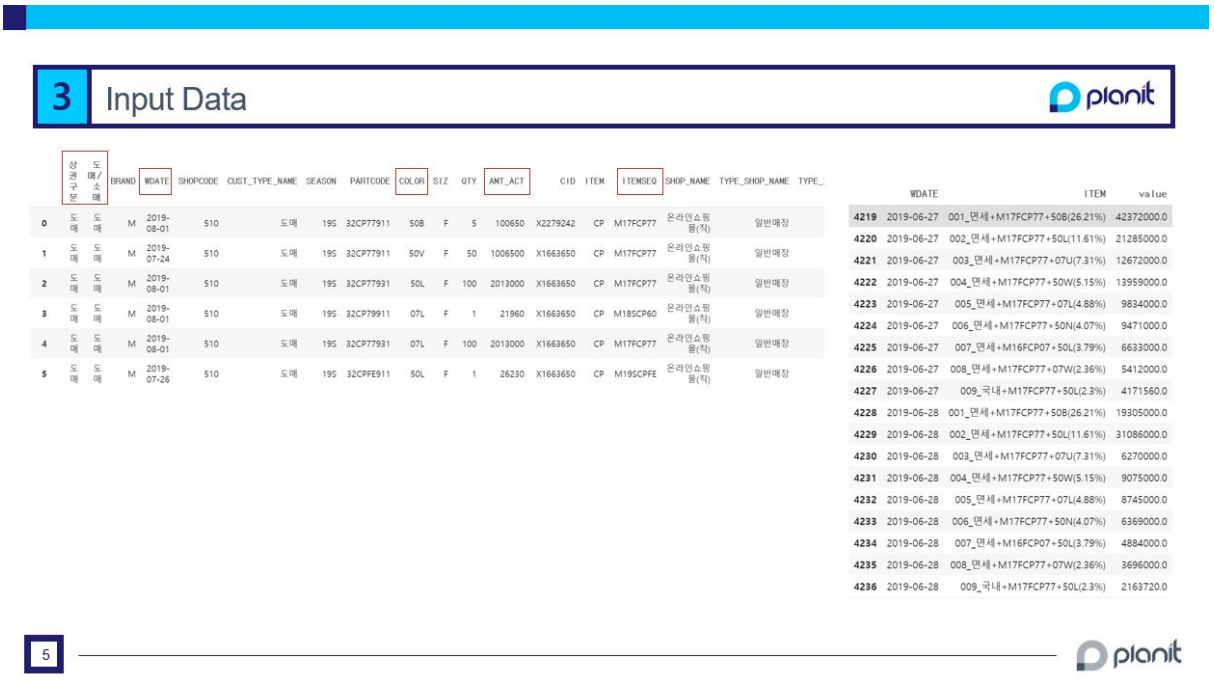
그리고 **선형**은 **상수곱과 덧셈**으로 이루어졌다는 말이고, 이차나 삼차같은 고차항이나 로그 같은 다른 함수가 아니라, 일차함수에서 보던 일차식으로 이해하시면 되겠습니다.

2 Traditional Way – F&F

1. 품목별 Weekly + Quantity Data 활용
2. 5주 이동 평균 + 연도별 추세 가중치로 반영
3. In Excel

F&F가 어떤 식으로 과거 매출을 예측하고 있었는지도 잠깐 살펴보겠습니다.
우선 일별 데이터가 아니라 **주별 데이터**로, 매출이 아니라 **수량을 기반**으로 **미래의 수량을 예측**하고 있었습니다. 제가 첫 번째 의미로 썼던 **5주 이동평균**을 이용하고 계셨고, **연도별 추세를 가중치로 반영**하신다고 하셨습니다. 이걸 엑셀로 하고 계셔서 신기하기도 하고 궁금하기도 해서 어떤 식으로 가중치를 구하시는지 여러 번 여쭤봐도 부끄러워하시며 안가르쳐 주시더라고요. 저는 매우 **합리적**인 방법이라고 생각했는데 그렇게 생각하지 않으셨나봅니다. 아니면 **저만 친하다고 생각**하고 과도한 정보를 요구했는지도 모르겠습니다.

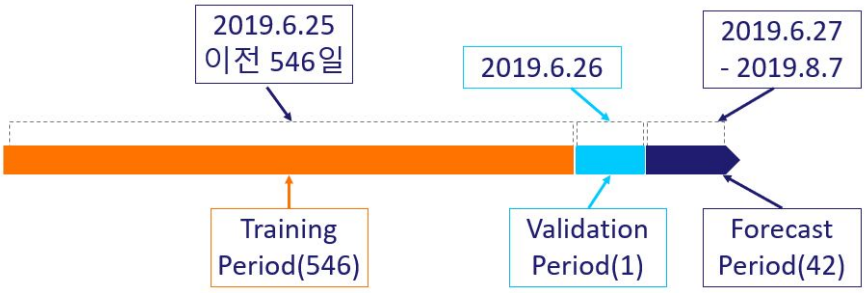
어쨌든 이제 진짜 매출 예측으로 한 번 들어가보겠습니다. 여기까지 썼더니 힘드네요. 잠시 쉬어가겠습니다.



쉬었다 왔습니다. **왼쪽**에 보시는 표는 **구매기록 raw data** 입니다. 해당 구매에 대해 **상품정보, 매장 정보, 가격, 수량 등**이 있습니다. 그 중 **관심있는 정보는 제품, 상권, 색상**이기 때문에 빨간 박스로 표시해 놓은 컬럼들만 뽑아 합쳐 오른쪽 표의 상태로 만듭니다.

오른쪽 표를 보시면 **일자, 항목, 매출**이 정리돼 있습니다. 항목은 매출 **순위_상권+품목+색상(매출비중)** 으로 구성돼 있습니다. 이렇게 구성했더니 총 360가지 항목이 나오는데요. 매출이 고르게 분포돼있지 않아 상위 품목 몇 개를 제외하고는 0에 가까운 것들이 많습니다. 이런 항목은 예측이 잘 안되거니와 돼도 의미가 없기 때문에, 매출 2%이상인 9개 종류에 대해서만 진행했습니다. 같은 날짜가 9개씩 있는 게 보이시나요?

4 Time-Line



다음은 Time-Line 입니다. 그 전에 **train-validation-test set** 에 대해 이해하시면 좋겠는데요. 본래 10000개의 record가 있다고 하면 전체 데이터에 대해 train시키거나 하진 않습니다. 그러면 train set 에 모델이 맞춰져서 새로운 data에 대해 예측해야할 때 정확도가 떨어지는 **overfitting** 문제가 발생하거든요. 그래서 일부인 6000개를 **랜덤하게** 골라 training을 하고, 나머지 2000개로 validation을 하며 새로운 data에 대해 대비를 하고, 마지막 2000개로 test를 하며 정확도를 봅니다. 비율은 꼭 6:2:2가 정답이 아니라 데이터셋에 따라 다르게 합니다. 데이터가 많을수록 training set의 비율을 증가시켜 99.8:0.1:0.1 도 가능하겠지요.

그런데 지금은 time-series data이기 때문에 평소처럼 랜덤하게 데이터를 고르면 안 됩니다. **과거의 데이터를 학습**해 **최근의 데이터로 검증**하고 **미래를 예측**하는 구조가 돼야하기 때문입니다.

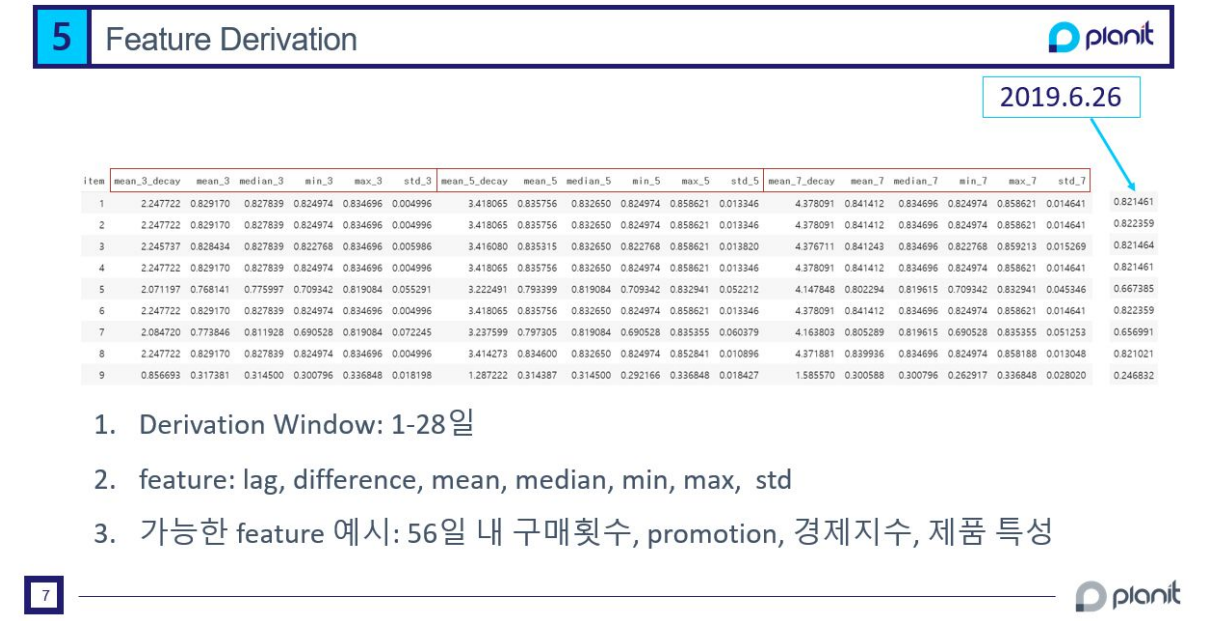
그래서 우선 **Forecast Period** 를 정합니다. 이번에는 42일을 하기로 했죠?

data는 2019.8.7까지인데요. 원래라면 8.8일부터 42일을 예측해야합니다. 하지만 지금은 모델의 성능을 평가해야하기 때문에, 가장 최근의 매출 data인 **6.27 - 8.7 까지 42일의 값을 예측**해보고 **실제 매출과 비교**해 보며 예측 성능이 얼마나 되는지 보겠습니다.

그러면 6.26일 이전의 모든 값을 training 시키면 좋겠지만 Validation Period도 필요합니다. 학습할 때 validation set을 이용해 평가하며 트리를 구성해 나가거든요. 그런데 이게 희안하게 데이터가 적은데도 **validation period**를 짧게 잡을수록 좋더라고요. 그래서 지금은 **1일**로 표시해놨습니다. 아무래도 validation period가 짧을수록 최근 기간을 training 시킬 수 있어 그런가 봅니다.

그러면 이제 2019. 6.25일 이전은 모두 training set으로 쓸 수 있는데요. data가 많으면 좋겠지만 이번에는 최근 제품이라 별로 없었습니다. 또 품목마다 기간이 달라 어떤 제품은 3년치가 있지만 어떤 제품은 1년치밖에 없고 그렇습니다. 상위 2% 품목들은 1년 반 정도 기간이 있어 **Training Period 를 546일(1년 6개월)** 로 잡았습니다. 시계열이니 **7단위**로 떨어지게 설정했습니다. 1년이면 364일을 하는 식으로 말이죠. 현재 9개 품목이니 data의 row는

training set: 546 * 9 = 4914 row
validation set: 1 * 9 = 9 row
test set: 42 * 9 = 378 row
가 되겠습니다.



그냥 날짜별로 매출 값만 넣고 학습하진 않습니다. 뭔가 의미있는 정보를 "이끌어내야" 합니다.

만약 2019.6.26일의 9개 품목 매출을 학습한다고 합시다. 그러면 "언제" 혹은 **"어떤 정보를 가질 때"** 이런 매출이 나오는지 알아야 할 겁니다. 그래야 반대로 예측할 때 "이런 정보를 가졌으니" 매출이 이렇게 될 것이다 예측을 하겠지요. 예를 들어 기상 예측은 한다면 온도, 습도, 기압, 풍량, 강수량, 일조량 등이 한달 동안 이렇게 되면 다음날 비가 오더라 하는 data를 쌓고, 거기에 따라 최근 한 달간 data를 이용해 내일 비 올 확률을 구하는 것처럼요.

예시로 슬라이드의 표를 가져왔습니다. 가장 우측의 한 column은 2019.6.26일의 9개 품목 매출입니다. 이제 왼쪽에는 "어떤 정보를 가질 때" 이 9개 매출이 나오는지 feature를 만든 건데요. 어떤 feature들을 생성시키는지 볼까요?

가장 왼쪽 네모 박스는 2019.6.26일 이전 3일, 그러니까
6.23-6.25 매출의 **mean, median, min, max, std(표준편차), mean_decay** 를 품목마다 구해준 정보입니다.
mean_decay는 6.26일에서 멀어질수록 0.9^n을 곱하며 감소시켜준 feature입니다.
(6.25일 매출) * 0.9,
(6.24일 매출) * 0.9^2,
(6.23일 매출) * 0.9^3
을 평균해준 값입니다. 이렇게 3일간의 정보를 이용하는 거죠. 마찬가지 방식으로 5, 7, 14, 28일의 정보를 이용해 줍니다. 이 일자는 꼭 정해진 건 아니고요. 쓰기 나름입니다.

슬라이드의 1. **derivation window**가 여기서 사용됩니다. 언제까지의 데이터를 feature로 만들어서 학습하겠냐 하는거죠. 지금은 28일로 설정해서 28일 내의 의미있는 일자로 구성합니다. 7, 14, 28일처럼요. 여러 일자를 넣어서 실험을 해보면 좋습니다. 저는 3, 5, 21일을 추가해서 했습니다. **business별로 의미있는 날짜**도 있을 것 같네요.

또 **lag** 값이 있습니다. 이건 그냥 이전 매출 값 자체를 feature로 넣는 건데요.

lag_1: 6.25일 매출,
lag_2: 6.24일 매출,
lag_3: 6.23일 매출

이런 식으로 7일 정도를 feature로 씁니다.

difference도 있습니다. 이전 ARIMA 설명 때 차분에 대해 말씀드렸는데 기억나시나요? 전날과의 차이값을 쓰는 건데요. 6.26일의 차분값은 (6.26일 매출) - (6.25일 매출)이 되는 거죠. 차분한 값을 마찬가지로 7, 14, 28일치 **mean, median, min, max, std, mean_decay**로 만들어 feature로 넣습니다. 이것도 마찬가지로 3일치, 5일치처럼 넣고싶은대로 넣어주면 됩니다. 정확도가 높아질 것 같은 방향을 생각해서요.

또 **item종류, 날짜(연도, 월, 일), 주말**인지 정보도 넣어 봤습니다. 다른 모델 같으면 이 정보들은 categorical variable이라 one-hot vector로 컬럼이 엄청나게 늘어나는데요. 아래 사진의 one-hot encoding처럼요.

음식 메뉴	ID	One-Hot Encoding
참치김치찌개	1	[1, 0, 0, 0, 0, 0]
스팸김치찌개	2	[0, 1, 0, 0, 0, 0]
생고기김치찌개	3	[0, 0, 1, 0, 0, 0]
등심부대찌개	4	[0, 0, 0, 1, 0, 0]
된장찌개	5	[0, 0, 0, 0, 1, 0]
참치초밥	6	[0, 0, 0, 0, 0, 1]

그런데 LGBM은 이걸 ID만 한 컬럼으로 넣을 수 있게 해주고 자동으로 one-hot vector들을 묶어서 처리해주는 알고리즘입니다. 정확도에 거의 손상이 없고 고속으로 처리해줄 수 있다고 하네요.

이렇게 날짜당 한 덩어리의 feature set이 나옵니다. 정리하면,

매출의 3, 5, 7, 14, 28일의 mean, median, min, max, std, mean_decay: 5*6 = **30개**
difference의 3, 5, 7, 14, 28일 mean, median, min, max, std, mean_decay: 5*6 = **30개**
lag_1 - **lag**_7: **7개**
item, year, month, day, weekend: **5개**

총 72개의 column이 생겼네요. 해당 숫자는 예시일 뿐이며 늘리거나 줄일 수 있습니다.

보시다보면 '이런 feature도 있으면 좋겠는데'하고 생각나시는 분들이 계실 것 같습니다. 어떤 것들이 있을까요? 눈을 감고 상상을 한 번 해 보시고 다른 사람들이 어떤 아이디어를 냈는지도 한 번 보시죠.

보통은 구매기록이나 제품정보들이 더 있어서, 날짜별 혹은 제품별로 **promotion 정보**를 넣을 수 있겠습니다. 또 **n일 내 구매횟수**를 넣는 경우도 있고요. **제품 특성**을 넣는 경우도 있습니다. 예를 들어 식자재 같은 경우 얘가 잘 상하는지 같은 거죠. 또 직접적인 영향을 미치는 **경제지수**도 넣을 수 있겠습니다. 주가지수, 소비자물가지수, 석유가격 등이 있겠습니다. 다만 연관된 지수를 잘 찾아야겠죠. 여기서는 소비자물가지수와 경기선행지수를 넣어봤었는데 영향력(feature importance)이 너무 적어 뺐습니다. feature importance는 나중에 설명드리겠습니다.

슬라이드의 숫자가 왜 매출같지 않은지 궁금하신 분도 계실 것 같습니다. 해당 숫자는 매출을 그대로 쓴 것이 아니라, -1에서 1 사이 값으로 scaling 한 결과입니다. 0에서 1 값으로 바꿀 수도 있고, 또는 자연로그만 씌워주거나, median을 빼고 std로 나눠주거나 하는 다양한 방법들이 있습니다. scaling에 따라 빠르고 정확하게 결과를 주기도 합니다. 로그화의 경우 우상향하는 데이터의 경우 효과를 발휘하기도 한답니다. 어떤 것이 가장 좋을지 모르기 때문에 매출값 그대로 넣는 것과 scaling 버전에 따라 두 세가지를 다 돌려봅니다.

feature를 만드는 것에 대한 설명이 길었는데요. 모델을 적용할 때 case마다 신경써야해서 이 부분은 새겨두시면 좋겠습니다.

이번에 살펴볼 내용은 **Forecast Distance**입니다. 이름에서부터 느껴지시겠지만 얼마까지의 기간을 예측하느냐인데요. 하지만 조금 더 내용이 있으니 건너뛰진 마세요.

42일을 어떻게 예측하면 좋을까요? 한 번에 42일치가 나오면 좋겠지만 생각처럼 쉽지 않습니다.
어떻게 설명드리나 고민하다 예시로 표를 그리기로 했습니다.

training set X	Y(매출)
feature set_1 (9줄)	2018.1.1 (9줄)
feature set_2 (9줄)	2018.1.2 (9줄)
⋮	⋮
⋮	⋮
feature set_546 (9줄)	2019.6.26 (9줄)

이게 현재 training set입니다. 위에서 말씀드린 것처럼 **날짜별로 feature를 생성한 후 붙인**거죠.

이제 2019.6.27 을 예측하기 위해 6.26일까지의 data로 마찬가지 feature set를 생성하지만

feature set_547 (9줄)	없음(6.27 예측)

결과값은 없이(예측해야하니까) feature set만 가집니다. 이러면 이전 데이터로 학습하고 6월 27일 값을 예측할 수 있습니다. 지금은 LGBM이니, 이전 데이터들로 숲과 같은 여러 개의 트리 구조를 만들고 마지막 feature set을 넣었을 때 9개 품목 한 줄 한 줄 예측값을 분류시켜주는 겁니다. 마지막에 9줄의 데이터를 넣고 9개의 예측값을 얻는 거죠.

그러면 2일 후인 6.28일은 어떻게 예측할까요?
눈을 감고 생각을 해보면서 역시 세상엔 대단한 사람이 많다는 것을 다시 한번 느껴 봅니다.

training set을 다음과 같이 만들어주면 됩니다.

feature set_1 (9줄)	2018.1.2 (9줄)
⋮	2018.1.3 (9줄)
⋮	⋮
⋮	⋮
feature set_545 (9줄)	2019.6.26 (9줄)

feature set 숫자와 날짜가 바뀐 것을 주목해주세요. 같은 feature set_1 에 대응하는 y 값이 2018.1.1에서 2018.1.2로 바꼈습니다. 마찬가지로 예측은

feature set_546 (9줄)	없음(6.27 예측)

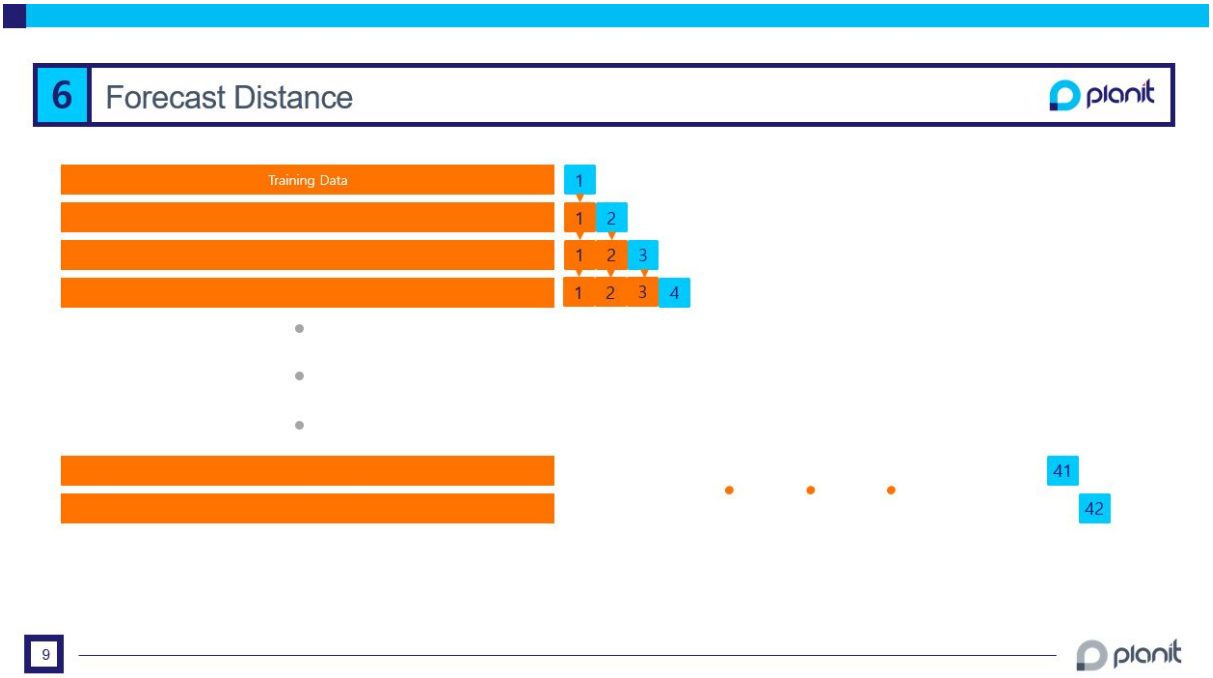
feature set_546으로 하면 6.27일을 예측할 수 있겠죠?

feature set_547 (9줄)	없음(6.28 예측)

feature set_547을 넣으면 6.28일을 예측할 수 있습니다.

이런 방식을 Forecast Distance가 2라고 합니다.

이렇게 42일까지 Foreacst Distance를 늘려주면 42일 후를 예측합니다. 그런데 여기서 한 단계 더 발전한 신박한 방법이 있습니다.



제가 그림으로 한 번 표현을 해봤는데요. 어떻게, 감이 오실 지 모르겠습니다.

Forecast Distance(FD)는 멀어질수록 예측 정확도가 떨어집니다. 보통은, 또한 상식적으로 FD가 1일 때 정확도가 높을 겁니다. **FD가 1을 유지하면서 42일 전체를 예측**하면 좋겠습니다. 그걸 또 누군가가 해냈네요.

바로 **예측값을 다시 training set에 넣어** FD를 1로 유지하면서 42일을 예측해나가는 방법입니다. 그림이 나쁘지 않죠? 이게 가능하다면 FD를 2,3,4,..., 42까지 나눠서 모두 결과를 얻을 수 있겠습니다. 그러고 결과가 가장 좋은 FD를 선택하는 거죠.

이번에는 이미 결과를 알고 있어서 FD가 1인 경우만 만들어봤습니다. 하지만 case에 따라 최적 FD가 3, 7, 12, 19 이렇게 다양하니 결국 해봐야 안다고 하겠습니다.

7 Evaluation

	2019-07-31 00:00:00	avg_35sum	max_35	min_35	avg_35rev	avg_rate	max_rate	min_rate	RMSE
001_면세+M17FCP77+50B(26.21%)	-2.11441e+08	-2.337872e+08	-2.114412e+08	-2.745164e+08	1.598712e+09	-0.146235	-0.132257	-0.171711	1.329803e+07
002_면세+M17FCP77+50L(11.61%)	-6.09671e+07	-1.129782e+08	-6.096714e+07	-1.429023e+08	7.080700e+08	-0.159558	-0.086103	-0.201819	9.133004e+06
003_면세+M17FCP77+07U(7.31%)	-3.8819e+07	-5.868414e+07	-3.881903e+07	-7.360188e+07	4.460775e+08	-0.131556	-0.087023	-0.164998	4.094675e+06
004_면세+M17FCP77+50W(5.15%)	1.55367e+07	7.211860e+06	1.556327e+07	-2.673896e+06	3.138850e+08	0.022976	0.049583	-0.008519	2.619614e+06
005_면세+M17FCP77+07L(4.88%)	-4.43891e+07	-5.246497e+07	-4.438909e+07	-6.013417e+07	2.975500e+08	-0.176323	-0.149182	-0.202098	2.786649e+06
006_면세+M17FCP77+50N(4.07%)	-8.92035e+06	-8.560813e+06	-7.866287e+05	-1.435053e+07	2.479917e+08	-0.034521	-0.003172	-0.057867	2.345284e+06
007_면세+M16FCP07+50L(3.79%)	2.48957e+07	1.608032e+07	2.489572e+07	1.090847e+07	2.310000e+08	0.069612	0.107774	0.047223	2.736619e+06
008_면세+M17FCP77+07W(2.36%)	1.48517e+07	1.165423e+07	1.486655e+07	8.979344e+06	1.438250e+08	0.081031	0.103366	0.062432	1.440200e+06
009_국내+M17FCP77+50L(2.3%)	-2.24111e+07	-3.046878e+07	-2.241114e+07	-3.826770e+07	1.404616e+08	-0.216919	-0.159554	-0.272443	1.814396e+06

+ MASE

예측값을 구한 이후엔 평가를 해야할 겁니다. 여러 가지 지표를 쓸 수 있겠지만 여기서는 RMSE와 MASE를 소개하겠습니다.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

웍!

수식으로 한 번 놀래켜 드렸습니다. 꺽쇠가 붙어있는 y는 예측값입니다. **실제값 - 예측값을 제곱해서 더하고 평균한 뒤 루트** 씌웠습니다. 절대적인 값이 의미를 갖는다기보단 모델간 비교할 때 씁니다.

$$\mathrm{MASE} = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{|e_t|}{\frac{1}{T-1}\sum_{t=2}^{T}|Y_t - Y_{t-1}|}\right) = \frac{\sum_{t=1}^{T}|e_t|}{\frac{T}{T-1}\sum_{t=2}^{T}|Y_t - Y_{t-1}|}$$

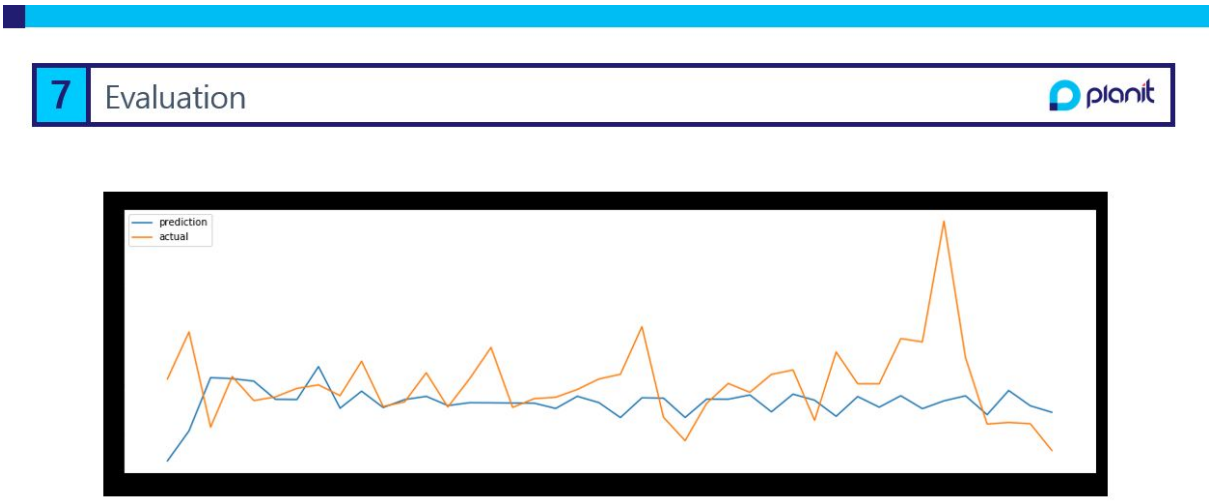
웍!

한 번 더 놀래켜 드렸습니다. MASE는 **과거 42일 매출을 그대로 예측값으로 썼을 때보다 얼마나 더 나아졌나를 비율**로 나타냅니다. 나아진 게 없으면 1이겠고요. 보통 더 나아져서 0.9, 0.8 이런 식으로 표현합니다. 마찬가지 절대적 값이 의미를 갖진 않고 모델간, 실험간 비교를 위해 쓸 수 있겠습니다.

또 이번에는 35일 잔차 합 8개를 본다고 말씀드렸는데요. 슬라이드 빨간 박스 순서대로 첫 **35일 잔차합, 35일 잔차합 8개의 mean, max, min**를 구했습니다. 또 avg_35rev는 해당

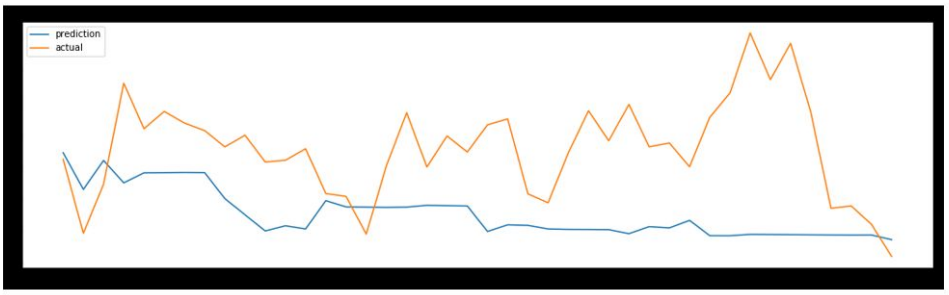
품목의 매출 35일 평균이고요. 그와 비교한 잔차합을 비율로 표시해 rate를 구했습니다. 다시 말해, **매출 대비 잔차합이 얼마나 큰가**를 나타내 줍니다.

하지만 지표만으로는 성능을 평가하기 어렵습니다. 실제값과 예측값의 그래프를 보고 판단하기도 해야합니다.



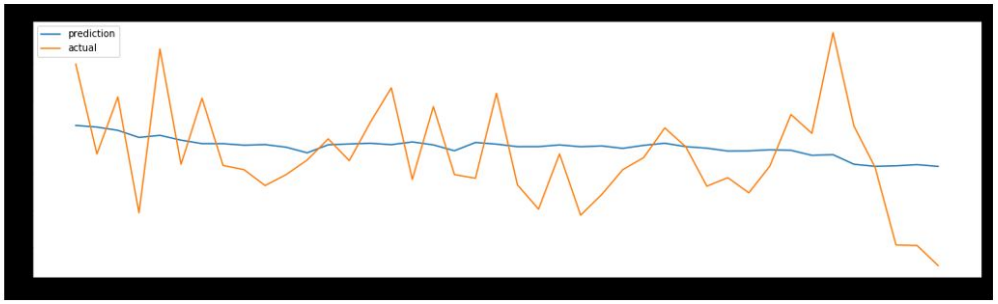
푸른색이 예측값, **주황색이 실제값**입니다. 저는 푸른색과 주황색을 좋아합니다.
오른쪽에 보시다시피 이렇게 튀어나오는 부분은 예측이 잘 안 됩니다. 이번 F&F는 특성상 매장에서 매출을 맞추기 위해 몰아찍는 경우, 중국 도매상의 불규칙적 출현으로 대량 구매가 발생하는 경우 등 예측 불가능한 시점에 매출이 뛰는 경우가 있어서 이런 경우는 정확도가 떨어집니다.

7 Evaluation



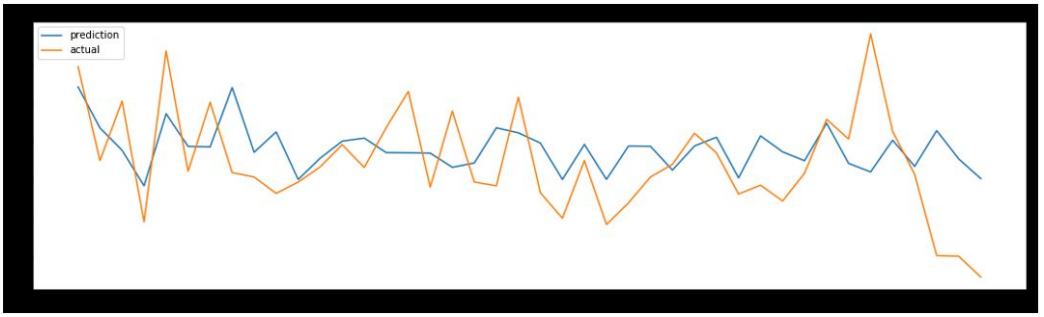
다양한 경우를 시도하다 보면 이렇게 이상하게 나올 때가 많습니다.

7 Evaluation



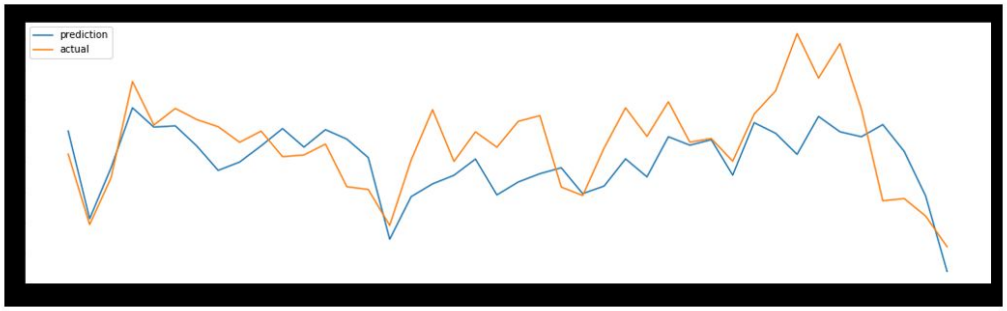
이런 경우는 평균이 맞아서 35일 잔차합 지표상으로는 훌륭하지만 매우 아쉽습니다.
RMSE나 MASE도 안 좋게 나올 겁니다. 좀 더 섬세하게, dynamic 한 선을 그리고 싶습니다.

7 Evaluation

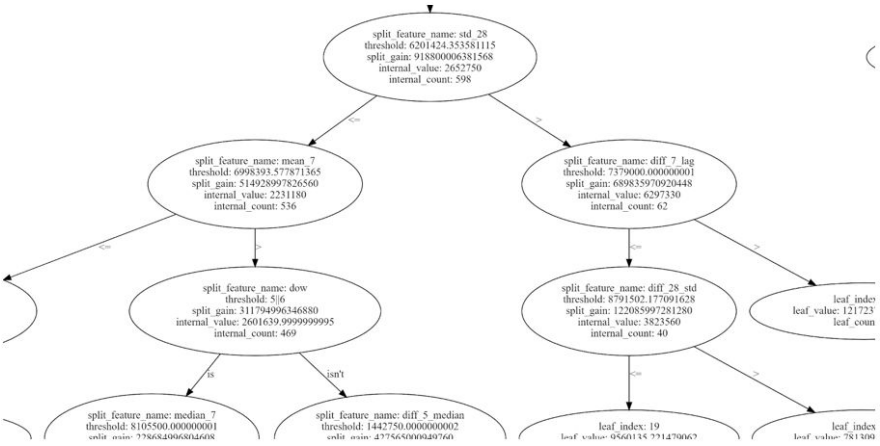


feature들, 품목 수, training&validation 기간, hyperparameter들을 조정해가며 여러 test를 하는데요. hyperparameter 중 하나인 learning rate를 올려봤더니 좀 더 dynamic한 선이 나옵니다.

7 Evaluation



이것도 아주 좋은 축에 속합니다. 오른쪽에서 한번 치고 올라갔으면 하지만 끝에 꺾이는 것까지 보기 좋습니다. 어떻게 하면 정확도가 더 좋아질까 고민하다보면 LGBM 자체에 대한 이해도가 필요합니다. 그래서 알고리즘이 만드는 트리 구조를 한 번 보겠습니다.



이것은 생성하는 **수많은 트리 중 하나**의 모습입니다. 이걸 트리라고 하다니 수백년간 인간을 지켜본 선정릉의 나무가 창밖에서 비웃는 듯합니다. 부끄러워 초록색이라도 입혀볼까 했지만 더이상 저의 색감을 뽐내지 않기로 디자인 스쿨을 졸업할 때 맹세했습니다.

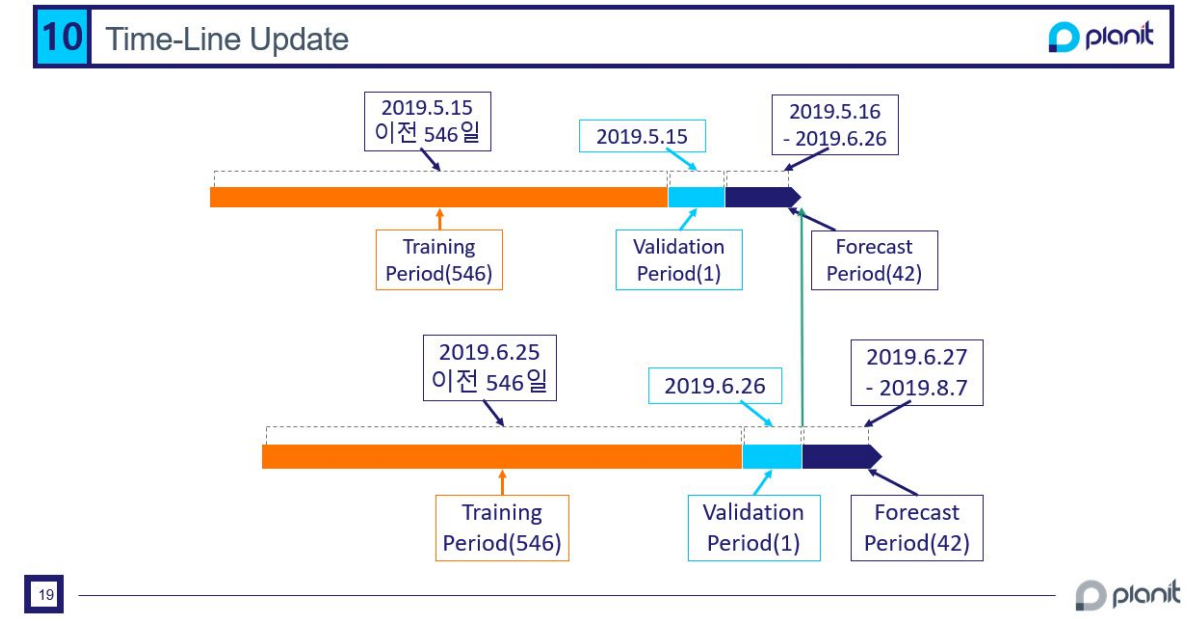
어떤 feature로 트리가 나뉘는지, 기준이 되는 값, 분할로 얻는 이득 등이 나옵니다. 이를 바탕으로 숲이 다 만들어진 후 **feature importance**가 계산되고, 그 결과를 보고 **feature selection**도 우리가 직접 할 수 있습니다. feature importance가 낮은 feature는 제외시키고 다시 돌리면 정확도가 올라갑니다. 모습을 봤으니 구조를 생각하면서 어떻게 트리를 만들도록 옵션을 조정하면 좋을까 고민해야 합니다. 그 옵션 조정이 hyperparameter tuning입니다.

9 Hyperparameters

```
best_params = {
    'boost_from_average': True,
    'boosting':'gbdt',
    'Decay': 'linear',
    'n_estimators': 5000,
    'early_stopping_rounds':200,
    'objective': 'regression_l2',
    'metric': 'l2',
    'num_threads': 16,
    'max_depth': None,
    'min_sum_hessian_in_leaf': 0,
    'min_data_in_leaf': 20,
    'feature_fraction': 0.3,
    'bagging_fraction': 0.8,
    'bagging_freq': 1,
    'lambda_l1': 0,
    'lambda_l2': 0,
    'learning_rate': 0.5,
    'max_bin': 255,
    'num_leaves': 120 }
```


LGBM에서는 hyperparameter가 거짓말이 아니라 100개 가까이 되는 것 같습니다. 세 보려고 했는데 쎄 보여서 그만뒀습니다. 라임 굿? 그걸 다 조정할 수는 없고 몇 개만 골라서 조정을 해 봤습니다. 여기서는 몇 개만 설명드리겠지만 궁금하신 분들은 LGBM documentation https://lightgbm.readthedocs.io/en/latest/Parameters.html 참고하시기 바랍니다.

n_estimators: 트리를 몇 개나 생성할까.
early_stopping_rounds: 평가 metric이 일정 수준 이상 나아지지 않는다면 거기서 멈춘다.
max_depth: 트리의 최대 깊이를 제한.
min_data_in_leaf: 맨 마지막 잎의 데이터 개수 최소값.
feature_fraction: 트리 하나 생성 시 전체 feature 중 얼마를 쓸 것인가.(random pick+seed)
bagging_fraction: 트리 하나 생성 시 전체 데이터 중 얼마를 쓸 것인가.(random pick+seed)
bagging_freq: 몇 번째 트리마다 bagging을 할 것인가.
num_leaves: 트리당 최대 잎을 얼마나 할 것인지.



튜닝을 하다보니 좋은 파라미터를 찾으면 정확도가 올라간다는 걸 알았습니다. 이제 실제 사용자의 입장에서 생각해 보겠습니다.

지금은 예측값을 실제값과 비교할 수 있었고 그 결과에 맞춰 hyperparameter를 조정했습니다. 하지만 **미래를 예측하는 입장에서 예측값을 실제값과 비교해볼 수 없습니다. 실제값이 아직 없으니까요.** 그러면 어떻게 hyperparameter를 설정할 수 있을까요? 더 나아가서 어떤 feature를 쓰고 버릴지, training & validation 기간은 얼마큼이 좋을지 어떻게 알 수 있을까요? 바로 **이전 기간 데이터로 예측 모델을 돌리고 그 중 결과가 좋은 feature & hyperparameter 세팅으로 원하는 기간에 대한 예측**을 하는 겁니다.

슬라이드의 아래 그림은 위의 타임라인에서 보셨던 그림입니다. 그리고 그 위는 hyperparameter를 찾기 위한 **backtest**의 타임라인입니다. 그림이 복잡해 보이지만 **초기 타임라인을 전체 42일 앞당긴 것** 뿐입니다.

예측기간인 42일(2019.5.16-2019.6.26)을 먼저 설정하고 그 이전 하루(2019.5.15)를 validation, 다시 이전 546일을 training 기간으로 삼습니다.

hyperparameter에서 시도해 볼 경우의 수가 너무 많기 때문에 일일이 다 해볼 수가 없습니다. 그래서 **bayesian optimization**이라는 방법을 써서 **hyperparameter마다 일정 범위를 지정해주고 최적값을 자동으로 찾아주는** 라이브러리가 있어 사용했습니다. 하지만 정확도가 아직 공식적인 인정을 받은 수준의 느낌은 아닙니다.

추가로, **validation period를 training set에 포함시키면 훨씬 정확도가 올라가는 것을 발견**했습니다. 데이터가 적기 때문일수도 있지만, validation period가 최근이기 때문에 최신 데이터에 overfitting 하는 느낌으로 받아들일 수 있겠습니다.

여기까지 LGBM을 이용한 시계열 예측 모델에 대해 알아봤습니다.