ROSSMANN 매출 예측 및 경영 지원 프로젝트

박누리

목차

프로젝트 주제 및 선정 배경, 기획의도

Rossmann은 7개 유럽 국가에 위치한 1,115개 이상의 약국을 운영하며, 각 약국은 독립적인 환경에서 운영

수천 개의 약국에서 예측을 수행하는 것은 복잡하며, 예측의 정확도는 약국 관리자와 지리적 위치에 따라 다름

효율적인 매출 예측과 가게 경영을 지원하는 서비스를 개발할 필요성

프로젝트 주제 및 선정
배경, 기획의도

부정확한 매출 예측은 제품 재고 및 재무 관리에 어려움을 초래하여 비용 손실을 발생시킴

불필요한 인력 및 자원을 할당하거나 부적절한 시간에 영업을 닫는 등 비효율적인 매장 운영으로 이어짐

고객은 제품 불일치, 서비스 부족 또는 장시간 대기 등으로 인해 불만족 발생

할인, 프로모션, 인력 할당 및 제품 조달을 개선하여 경쟁력을 유지하는 데 실패할 수 있음

이미지 출처: 게티이미지뱅크

프로젝트 내용

- 선형 회귀 및 그래디언트 부스트 모델을 활용하여 각 약국의 매출을 예측
- 예측된 매출 데이터를 기반으로 인사이트와 권장 사항을 제공하여 경영 결정을 지원하는 시스템 구축
- 시각적으로 매출 예측 및 인사이트를 확인할 수 있는 대시보드 구현

활용방안 및 기대효과

- 활용방안
  - 시간대별, 일별, 주별 매출 예측 그래프를 확인할 수 있는 대시보드로 각 약국의 매출 예측 데이터를 시각적으로 확인
  - 프로모션 실행 일정 및 주기, 특정 상품 재고 유지 등 경영 인사이트와 권장 사항을 제공
  - 언제 어디서나 매출 예측 및 경영 데이터에 액세스할 수 있으며, 실시간으로 재고 업데이트를 수행
- 기대효과
  - 미래 매출을 정확하게 예측하여 재고 관리, 스태프 일정 조정 및 프로모션 계획을 최적화
  - 경영 권장 사항을 통해 경영 전략을 최적화 및 비즈니스 성과 향상
  - 데이터를 활용하여 고객 서비스와 비즈니스 성과를 향상하고, 궁극적으로는 경쟁력 강화

프로젝트 절차 및 구조

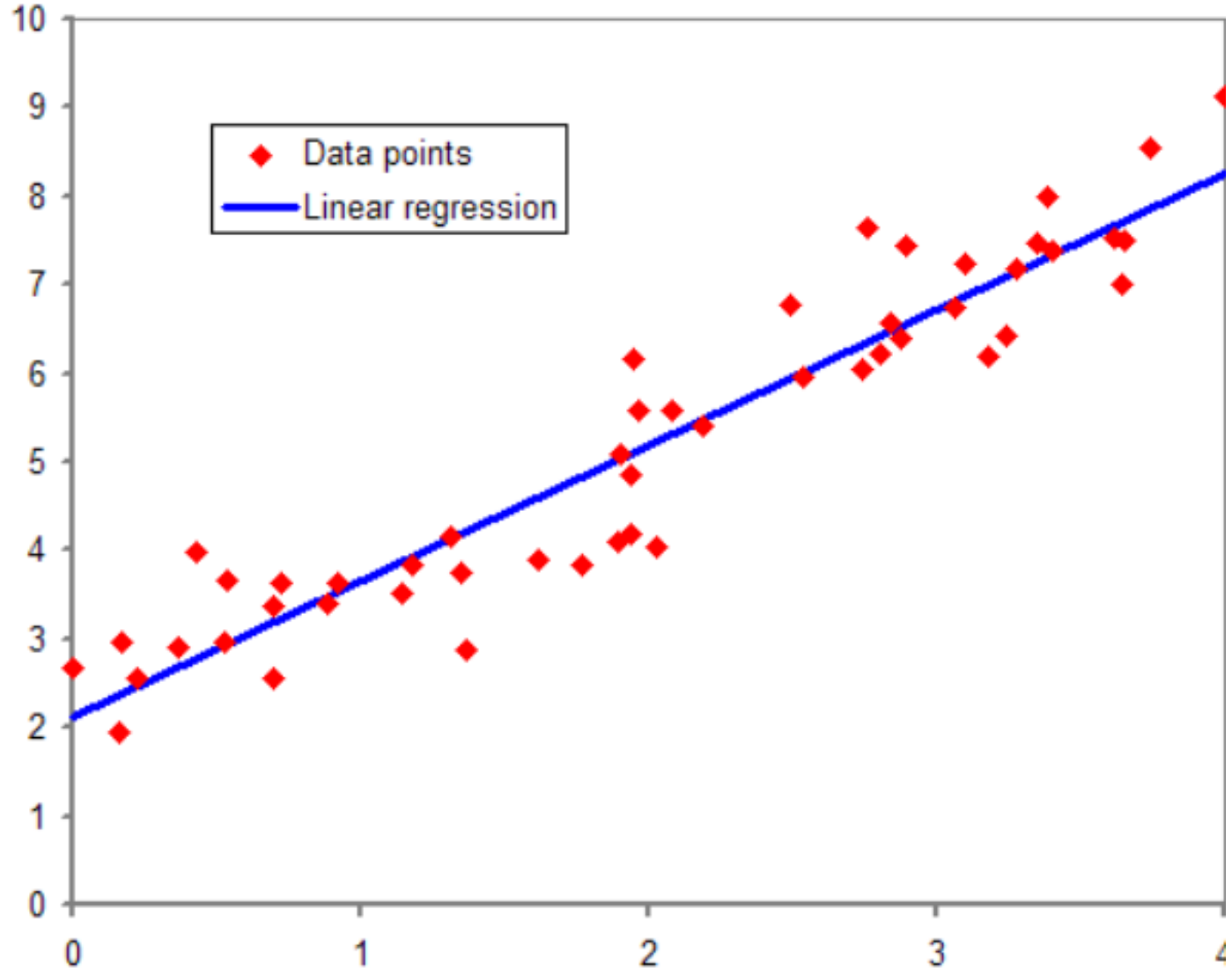
활용 장비 및 재료

Linear Regression

- **선형 회귀**(linear regression)는 종속변수 *y*와 한 개 이상의 독립변수(또는 설명 변수) *X*와의 선형 상관 관계를 모델링하는 회귀분석 기법

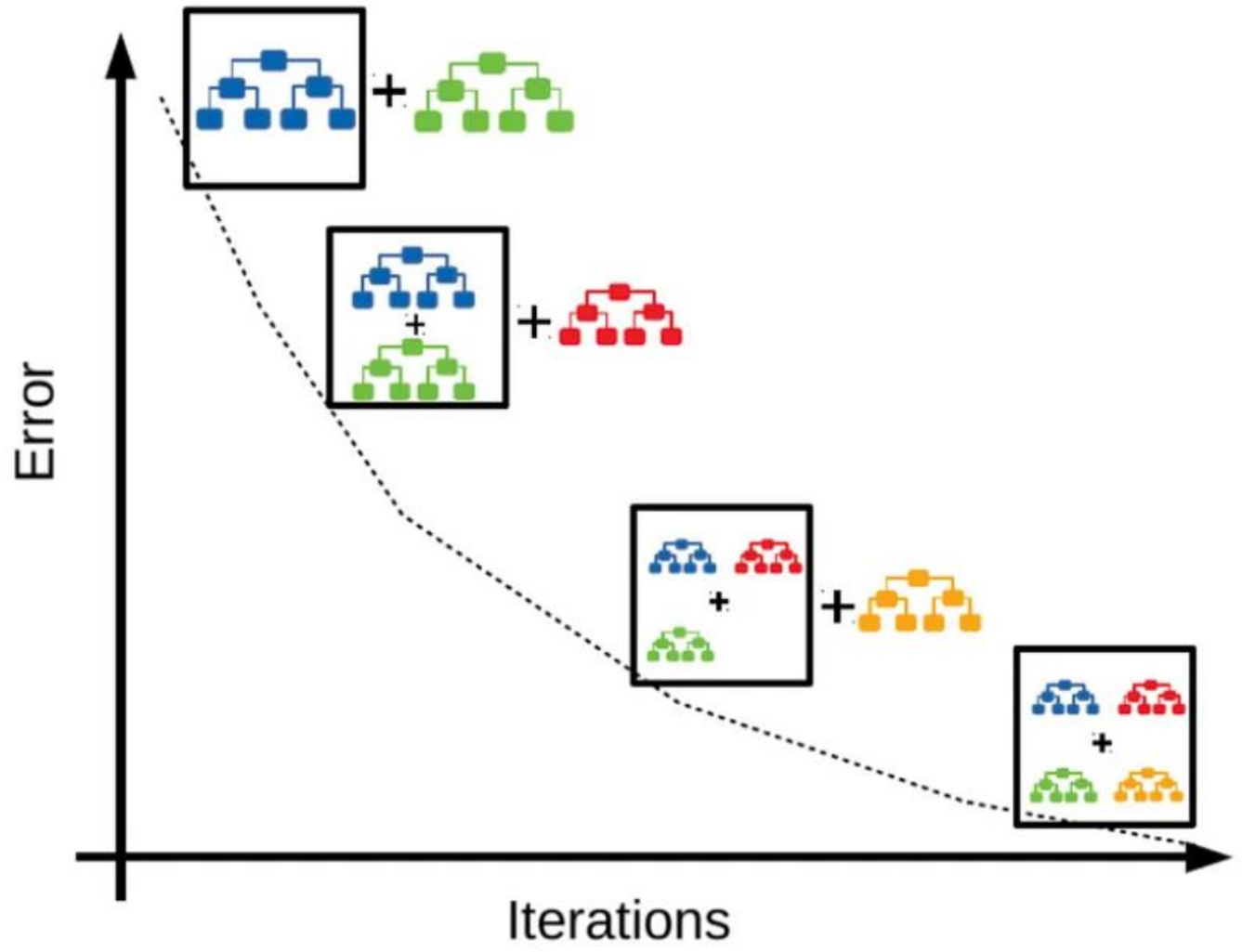
- 지도학습 방법 중 하나

- 데이터들을 가장 잘 대변할 수 있는 **"직선"**을 찾아 새로운 데이터 값을 넣었을 때의 결괏값을 예측할 수 있도록 하는 것



Gradient Boosting

- 그래디언트 부스팅(Gradient Boostiing)은 이전 학습의 결과에서 나온 오차를 다음 학습에 전달해 이전의 오차(잔여 오차)를 점진적으로 개선하는 boosting 기법이자 앙상블 기법
- XGBoost, LightGBM, CatBoost가 있음
- 큰 데이터에 적용 가능하며 scale 변환이 불필요하고 일반화 및 성능이 좋다는 장점이 있지만 파라미터 조정에 민감해 학습률 등의 결과가 달라질 수 있고 개별 트리 분석이 어렵다는 단점이 있음



Why Linear Regression & Gradient Boosting?

선형 회귀 (Linear Regression):

- 매출과 예측 변수 사이에 선형 관계를 가정. 많은 비즈니스 시나리오에서 매출은 여러 요인에 비례하여 증가하거나 감소할 수 있음.
- 각 예측 변수의 계수(가중치)를 해석하기 쉬움. 따라서 특정 요인이 매출에 미치는 영향을 정량적으로 파악할 수 있음. 이것은 비즈니스 의사 결정에 매우 유용함.

그래디언트 부스팅 (Gradient Boosting):

- 매출 예측 문제에서 비선형 관계 및 복잡한 상호 작용이 포함될 수 있음. 그래디언트 부스팅은 이러한 비선형성을 모델링하기에 뛰어남.
- 여러 결정 트리를 앙상블하여 높은 예측 정확도를 제공. 복잡한 패턴과 관계를 잘 파악하며, 고차원 데이터에서 효과적임.
- 범주형 변수, 연속형 변수, 이상치 및 결측값을 다룰 수 있어서 다양한 종류의 데이터를 처리하기에 적합함.

데이터 전처리 과정

```
df_new = df.merge(store,on=["Store"], how="inner")
print(df_new.shape)
```

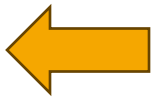
Shape of the Dataset: (1115, 10)

(1017209, 18)

Train 데이터와 Store 데이터를 합침

```
Missing values in  Store is -      0.0
Missing values in  DayOfWeek is -      0.0
Missing values in  Date is -      0.0
Missing values in  Sales is -      0.0
Missing values in  Customers is -      0.0
Missing values in  Open is -      0.0
Missing values in  Promo is -      0.0
Missing values in  StateHoliday is -      0.0
Missing values in  SchoolHoliday is -      0.0
Missing values in  StoreType is -      0.0
Missing values in  Assortment is -      0.0
Missing values in  CompetitionDistance is -      0.25973030124586
Missing values in  CompetitionOpenSinceMonth is -      31.787764363075826
Missing values in  CompetitionOpenSinceYear is -      31.787764363075826
Missing values in  Promo2 is -      0.0
Missing values in  Promo2SinceWeek is -      49.94362023930186
Missing values in  Promo2SinceYear is -      49.94362023930186
Missing values in  PromoInterval is -      49.94362023930186
Missing values in  Month is -      0.0
Missing values in  Quarter is -      0.0
Missing values in  Year is -      0.0
Missing values in  Day is -      0.0
Missing values in  Week is -      0.0
Missing values in  Season is -      0.0
```

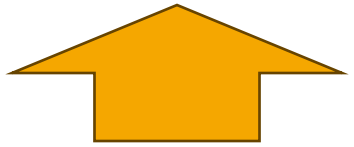
결측치 또는 누락된 값을 제거하기 위해 비율 확인 및 제거



```
df_new["CompetitionDistance"]=df_new["CompetitionDistance"].fillna(df_new["CompetitionDistance"].mode()[0])
```

데이터 수집

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1
3	4	5	2015-07-31	13995	1498	1	1	0	1
4	5	5	2015-07-31	4822	559	1	1	0	1



```
import pandas as pd
df = pd.read_csv("train.csv")
```

```
df.head(5)
```

모델링

첫 번째 모델은 단일 **Dense** 층과 ReLU 활성화 함수를 사용하며, 이 모델은 특정 데이터에 대한 성능을 측정하기 위해 사용됨.

두 번째 모델은 새로운 Sequential 모델로 두 개의 **Dense** 층을 가지며, 각 층은 350개의 뉴런을 사용하고 ReLU 활성화 함수를 사용. 회귀 모델로, 'adam' 옵티마이저 및 평균 제곱 오차(MSE) 손실 함수를 사용하여 컴파일되며, 두 개의 은닉층을 가진 모델.

세 번째는 더 복잡한 모델 아키텍처를 사용하여 모델을 만듦. 여러 개의 **Dense** 층을 가지며, 각 층은 350개의 뉴런을 사용하고 ReLU 활성화 함수를 사용. 이 또한 회귀 모델로, 'adam' 옵티마이저 및 평균 제곱 오차(MSE) 손실 함수를 사용하여 컴파일되며, 다섯 개의 은닉층을 가지고 있으므로 모델의 복잡성이 상대적으로 더 높음.

세 번째 모델은 두 번째 모델보다 더 깊고 넓은 신경망 아키텍처를 사용하며, 더 많은 학습 파라미터를 가질 수 있음. 이로 인해 세 번째 블록의 모델은 보다 복잡한 패턴을 학습할 수 있음.

```
model = Sequential()
model.add(Dense(150,input_dim = 44,activation="relu"))
#The input_dim =44, since the width of the training data=44 (refer data engg section)
model.add(Dense(1,activation = "linear"))

model.compile(optimizer='adam',loss="mean_absolute_error", metrics=["mean_absolute_error"])

model.fit(x_train.values,y_train.values, validation_data=(x_val,y_val),epochs=10,batch_size=64)
```

```
Metric  loss : 725.77
Metric  mean_absolute_error : 725.77
```

```
model = Sequential()
model.add(Dense(350,input_dim = 44,activation="relu"))
model.add(Dense(350,activation="relu"))
model.add(Dense(1,activation = "linear"))
model.compile(optimizer='adam',loss="mean_squared_error",metrics=["mean_absolute_error"])
model.fit(x_train,y_train, validation_data=(x_val,y_val),
epochs=15,batch_size=64)
result = model.evaluate(x_test,y_test)
for i in range(len(model.metrics_names)):
    print("Metric ",model.metrics_names[i],":",str(round(result[i],2)))
```

```
Metric  loss : 840759.19
Metric  mean_absolute_error : 631.05
```

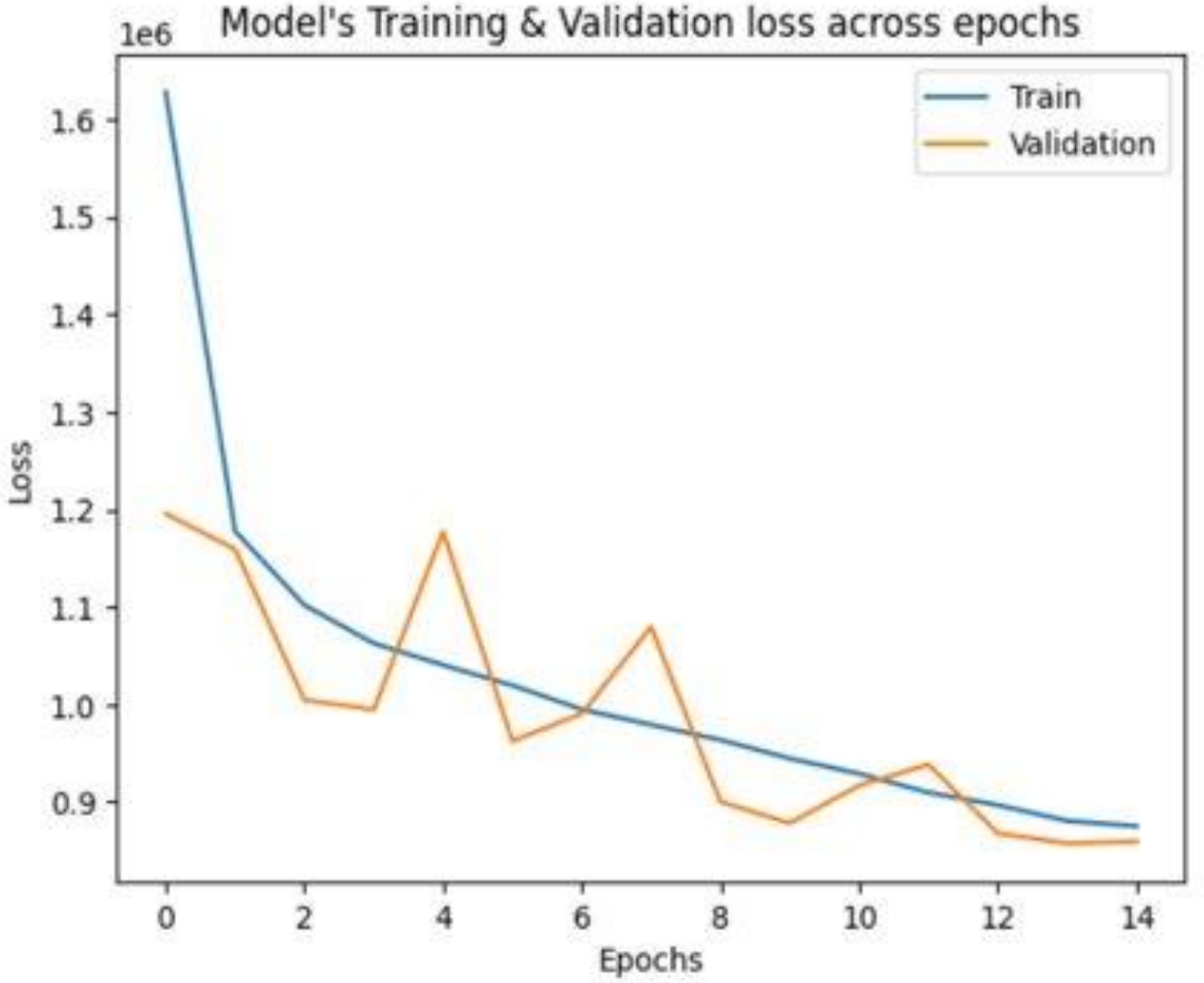
```
from keras.callbacks import History
history = History()
model = Sequential()
model.add(Dense(350,input_dim = 44,activation="relu"))
model.add(Dense(350,activation="relu"))
model.add(Dense(350,activation="relu"))
model.add(Dense(350,activation="relu"))
model.add(Dense(350,activation="relu"))
model.add(Dense(1,activation = "linear"))
model.compile(optimizer='adam',loss="mean_squared_error",metrics=["mean_absolute_error"])
model.fit(x_train,y_train, validation_data=(x_val,y_val),
epochs=15,batch_size=64,callbacks=[history])
result = model.evaluate(x_test,y_test)

for i in range(len(model.metrics_names)):
    print("Metric ",model.metrics_names[i],":",str(round(result[i],2)))
```

```
Metric  loss : 860302.25
Metric  mean_absolute_error : 618.73
```

모델 훈련 및 검증 손실(Training and Validation Loss) 시각화

- 모델의 훈련과 검증 손실을 에포크별로 추적하고, 이 손실을 그래프로 시각화하여 모델의 성능 및 학습 진행 상황을 평가

- 그래프를 통해 훈련과 검증 손실이 어떻게 변하는지 확인할 수 있으며, 손실이 감소하거나 증가하는 패턴을 파악할 수 있음



모델 예측 및 성능 평가

모델을 사용하여 테스트 데이터에 대한 판매량 예측을 수행

모델의 예측이 얼마나 정확한지를 평가하기 위해 모델의 성능을 직접 시각화하고 예측 결과를 실제 데이터와 비교하여 모델의 품질을 평가

평균 제곱 오차(Mean Squared Error, MSE)와 평균 절대 오차(Mean Absolute Error, MAE)를 계산하고 출력하여여 모델의 성능을 수치적으로 평가하고 예측의 품질을 평가

MSE와 MAE가 작을수록 모델의 예측이 더 정확하고, 높은 값은 모델의 예측이 더 부정확함을 나타냄

```
6358/6358 [==============================] - 23s 4ms/step
        Actual Sales  Predicted Sales
115563             0        -0.222091
832654             0        -0.222091
769112          2933      2978.945801
350588          8602      7266.452637
141556          6975      6395.230469
84435           9239      9011.111328
53018              0        -0.222091
262419             0        -0.222091
702267          5885      5237.320312
981431             0        -0.222091
```

```
MSE : 860302.833828449
MAE : 618.7251394011269
```

앙상블 모델링을 활용한 예측 성능 향상

- 선형 회귀 모델과 그래디언트 부스팅 모델을 별도로 학습
- 선형 회귀 모델과 그래디언트 부스팅 모델의 예측 결과를 조합하여 메타 모델을 학습
- 이 메타 모델은 선형 회귀를 사용하며, 두 모델의 예측을 입력으로 사용하여 종속 변수 (meta_model_y)를 학습
- 메타 모델을 사용하여 선형 회귀 모델과 그래디언트 부스팅 모델의 예측을 조합하여 최종 예측을 수행. 여러 모델의 강점을 결합하여 더 정확한 예측을 얻을 수 있음.
- 평균 제곱 오차(MSE) 및 평균 절대 오차(MAE)가 이전보다 낮게 나타나 모델의 성능이 이전보다 더 좋아진 것을 알 수 있음

```python
# 선형 회귀 모델
linear_reg = LinearRegression()
linear_reg.fit(x_train, y_train)

# 그래디언트 부스팅 모델
gradient_boosting = GradientBoostingRegressor()
gradient_boosting.fit(x_train, y_train)

# 각 모델의 예측
linear_reg_predictions = linear_reg.predict(x_test)
gradient_boosting_predictions = gradient_boosting.predict(x_test)

# 예측 결과를 입력으로 사용하여 메타 모델 (선형 회귀) 학습
meta_model_X = np.column_stack((linear_reg_predictions, gradient_boosting_predictions))
meta_model_y = y_test

meta_model = LinearRegression()
meta_model.fit(meta_model_X, meta_model_y)

# 각 모델의 예측을 조합하여 최종 예측을 수행
final_predictions = meta_model.predict(meta_model_X)

# 최종 모델의 성능 평가
final_mse = mean_squared_error(meta_model_y, final_predictions)
final_mae = mean_absolute_error(meta_model_y, final_predictions)
print("Final Mean Squared Error:", final_mse)
print("Final Mean Absolute Error:", final_mae)
```

```
Final Mean Squared Error: 571023.2256110471
Final Mean Absolute Error: 481.10806247744347
```

자체 평가

- **잘한 점**

(1) 다양한 접근 방식

(2) 계획대로 진행

- **보완해야 할 점**

(1) 시각화 자료 부족

(3) 활용 방안을 구체화할 필요