

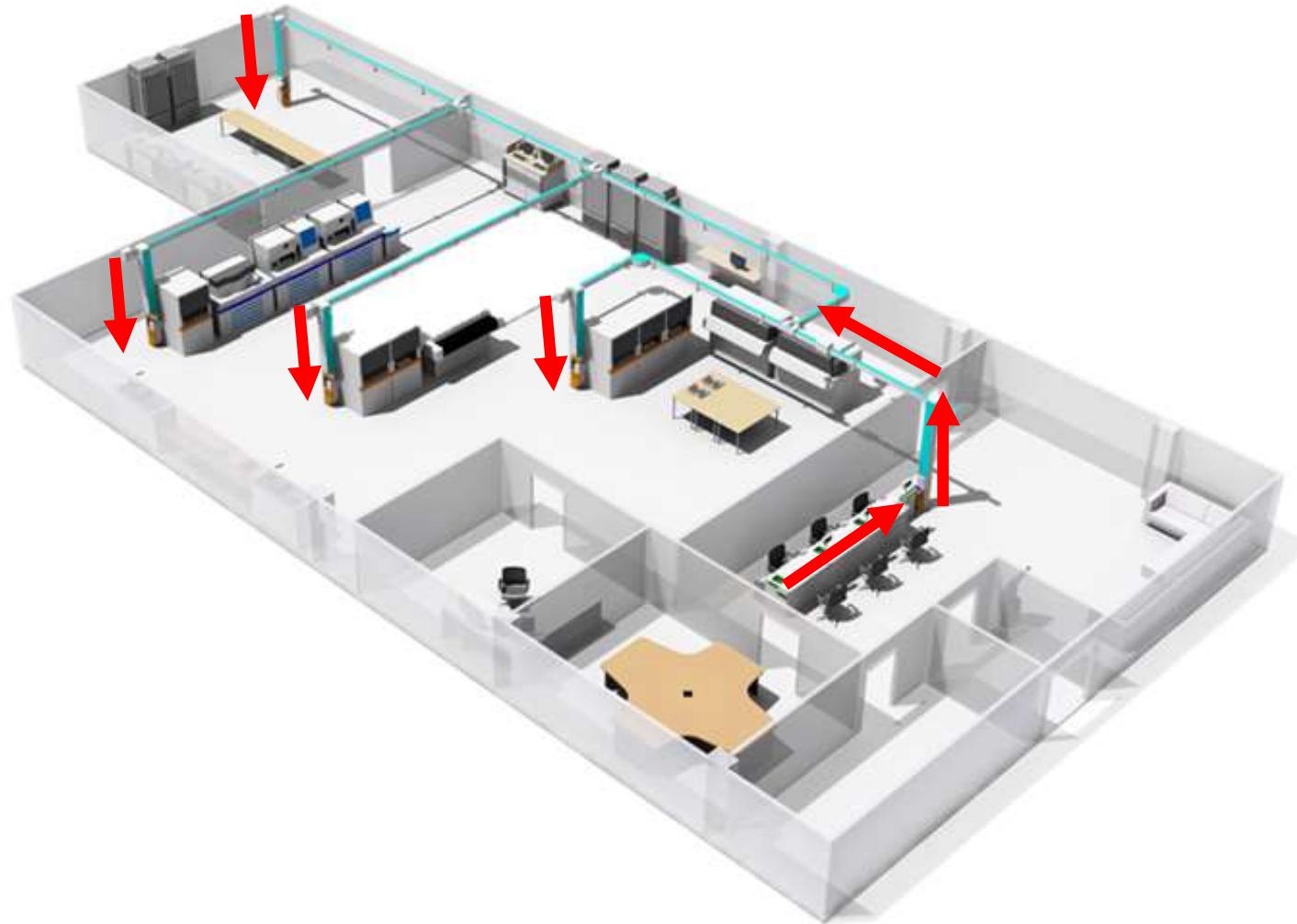
Final Project

Log데이터를 통한 병원 환자 추이 분석

데이터 분석의 필요성 및 목적

- ◎ 채혈 검사량 증가 및 감소 추이를 분석하여 병원 운영에 관한 활용 데이터로 사용하고자 함
- ◎ 시간대 별 채혈 검사량 데이터를 이용한 효율적인 검사 인력 분배
- ◎ 검사 시약 및 물품 관리에 데이터 활용성

활용 데이터 설명



이송 컨베이어 Log 데이터

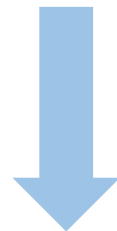
채혈실에서 환자로부터 채혈한 혈액
검체를 자동으로 각 검사항목별로 분
배 이송하는 컨베이어 장비의 Log
Data를 활용

(2020년도 8월 ~ 2021년도 9월)

활용 데이터 설명

T_NUMBER	P_START	P_END	T_START	T_END
450600123	S1U1	M3D9	2020-08-07 8:03	2020-08-07 8:07
450600128	S1U1	M3D9	2020-08-07 8:04	2020-08-07 8:08
450600043	S2U1	M3D9	2020-08-07 8:06	2020-08-07 8:10
450600017	S2U1	M3D9	2020-08-07 8:07	2020-08-07 8:10
450600048	S2U1	M3D9	2020-08-07 8:07	2020-08-07 8:10
450500661	S2U1	M3D9	2020-08-07 8:07	2020-08-07 8:10
450600131	S1U1	M3D9	2020-08-07 8:08	2020-08-07 8:12
450600052	S2U1	M3D9	2020-08-07 8:11	2020-08-07 8:15
450600133	S1U1	M3D9	2020-08-07 8:11	2020-08-07 8:17

T_NUMBER : 혈액 바코드 정보
P_START : 접수 위치 코드
P_END : 처리 완료 위치 코드
T_START : 접수 시작 시간
T_END : 처리 완료 시간



시간대 별 혈액 접수 수량을 확인하여 내원
환자 수 확인 및 최초 1시간 환자수를 통한
당일 전체 환자 수 예측

상관 관계 이론적 예상

1. (가설) 접수 된 혈액의 수와 환자의 수가 일치한다.
2. (가설) 오전 8시~9시에 내원하는 환자수가 가장 많다.
3. (가설) 오전 8시~9시에 내원하는 환자수가 많다면 당일 전체 방문하는 환자 수도 많을 것이다.

=> 최초 1시간 데이터를 통해 당일 전체 환자수를 예측하여 병원 인력 배치, 검사 시약 및 기타 물품 사전 구비 등 다양한 대응을 할 수 있을 것이다.

데이터 전처리

① 최초 데이터 셋 확인



② 년도, 월, 일, 시간 Column 추가



③ 불필요 Column 및 Null 데이터 제거



④ 비정상 데이터 제거



⑤ 분석에 필요한 'Count' Column 추가

데이터 전처리

#step 1 : 최초 데이터 셋 확인

```
print('-----STEP1-----')
```

```
data = pd.read_csv('m_db2.csv', sep=',')
```

```
data_1 = data.copy()
```

```
print(data_1.info())
```

최초 데이터 확인

총 5개의 열 및 데이터 개수 약
19000개 확인.

#	Column	Non-Null Count	Dtype
0	T_NUMBER	194160 non-null	object
1	P_START	192882 non-null	object
2	P_END	194160 non-null	object
3	T_START	194160 non-null	object
4	T_END	194160 non-null	object

데이터 전처리

```
#step 2 : 년도, 날짜, 시간 컬럼 추가
print('-----STEP2-----')

data_1['T_START'] = pd.to_datetime(data_1['T_START'])
data_1['T_END'] = pd.to_datetime(data_1['T_END'])
data_1['YEAR'] = data_1['T_END'].dt.year
data_1['MONTH'] = data_1['T_END'].dt.month
data_1['DAY'] = data_1['T_END'].dt.day
data_1['HOUR'] = data_1['T_END'].dt.hour
data_1.dropna(axis=0) #null값 제거
print(data_1.info())

print('-----')
```

#	Column	Non-Null Count	Dtype
0	T_NUMBER	194160 non-null	object
1	P_START	192882 non-null	object
2	P_END	194160 non-null	object
3	T_START	194160 non-null	datetime64[ns]
4	T_END	194160 non-null	datetime64[ns]
5	YEAR	194160 non-null	int64
6	MONTH	194160 non-null	int64
7	DAY	194160 non-null	int64
8	HOUR	194160 non-null	int64

년도, 날짜, 시간 열 추가

시간대 별 환자 숫자를 확인 하기 위
하여 T_END 열을 date time형식으로
변경 후 해당 열로부터 년도, 월, 일,
시간 열을 추가로 생성

데이터 전처리

#step 3 : 날짜 시간별 카운트를 위한 불필요 열 제거

```
print('-----STEP3-----')
```

data_1.dropna(axis=0) #null값 제거

data_2 = data_1.drop(['T_NUMBER', 'P_START', 'P_END', 'T_START', 'T_END'], axis=1)

```
print(data_2.info())
```

```
print('-----')
```

-----STEP3-----

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 194160 entries, 0 to 194159

Data columns (total 4 columns):

#	Column	Non-Null Count	Dtype
0	YEAR	194160 non-null	int64
1	MONTH	194160 non-null	int64
2	DAY	194160 non-null	int64
3	HOURL	194160 non-null	int64

dtypes: int64(4)

불필요 열 제거 및 Null값 제거

분석 용이성을 위해 분석 시 필요한 날짜 및 시간 데이터를 제외한 열을 제거하고 Null값인 데이터를 제거


데이터 전처리

#step 4: 비정상 데이터 제거

```
print('-----STEP4-----')
```

```
print(data_2.groupby('YEAR').count())  
error_idx = data_2[data_2['YEAR'] == 1899].index  
data_3 = data_2.drop(error_idx)  
print(data_3.groupby('YEAR').count())
```

-----STEP4-----			
	MONTH	DAY	HOUR
YEAR			
1899	1217	1217	1217
2020	53947	53947	53947
2021	138996	138996	138996



	MONTH	DAY	HOUR
YEAR			
2020	53947	53947	53947
2021	138996	138996	138996

비정상 데이터 제거

분석 전 년도가 1899로 표시된 비정상 적인 데이터를
확인하였고 해당 데이터를 제거

데이터 전처리

#step 5 : 카운트 값 추가

```
print('-----STEP5-----')
```

```
# c_data['CNT'] = c_data['YEAR'].groupby()
```

```
# c_data['CNT'] = c_data.value_counts()
```

```
data = pd.read_csv('final_data2.csv', sep=',')
```

```
data_4 = data.copy()
```

```
data_4 = data_4.drop_duplicates(keep='first')
```

```
print(data_4.info)
```

```
data_4.to_csv("final_data3.csv")
```

```
print('-----STEP5-----')
```

#step 6 : 최종 전처리 데이터 확인

```
print('-----STEP6-----')
```

```
final_data = pd.read_csv('final_data4.csv', sep=',')
```

```
no_use_idx = final_data[final_data['HOUR'] != 8].index
```

```
final_data = final_data.drop(no_use_idx)
```

```
print(final_data.info)
```

```
print('-----STEP6-----')
```

[2775 rows x 5 columns]>

<bound method DataFrame.info of						YEAR	MONTH	DAY	HOUR	CNT	TOTAL_CNT
0	2020	8	7	8	66	371					
10	2020	8	10	8	99	549					
20	2020	8	11	8	91	558					
29	2020	8	12	8	58	472					
38	2020	8	13	8	60	558					
...					
2726	2021	9	27	8	309	1101					
2736	2021	9	28	8	58	627					
2745	2021	9	29	8	141	876					
2756	2021	9	30	8	208	869					
2765	2021	10	1	8	192	805					

당일 오전
8시~9시
환자수

당일 전체
환자수

카운터 값 추가 및 최종 데이터 확인

분석에 중요한 당일 오전 8시~9시 병원 방문 환자수
및 당일 전체 방문 환자수를 추가 하고 최종 데이터
확인. 확인 결과 총 2775일에 대한 데이터를 확보.

탐색적 데이터 분석

#step 7 : numpy를 이용한 데이터 시각화

```
print('-----STEP7-----')
```

```
np_final_data = np.array(final_data)
```

```
x_data = np_final_data[:,4]
```

```
y_data = np_final_data[:,5]
```

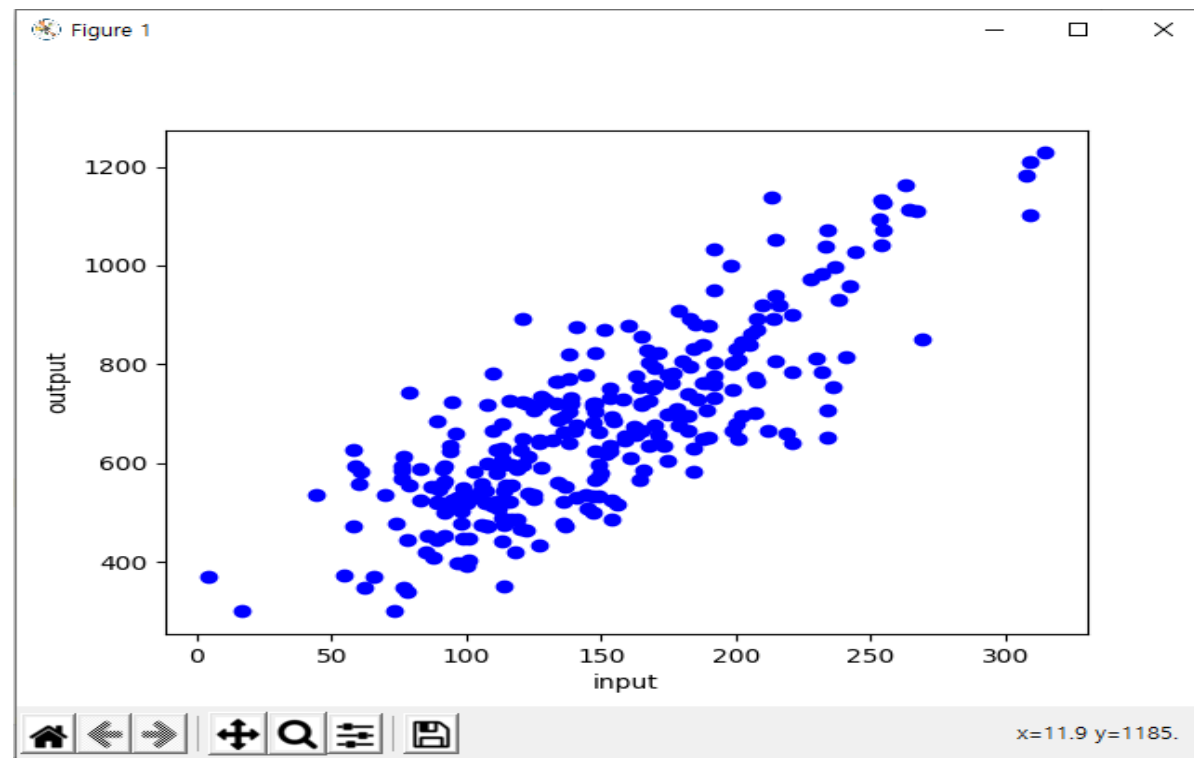
```
plt.plot(x_data,y_data,'bo')
```

```
plt.xlabel('input')
```

```
plt.ylabel('output')
```

```
plt.show()
```

```
print('-----')
```



데이터 시각화

데이터 전처리 과정이 끝난 당일 오전 8시~9시 환자수
(X축)에 대한 당일 전체 환자 수(Y축)를 시각화 한 결과
선형적인 형태를 보이는 것을 확인

딥러닝 데이터 분석 – Linear Regression

#step 8 : sklearn 선형 회귀를 통한 분석 및 예측

```
print('-----STEP8-----')
```

#train data 및 test data 나누기

```
x = final_data[['CNT']]
```

```
y = final_data[['TOTAL_CNT']]
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y,train_size=0.85, test_size=0.15)
```

#예측 모델 생성

```
lr = LinearRegression()
```

```
lr.fit(x_train,y_train)
```

#정확도 측정

```
train_accuracy = lr.score(x_train,y_train)
```

```
print("lr train acc : {:.3f}".format(train_accuracy))
```

```
test_accuracy = lr.score(x_test,y_test)
```

```
print("lr test acc : {:.3f}".format(test_accuracy))
```

```
-----STEP8-----
```

```
lr train acc : 0.657
```

```
lr test acc : 0.669
```

```
10897.20091093864
```

```
w : [[2.65322895]] [280.95367874]
```

```
1
```

Linear Regression 분석 실행

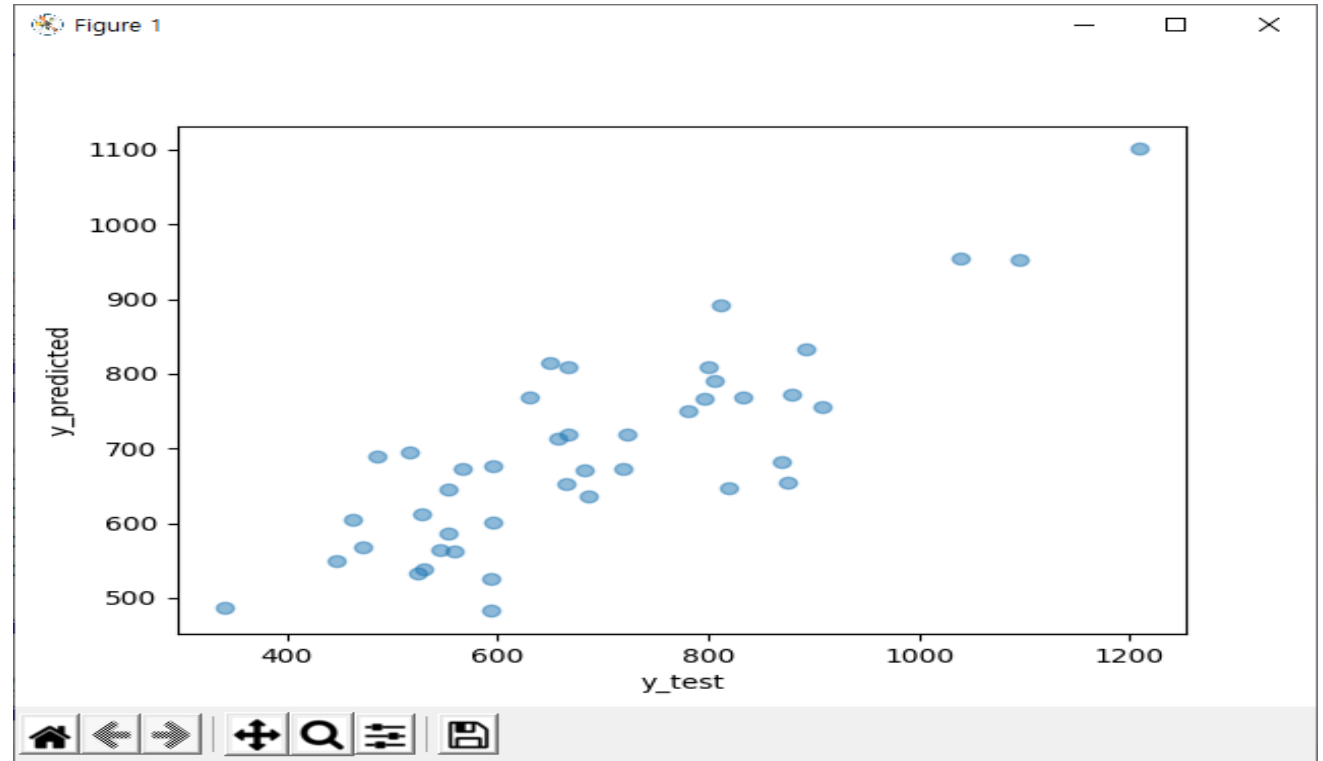
분석 결과 Accuracy : 66.9%

Weight : 2.65

Bias : 280.95

딥러닝 데이터 분석 – Linear Regression

```
#산포도 구해보기  
plt.scatter(y_test,y_predicted, alpha=0.5)  
plt.xlabel('y_test')  
plt.ylabel('y_predicted')  
plt.show()
```



Linear Regression 결과값 산포도 확인

Test data에 대한 예측 값과 실제 값을 비교한 그래프
를 확인한 결과 선형적인 모습은 확인 하였으나 오차
값이 큰 부분이 존재

딥러닝 데이터 분석 - Ridge

```
#예측 모델 생성
rdg = Ridge()
rdg.fit(x_train,y_train)
rdg01 = Ridge(alpha=0.1)
rdg01.fit(x_train,y_train)
rdg02 = Ridge(alpha=0.0001)
rdg02.fit(x_train,y_train)
rdg03 = Ridge(alpha=10000)
rdg03.fit(x_train,y_train)

#결과 확인
rdg_test_accuracy = rdg.score(x_test,y_test)
print("rdg test acc : {:.3f}".format(rdg_test_accuracy))
print('w : ',rdg.coef_,rdg.intercept_)
rdg01_test_accuracy = rdg01.score(x_test,y_test)
print("rdg01 test acc : {:.3f}".format(rdg01_test_accuracy))
print('w : ',rdg01.coef_,rdg01.intercept_)
rdg02_test_accuracy = rdg02.score(x_test,y_test)
print("rdg02 test acc : {:.3f}".format(rdg02_test_accuracy))
print('w : ',rdg02.coef_,rdg02.intercept_)
rdg03_test_accuracy = rdg03.score(x_test,y_test)
print("rdg03 test acc : {:.3f}".format(rdg03_test_accuracy))
print('w : ',rdg03.coef_,rdg03.intercept_)
```

```
-----STEP9-----
rdg test acc : 0.669
w : [[2.65322529]] [280.95422177]
rdg01 test acc : 0.669
w : [[2.65322859]] [280.95373304]
rdg02 test acc : 0.669
w : [[2.65322895]] [280.95367879]
rdg03 test acc : 0.667
w : [[2.61714896]] [286.31021482]
-----
```

Ridge 모델 분석 실행

분석 결과 Accuracy : 66.9%

Weight : 2.65

Bias : 280.95

-> Linear Regression 모델과 큰 차이가 없음. Alpha값에 대한 영향이 크지 않음

딥러닝 데이터 분석 - Lasso

```
#step 9 : sklearn Lasso 모델을 통한 분석 및 예측
print('-----STEP10-----')

#예측 모델 생성
lso = Lasso()
lso.fit(x_train,y_train)

lso01 = Lasso(alpha=0.01, max_iter=10000)
lso01.fit(x_train,y_train)

lso02 = Lasso(alpha=100, max_iter=10000)
lso02.fit(x_train,y_train)

#결과 확인
lso_test_accuracy = lso.score(x_test,y_test)
print("lso test acc : {:.3f}".format(lso_test_accuracy))
print('w : ',lso.coef_,lso.intercept_)
|
lso01_test_accuracy = lso01.score(x_test,y_test)
print("lso01 test acc : {:.3f}".format(lso01_test_accuracy))
print('w : ',lso01.coef_,lso01.intercept_)

lso02_test_accuracy = lso02.score(x_test,y_test)
print("lso02 test acc : {:.3f}".format(lso02_test_accuracy))
print('w : ',lso02.coef_,lso02.intercept_)
```

```
-----STEP10-----
lso test acc : 0.669
w : [2.65289533] [281.00320904]
lso01 test acc : 0.669
w : [2.65322561] [280.95417404]
lso02 test acc : 0.667
w : [2.61986686] [285.90670873]
-----
```

Lasso 모델 분석 실행

분석 결과 Accuracy : 66.9%

Weight : 2.65

Bias : 281

-> Linear Regression 모델과 큰 차이가 없음. Alpha값에 대한 영향이 크지 않음

데이터 분석 결과 요약

- 정확도의 경우 66.9% 정도로 예상보다 정확한 예측을 하지 못하였다.
- 분석 모델 및 Alpha값에 따른 결과 차이가 크지 않다.
- 기존의 가설 내용들을 보완한다면 보다 정확한 데이터 분석 및 예측이 가능할 것이라 생각한다.

(가설) 접수 된 혈액의 수와 환자의 수가 일치한다.

-> 전날 처리 되지 못한 혈액이 다음날 오전에 일괄 접수됨.

(가설) 오전 8시~9시에 내원하는 환자수가 가장 많다.

-> 장비 이슈로 인하여 오전에 장비에 데이터가 제대로 접수되지 않거나 병원 자체 이슈로 오픈 시간이 늦어지는 경우 발생.

(가설) 오전 8시~9시에 내원하는 환자수가 많다면 당일 전체 방문하는 환자 수도 많을 것이다.

=> 선형적인 관계에 있음을 확인.



감사합니다.