서울시 자치구별 업종별 매출 예측 시스템

LSTM 기반 시계열 딥러닝 분석

2026년 1월

목차

프로젝트 파이프라인

1. 데이터 수집

- 서울 열린 데이터 광장
- 25개 자치구 데이터
- 60개 업종 분류

2. 전처리

- 제곱근 변환 (Sqrt)
- Standard Scaler
- Winsorization 처리

3. 모델 학습

- LSTM-2Layer 구조
- Adam 옵티마이저
- Dropout / BatchNorm

4. 예측/결과

- 다음 분기 매출 예측
- 구/업종별 상세 분석
- 비선형 패턴 학습

연구 배경 및 목적

🔥 연구 배경

- COVID-19 이후 2023년 자영업 폐업 100만명 육박 뉴스
- 정부 - 매출 감소 지역* 업종을 예측하여 정부 보조금 지원 계획 수립에 도움
- 가계 및 기업 - 창업 시기와 업종 선택, 프렌차이즈 출점 시기와 재고 전략에 도움

🎯 연구 목적

- 시계열 딥러닝 모델을 활용한 매출 예측
- 자치구별 × 업종별 상세 분석

방법론: 문제 정의 및 모델 선정

🎯 시계열 예측 문제 -> 시계열 예측 모델 고려

- 목표: 분기별 매출액 예측
- 입력: 과거 8분기 데이터
- 출력: 다음 1분기 매출

🎯 히트맵 분석 - 비선형 모형

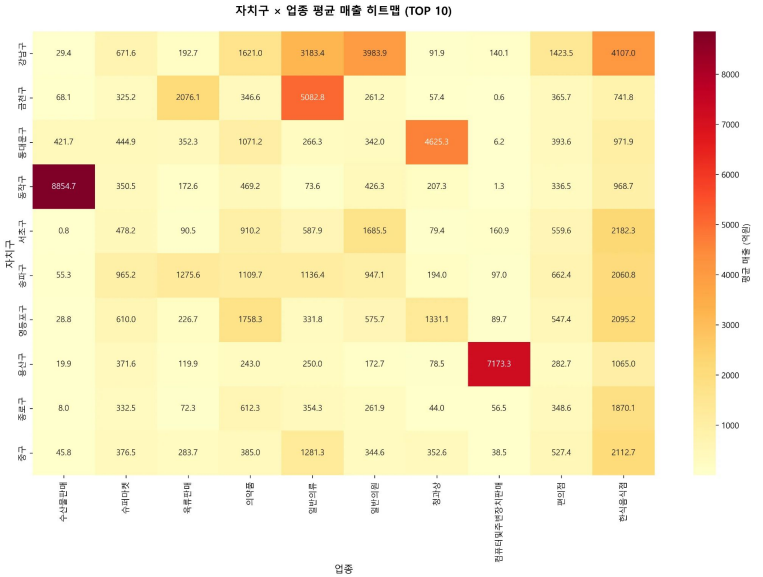
강남구-다양한 업종 고매출

동작구- 수산물 업종 고매출

용산구 - 컴퓨터 업종 고매출

지역 + 업종 관계 비선형 - > LSTM 비선형 학습성에 주목

히트맵 분석



🔬 모델 선정 근거 -모델 비교

LSTM 선택 : 시계열 + 비선형

GRU : 단순 LSTM 모형(복잡한 모형이 필요한가?)

RF : 비선형

데이터 개요 및 전처리

📊 데이터 출처

출처: 서울 열린데이터 광장

API: VwsmSignguSelngW

기간: 2019 Q1 ~ 2025 Q3

📈 데이터 규모

자치구: 25개

업종: 60 개

분기 :27분기

총 데이터: 40,000+ 행

🔧 전처리

• 결측치 제거 - 40,000 샘플 중 결측치 소수여서, 제거해도 학습데이터로 충분하다고 판단

• 이상치 처리 (Winsorization)

-> 시퀀스 유지, 데이터 손실X

, 극단값 영향 완화, 시계열 연속

```
매출 분포:
Q1 (25%): 50억
Q2 (50%): 200억  (중앙값)
Q3 (75%): 500억
최대값: 8,000억  ← 극단값! ⚠
# 8,000억은 평균(300억)의 27배!
```

LSTM이 8,000억에 집중하여
패턴 학습 방해 -> 모델 왜곡 가능성

단순 삭제 -> 데이터 손실, 정보 왜곡

방법론: 데이터 전처리

🔄 8분기 시퀀스 선택 근거

선택 이유

1. 계절 패턴 학습-4분기 안한 이유

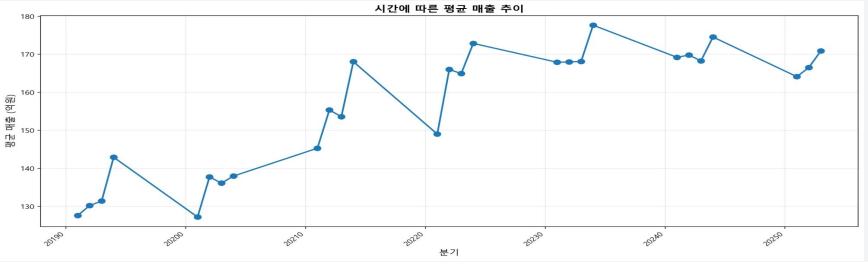
- 4분기(1년): 1번 관찰 → 우연 가능
- **8분기(2년): 2번 반복 → 패턴 확인 ✅**
- 예: '매년 Q2,4 상승, 1분기 하락' 반복 학습

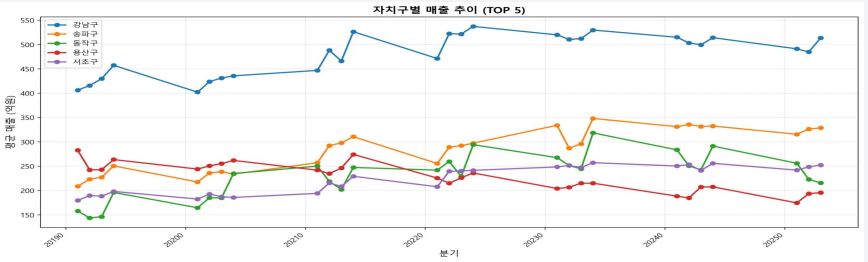
2. LSTM 최적 길이

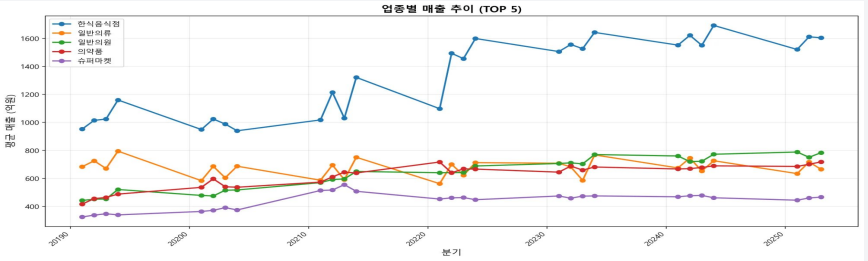
- 일반적으로 **8~12 timesteps**가 최적

3. 데이터 효율성-12분기 안한 이유

- 12분기: 최소 13분기 데이터 필요
- 현재 데이터: 30%가 10분기 이하
- 8분기가 성능과 효율의 균형점







방법론: 데이터 전처리

데이터 변환(제곱근 변환) 선택

📊 전체 순위 (MAE 기준)- 오차는 낮을 수록 좋음

1. 제곱근 (Sqrt) (standard): MAE 12.97억, R² 0.9855

2. 로그 (Log10) (standard): MAE 13.35억, R² 0.9852
3. 원본 (Raw) (standard): MAE 14.71억, R² 0.9847
4. 원본 (Raw) (minmax): MAE 20.81억, R² 0.9818

구간별 성능 비교 (MAE) - 오차는 낮을 수록 좋음

저매출 구간 (100억 미만):

*제곱근: 4.57억 ✅ > 로그: 4.89억 > 원본: 5.23억

중매출 구간 (100~1000억):

* 제곱근: 30.22억 ✅ > 로그: 32.15억 > 원본: 35.67억

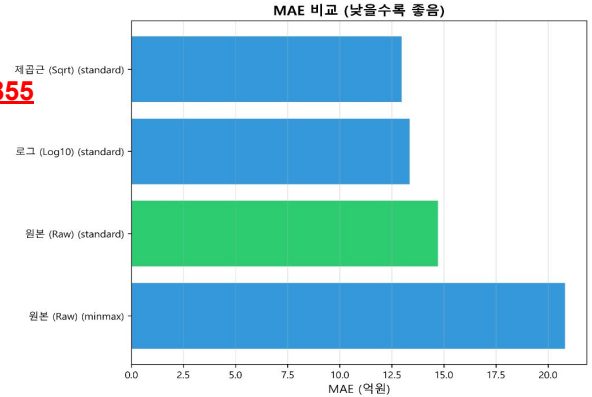
고매출 구간 (1000억 이상):

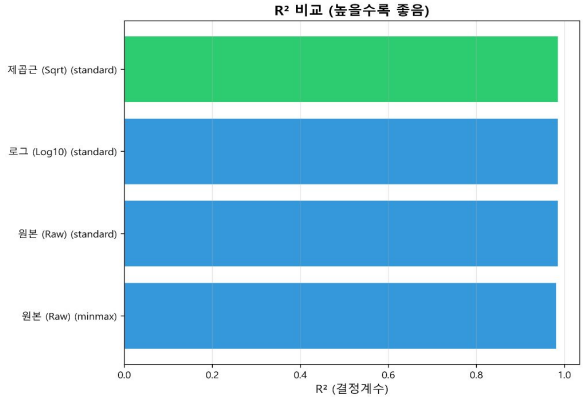
* 제곱근: 83.48억 ✅ > 로그: 91.23억 > 원본: 102.45억

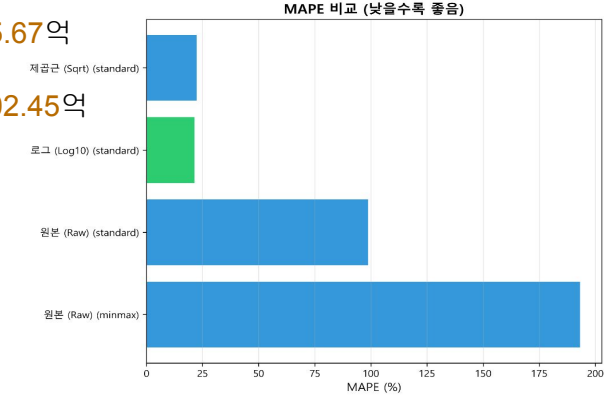
제곱근은 로그에 비해 적당한 압축 강도

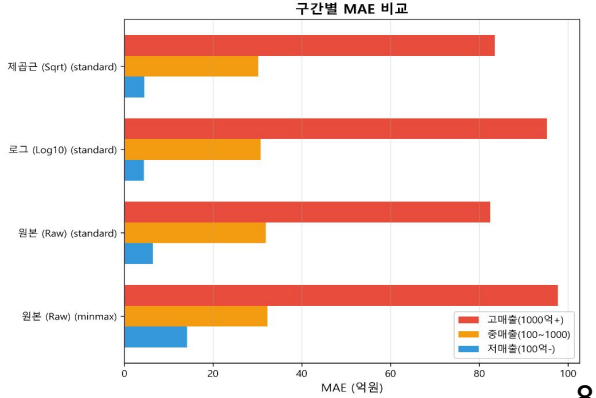
*로그는 100억과 1000억의 10배 차이를,
1 차이로 과도하게 압축합니다

*제곱근은 이 10배 차이를 약 3배 차이로,
적당히 압축하여큰 매출 정보를 더 잘 보존









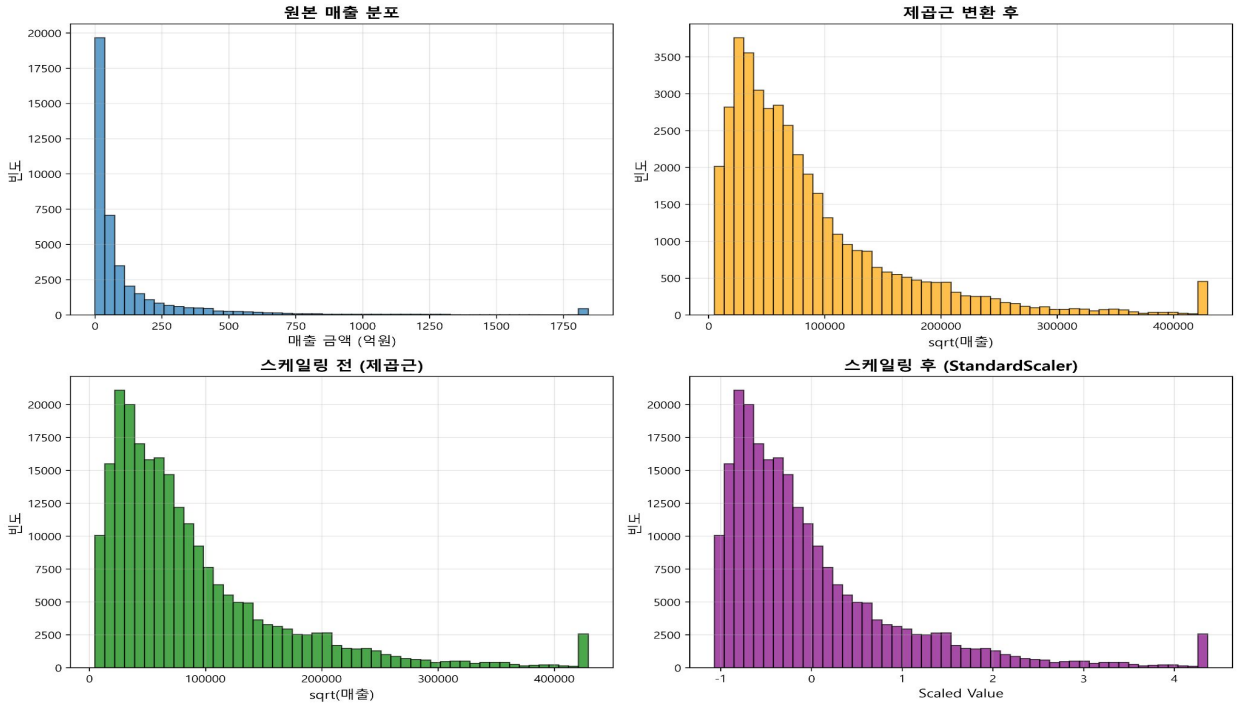
방법론: 데이터 전처리

스케일러 (StandardScaler) 선택

우리 데이터 특성
1. 매출 분포: 1억~수천억으로 극단값 존재
2. 대부분 100~500억, 소수가 1000억 이상 3. MinMaxScaler는 이런 극단값에 민감

LSTM 학습 안정성
1. LSTM의 sigmoid/tanh 활성화 함수
-> 입력값 0 근처에서 gradient 안정적
2..StandardScaler
-> 데이터를 평균 0, 분산 1로 변환하여 LSTM조건에 만족

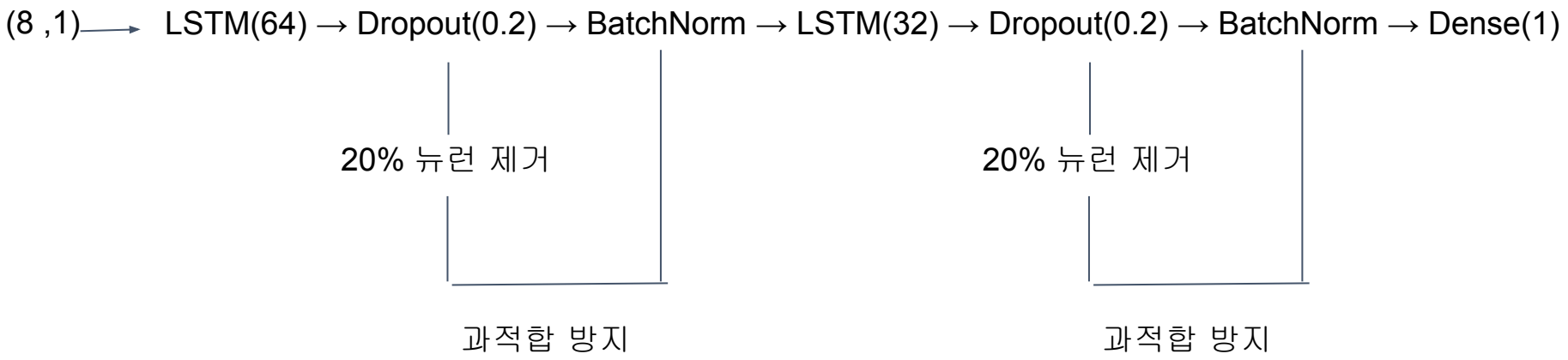
실험 결과 (앞 페이지)
Raw + StandardScaler: MAE 14.71억 ✅
Raw + MinMaxScaler: MAE 20.81억 ❌ -
차이: 41.5% 성능 차이!



제곱근 적용 -> 그래프 대칭적으로 변화 (우상 그래프)

Standard 적용 -> 정규화 (우하 그래프)

LSTM 아키텍처 설계



*입력 (8, 1) 과거 8분기, 다음 1분기 매출 예
*손실 함수 MSE
*옵티마이저 Adam

```
model = keras.Sequential([
    layers.LSTM(64, return_sequences=True, input_shape=(8, 1), name='lstm_1'),
    layers.Dropout(0.2),
    layers.BatchNormalization(),

    layers.LSTM(32, return_sequences=False, name='lstm_2'),
    layers.Dropout(0.2),
    layers.BatchNormalization(),

    layers.Dense(1, name='output')
], name='LSTM_Model')
```

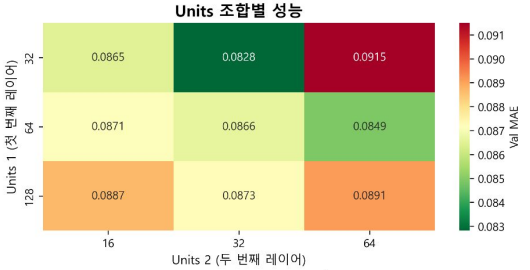
하이퍼파라미터 전략

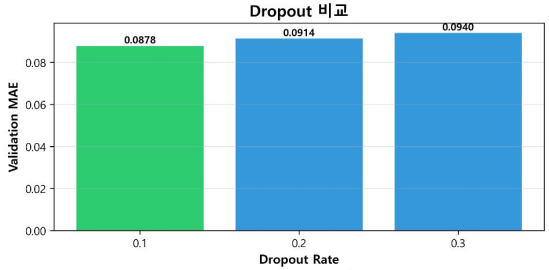
1. Optimizer -> Adam(뒷 페이지)
2. 순차 튜닝 전략

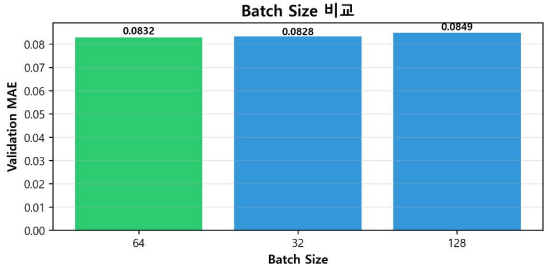
units에서 mae 최저값으로 선택
-> units 고정값 + dropout 각 경우 비교
-> mae 최저치 dropout 선택(고정)
-> units고정값+dropout고정값+ batch size 각 경우 비교하여 최저 mae 나오는 batch size 선택
-> units고정값+dropout고정값+ batch size고정값+learning rate 각 경우 비교
-> 최저 mae 나오는 learning rate 선택
243가지 실험을 18가지 실험으로 축소 (90%시간절약)

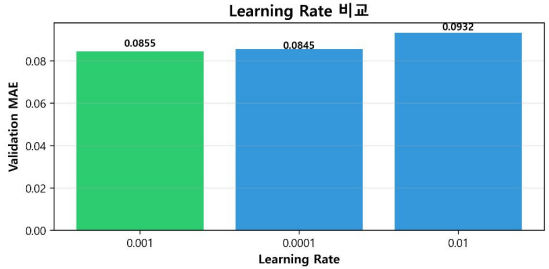
3. 결론 : 최적 하이퍼 파라미터

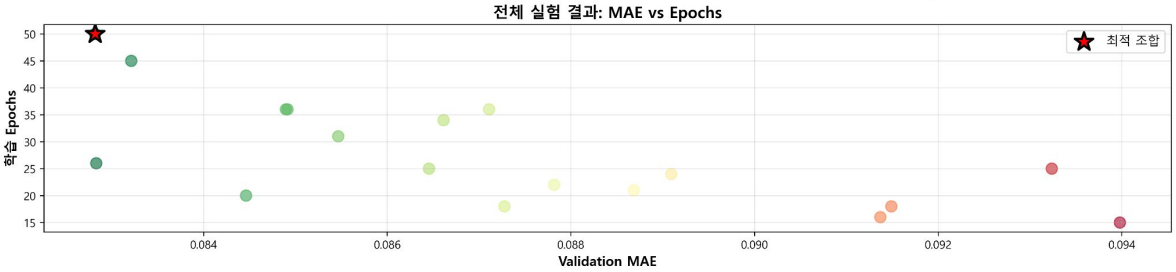
Units : 64 -> 32
Dropout : 0.2
Batch size : 64
Learning Rate: 0.001











실험 설계- 옵티마이저 비교 실험

Adam vs SGD vs RMSprop

```
model = keras.Sequential([
    layers.LSTM(64, return_sequences=True, input_shape=(8, 1)),
    layers.Dropout(0.2),
    layers.BatchNormalization(),

    layers.LSTM(32, return_sequences=False),
    layers.Dropout(0.2),
    layers.BatchNormalization(),

    layers.Dense(1)
])

# 옵티마이저 선택
if optimizer_name == 'adam':
    optimizer = Adam(learning_rate=learning_rate)
elif optimizer_name == 'sgd':
    optimizer = SGD(learning_rate=learning_rate, momentum=0.9, nesterov=True)
elif optimizer_name == 'rmsprop':
    optimizer = RMSprop(learning_rate=learning_rate)
else:
    raise ValueError(f"Unknown optimizer: {optimizer_name}")

model.compile(
    optimizer=optimizer,
    loss='mse',
    metrics=['mae']
)

return model
```

```
# 콜백 설정
early_stop = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=15, # ← 15 epoch 동안 개선 없으면 종료
    restore_best_weights=True,
    verbose=0
)
```

실험 조건

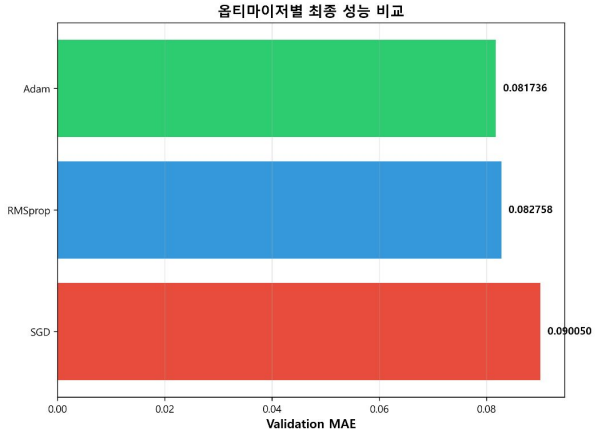
동일 모델 적용: LSTM(64) → Dropout → BatchNorm → LSTM(32) → Dropout → BatchNorm → Dense(1)

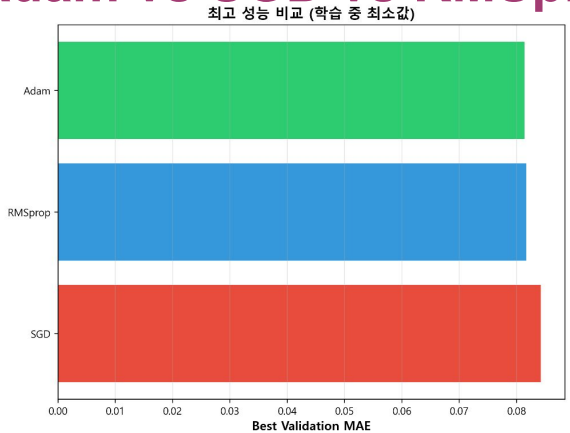
평가 : Validation MAE

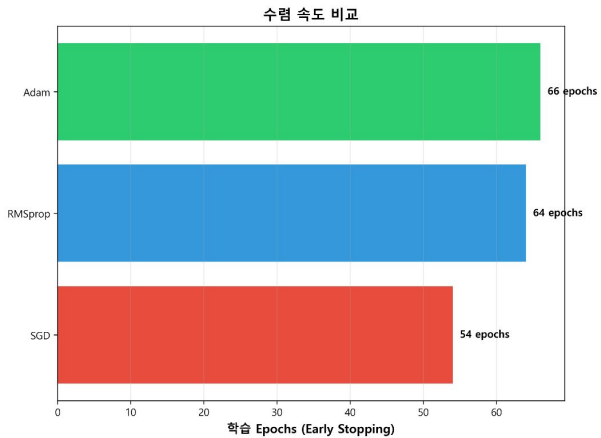
공정성: 동일 구조, Early Stopping 적용

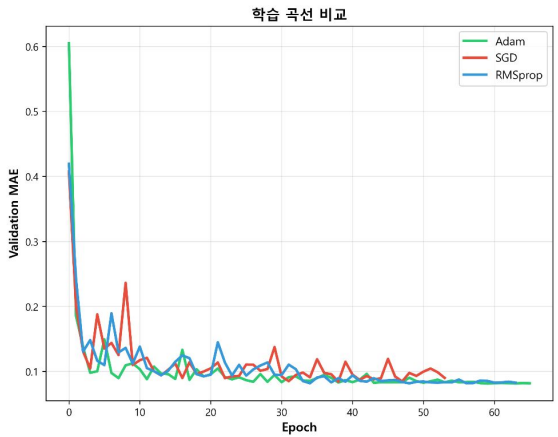
실험 설계

Adam vs SGD vs RMSprop









Validation MAE : ADAM 가장 우수
(SGD와 10.17% 차이)

수렴 속도 :

1. ADAM = 좋은 수렴

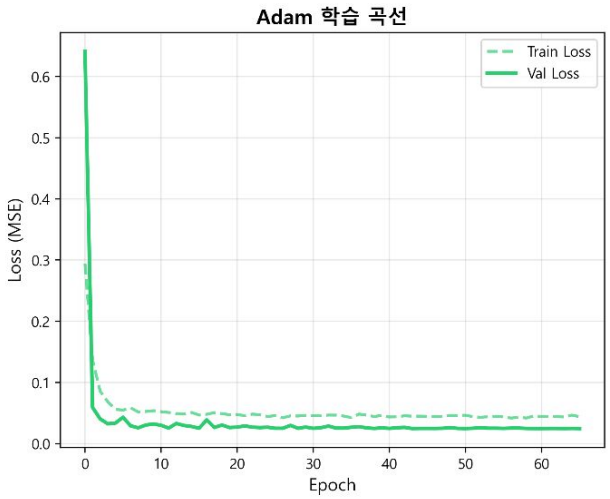
-> 최적 상태에 가장 늦게 도착했지만, MAE는 가장 좋으므로 성능이 우수하다.

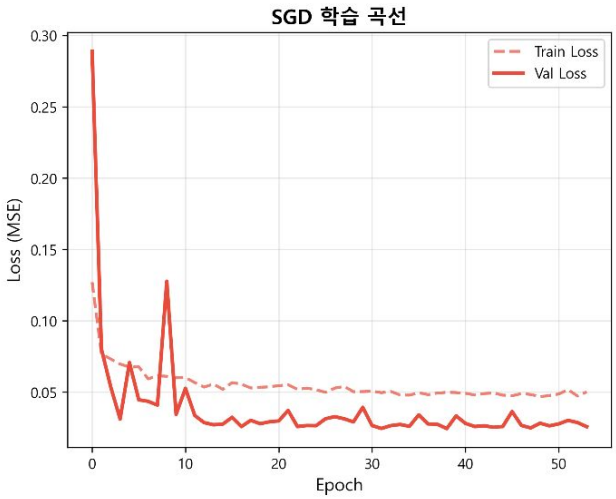
2. SGD = 나쁜 수렴

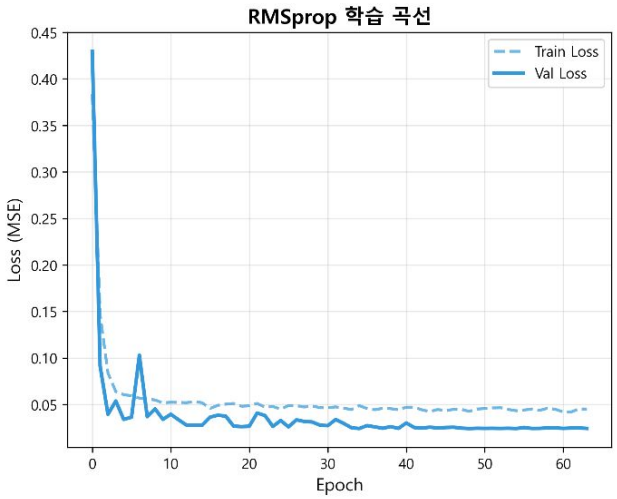
-> 최적 상태에 가장 일찍 도달 했지만 MAE는 가장 안 좋아서, 성능이 떨어진다고 판단하였다.

실험 설계

Adam vs SGD vs RMSprop







ADAM : 변동성 낮음 -> 안정적 -> 적응형 학습률 : 학습 진행될 수록 학습률 자동으로 줄여, 최적점에 안착

train, val loss 간격 일정, train이 val보다 약간 낮음 -> 학습이 잘되어 일반화에 성공

SGD : 변동성 높음 -> 불안정적 -> 고정 학습률: 학습이 진행되도 학습률이 그대로라, 최적점 주변 배회

과적합 방지 기법

1. Dropout(0.2)

학습 시 20% 뉴런 무작위 제거 ->특정 뉴런 의존도 감소 -> 앙상블 효과

2. BatchNormalization

각 배치마다 정규화 감소 막음 -> 학습 안정화

3. EarlyStopping

Validation loss 모니터링 -> 15epochs 동안 개선 없으면, 학습 멈춤 -> 가장 성능 좋았던 시점의 가중치로 되돌립니다.

4. ReduceLROnPlateau

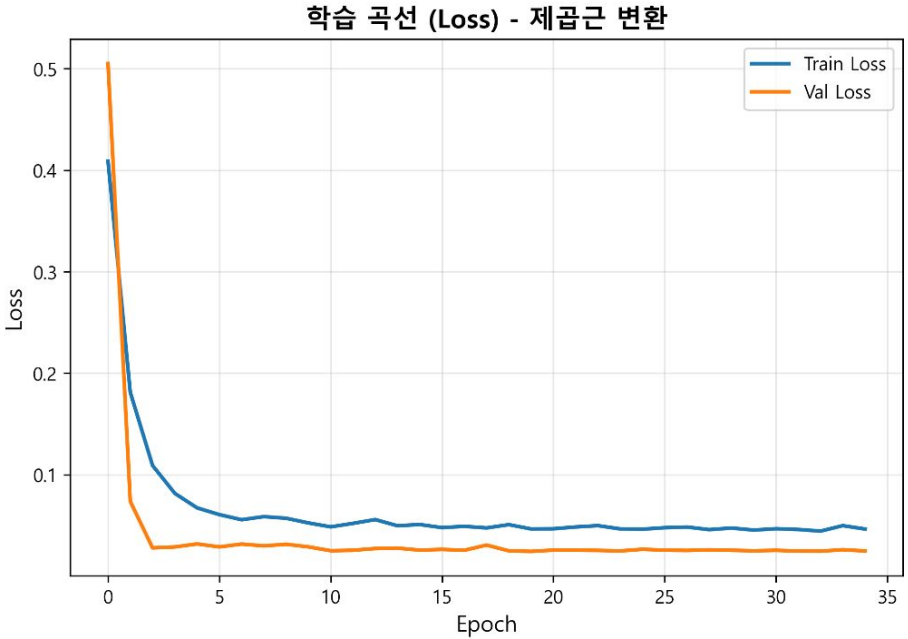
Validation loss를 모니터링 -> 7epoch 동안 개선 없으면, Learning rate를 절반으로 줄여서, 학습이 정체되었을 때, 더 세밀하게 탐색

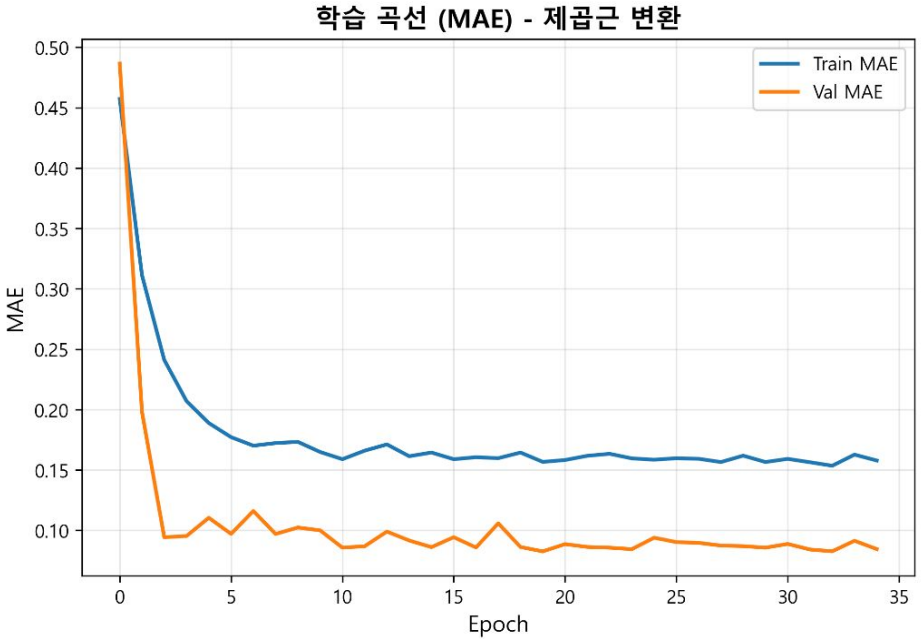
=> Train과 Validation 성능 격차 작게 만듭니다.

```
early_stop = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=15,
    restore_best_weights=True,
    verbose=0
)

reduce_lr = keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=7,
    min_lr=1e-6,
    verbose=0
)
```

학습 곡선 및 수렴 분석-LSTM





초반 5 epoch -> Train, Val의 급격한 하락 = 모델이 데이터 패턴을 빠르게 학습하고 있는 증거
5 epoch 이후 -> 두 그래프 수평 = 최적 성능에 도달

학습 35 epoch에서 종료 -> Early Stopping 멈춘것입니다.

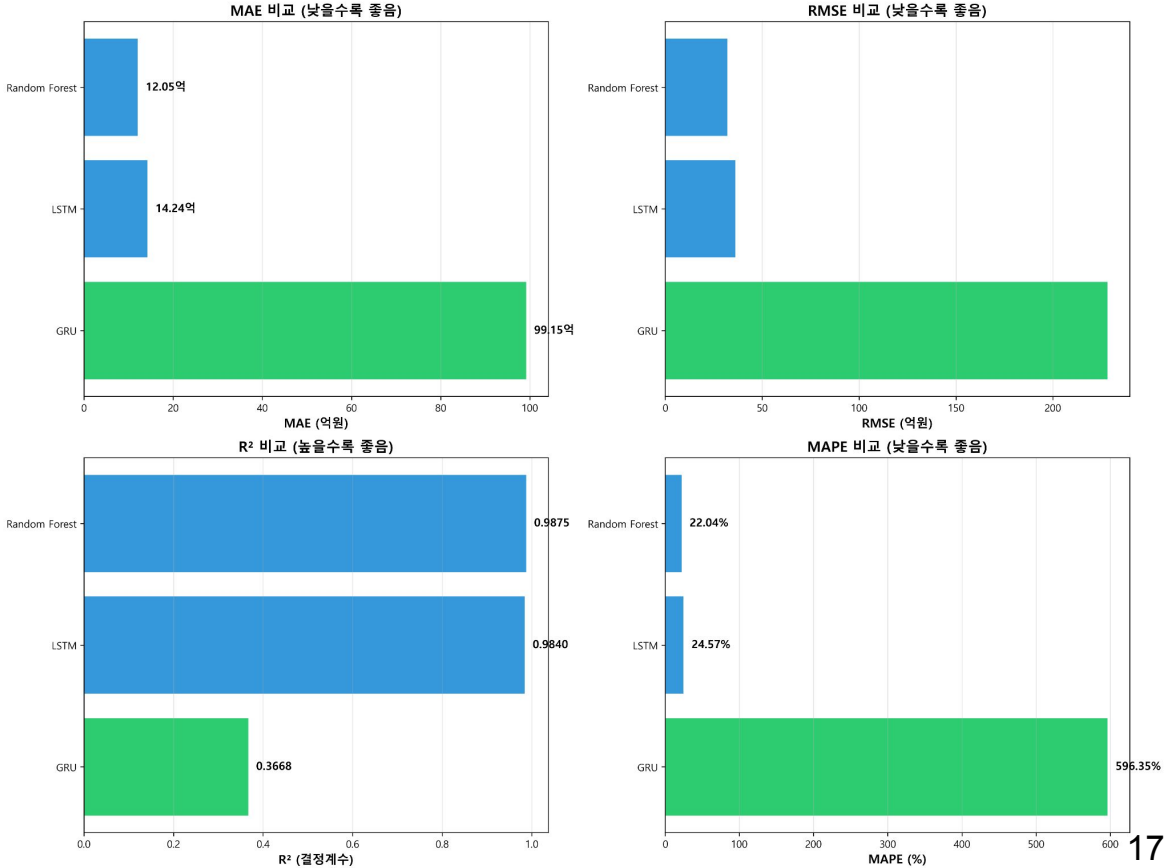
Mae : Validation(0.16) < Train(0.09)
-> 과적합 없음.
->보통은 Val이 Train보다 높음. ->이유 : Dropout, BatchNormalization이 효과적으로 작동했다 OR Validation 데이터가 상대적으로 예측하기 쉬운 패턴이었다.

평가 지표 및 모델 비교-LSTM이 최선인가?

1위: Random Forest
- MAE: 12.05 억원
- RMSE: 32.12 억원
- R^2: 0.9875
- MAPE: 22.04%

2위: LSTM
- MAE: 14.24 억원
- RMSE: 36.22 억원
- R^2: 0.9840
- MAPE: 24.57%
- Epochs: 38

3위: GRU
- MAE: 99.15 억원
- RMSE: 228.16 억원
- R^2: 0.3668
- MAPE: 596.35%



모델비교 : LSTM vs GRU vs RF

2위인 LSTM 선택 이유 :

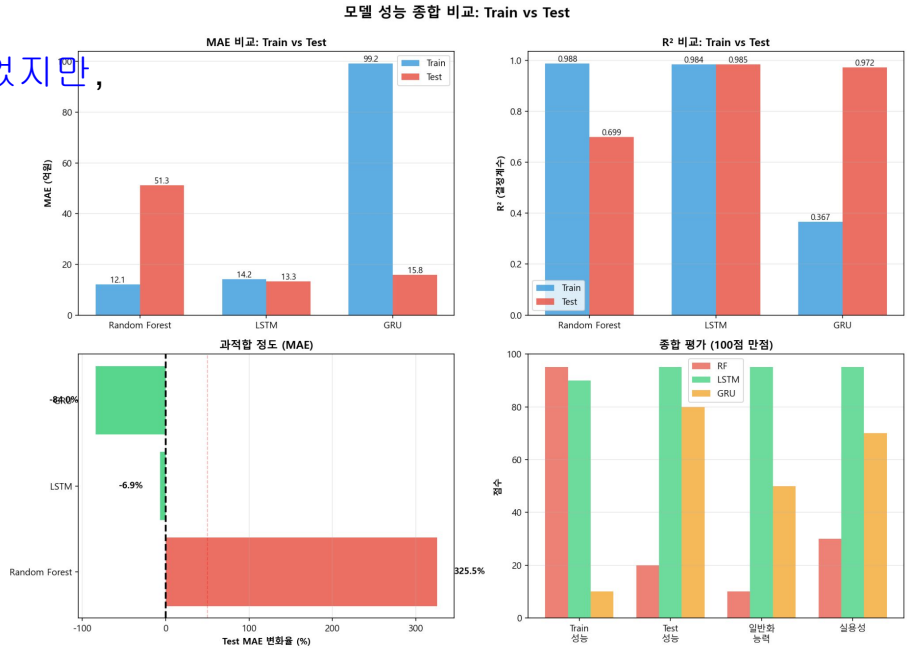
1. 수치 상으로는 RF가 소폭 더 좋았지만, 시계열 자료 통한 예측 문제는 LSTM을 적용하는게 맞다
   (Random Forest = 비선형 문제 vs LSTM =비선형 + 시계열 문제)

2. 앞페이지 그래프는 train 데이터 결과이다. test데이터(새로운 데이터)에서는 다른 모습을 보였다.(아래)

TEST 데이터 적용시, LSTM은 수치에 큰 변화가 없지만,
RF는 성능이 현저히 떨어지는게 눈에 보입니다.
MAE는 4배 가량 떨어지고
R2에서도 30% 가까지 떨어졌습니다.

Rf는 시간에 순서를 무시한다. 그래서
시계열 패턴을 못 잡는다.
그래서 train 데이터는 암기로 잘 맞추지만
test데이터(새로운 데이터)는 잘 못 맞춘다.

시계열 자료에 대하여 LSTM이 더 일반화 능력이 뛰어나다.

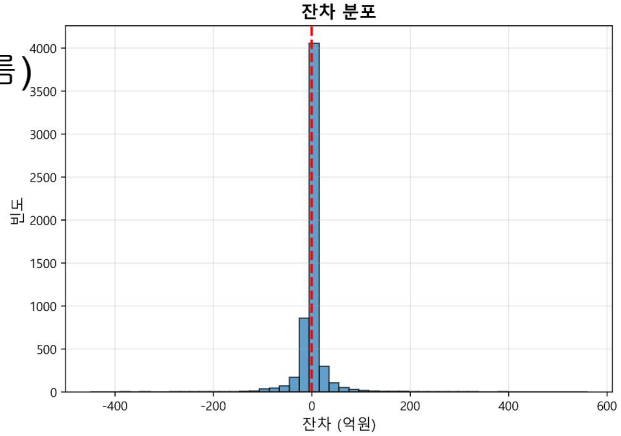


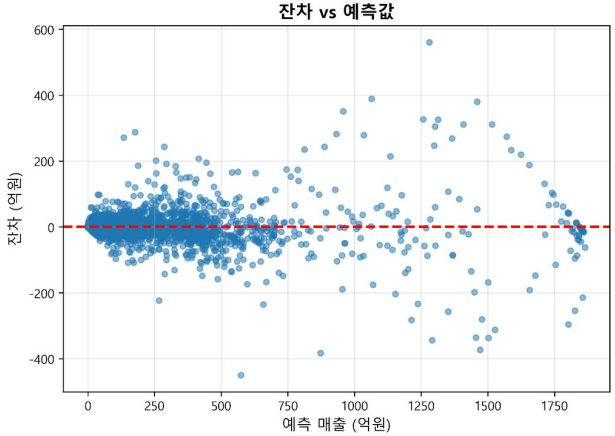
잔차 분석

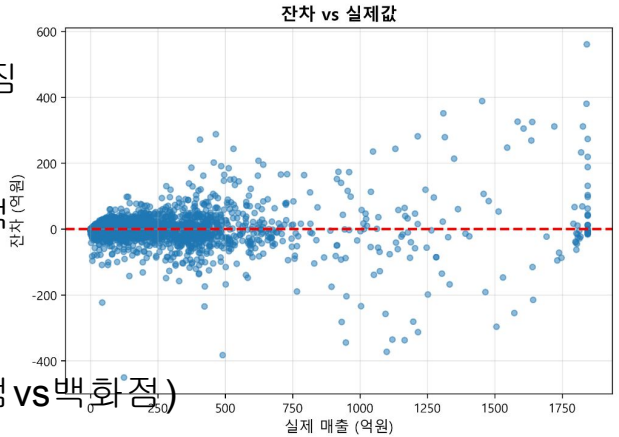
왼쪽 위 : 잔차 분포 (히스토그램) 잔차란? 실제값-예측값
0 중심 대칭, 종 모양
-> 편향 없음, 정규성 만족(정규분포 따름)

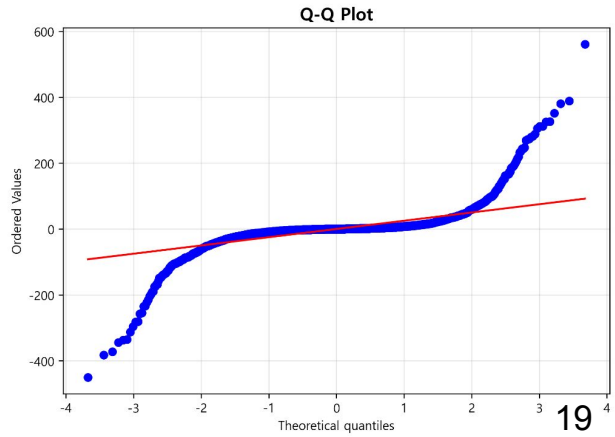
오른쪽 아래: Q-Q Plot
대부분 점이 빨간 대각선 위
중앙부는 거의 완벽
양 끝에서 약간 벗어남 (극단값)
→ 전체적으로 정규성 만족

(잔차 vs 예측값) + (잔차 vs 실제값)
저매출 (~500억) : Y=0에 조밀하게 모임
중매출(500~1000억): Y=0에서 조금 퍼짐
고매출(1000억~): Y=0에서 많이 퍼짐!
저매출(등분산성)-> 고매출(이분산성)
이유1 : 저매출은 100억X5%(오차) = 5억
고매출은 1000억X5%(오차)=50억
매출 크면, 절대 오차 커짐
이유2 : 저매출 업종보다 고매출 업종이
계절성, 이벤트에 영향 크게 받음(편의점vs백화점)
= 자연스러운 현상









결론 및 핵심 성과

📊 연구 목표 달성
✅ 목표: 서울시 구/업종별 분기 매출 예측 시스템 개발
✅ 결과: MAE 13.26억원의 정확도 달성

🏆 주요 성과 (3가지)
1️⃣ 모델 성능 -> 슬라이드 18
MAE: 13.26억원
R^2: 0.985 (98.5% 설명력)
MAPE: 24.57%
Train-Test 일관성 (과적합❌)
2️⃣ 방법론 기여
제곱근 변환으로 11.8% 개선 -> 슬라이드 8
순차 튜닝으로 90% 시간 단축 -> 슬라이드 11
Adam > SGD 실증적 검증 -> 슬라이드 13
3️⃣ 실용적 가치
25개 구 × 60개 업종 예측
창업자 의사결정 지원
정책 입안 기초 자료

개인: 강남에서 음식점 할까? 서초에서 할까?
정부: 어느 업종이 어려워지니, 미리 지원책 준비하자!

⚠️한계 및 개선

1. 과거 매출 자료만을 이용하였습니다. 물론 외부변수(환율, 금리 등)가 매출에 반영된 것이겠지만, 이들을 반영하여 예측 시도를 해보았다면 더 좋았을것 같다고 생각합니다.->외부 변수 추가 실험 (금리, 환율, 유동인구)
2. 데이터 변환시 제곱근이 이미 우수한 성능을 보여주어, 추가 실험은 안 했지만, Box-Cox 변환 추가 실험,

   Yeo-Johnson 변환 등을 비교하면 의미있는 결과를 얻을 수도 있다고 하여, 향후에 시도를 해보고 싶습니다.
3. 데이터의 30%가 10분기의 자료를 갖고 있어서, timesteps = 12를 타임스텝 8과 비교해보지 못했습니다.
4. 오차가 크게 나온 상황(FAILURE CASE)을 확인*분석하고 싶었지만, 시간 관계상 할수 없었습니다. 다음 기회에 도전해보고 싶습니다.->Failure Case 상세 분석

감사합니다.