당뇨병 예측

2021108241 컴퓨터공학전공 강소민

목차

01 배경 및 개요

당뇨란?

신체가 인슐린을 충분히 생산하지 못하거나 인슐린에 정상적으로
반응하지 못하여 혈당(포도당) 수치가 비정상적으로 높아지는 질환

신경 손상, 혈관 손상, 심장 마비, 쇼크 등을 야기

01 배경 및 개요

20대 사이에서 증가하는 당뇨 환자

음주, 흡연에 쉽게 노출

당뇨 예측 및 조기 발견 필요

02 데이터 셋

# Pregnancies PFA Data Dictionary	# Glucose PFA Data Dictionary	# BloodPressure PFA Data Dictionary	# SkinThickness PFA Data Dictionary	# Insulin PFA Data Dictionary
 0 17	 0 199	 0 122	 0 99	 0 846
6	148	72	35	0
1	85	66	29	0
8	183	64	0	0
1	89	66	23	94
0	137	40	35	168
5	116	74	0	0
3	78	50	32	88
10	115	0	0	0

02 데이터 셋

Pregnancies - 임신

Glucose - 혈당

BloodPressure - 혈압

SkinThickness - 피부 두께

Insulin - 인슐린

BMI - 체질량 지수

DiabetesPedigreeFunction - 당뇨가계통 기능 (당뇨의 유전적 요소)

Age - 나이

Outcome - 결과

03 데이터 분석

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

파이썬 라이브러리 호출

pandas : 데이터를 다루는 라이브러리
numpy : 수학적 연산을 위한 라이브러리
seaborn : 시각화를 위한 라이브러리

03 데이터 분석

```
path = "/kaggle/input/predict-diabities/diabetes.csv"
diabetes = pd.read_csv(path)

df = diabetes.copy()
```

Kaggle 환경에서 제공하는 데이터셋에 csv 파일 호출 후 데이터프레임으로 저장

03 데이터 분석

```
df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

데이터프레임의 정보

768개의 행 X 9개의 열

03 데이터 분석

```
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

데이터프레임의 결측지 값의 개수 파악

```
df.nunique()
```

```
Pregnancies                17
Glucose                   136
BloodPressure              47
SkinThickness              51
Insulin                   186
BMI                       248
DiabetesPedigreeFunction  517
Age                        52
Outcome                     2
dtype: int64
```

데이터프레임의 고유값 개수 반환
ex) Outcome의 값은 0과 1로 2개

03 데이터 분석

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

데이터프레임의 기본 정보 요약

- 전체 행, 열의 개수 -
- 각열의 데이터 타입 -
- 각 열의 결측지 개수 -
- 데이터프레임의 메모리 사용량 -

03 데이터 분석

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

데이터프레임의 기술 통계량

count : 모든 열의 데이터 값 개수
mean : 각 열의 평균값
std: 각 열의 표준편차
min : 각 열의 최솟값
max : 각 열의 최대값

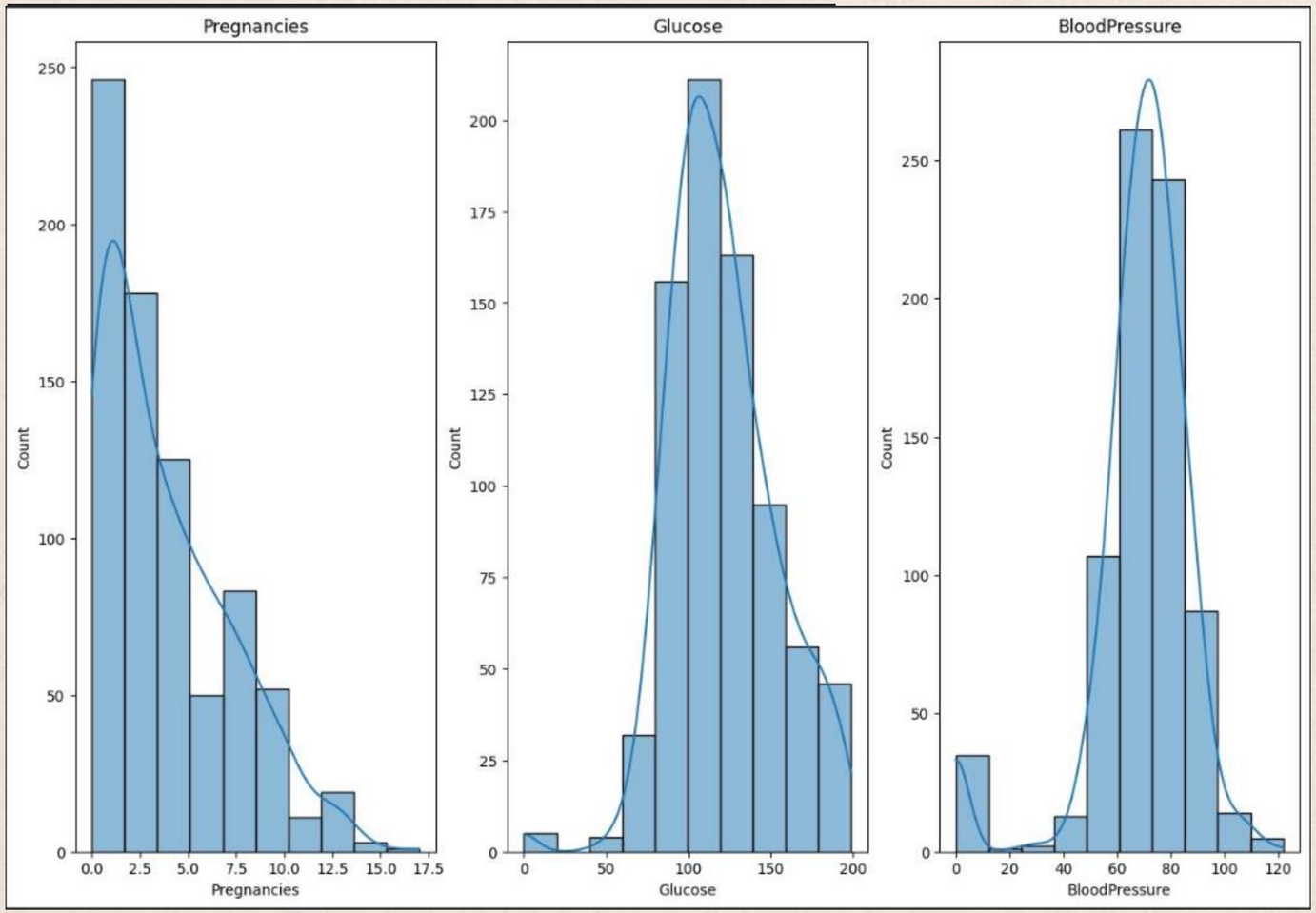
03 데이터 분석

```
fig, axes = plt.subplots(1, 3, figsize = (15, 10))
axes = axes.flatten()

sns.histplot(ax = axes[0], x = df['Pregnancies'],
             bins = 10,
             kde = True,
             cbar = True).set(title = 'Pregnancies')

sns.histplot(ax = axes[1], x = df['Glucose'],
             bins = 10,
             kde = True,
             cbar = True).set(title = 'Glucose')

sns.histplot(ax = axes[2], x = df['BloodPressure'],
             bins = 10,
             kde = True,
             cbar = True).set(title = 'BloodPressure')
```

```
[Text(0.5, 1.0, 'BloodPressure')]
```

데이터프레임 중 임신, 혈당, 혈관에 대한 히스토그램 시각화

03 데이터 분석



임신 횟수는 0회에서 2.5회
혈당과 혈관 수치는
중간 수치에 가장 많은 분포

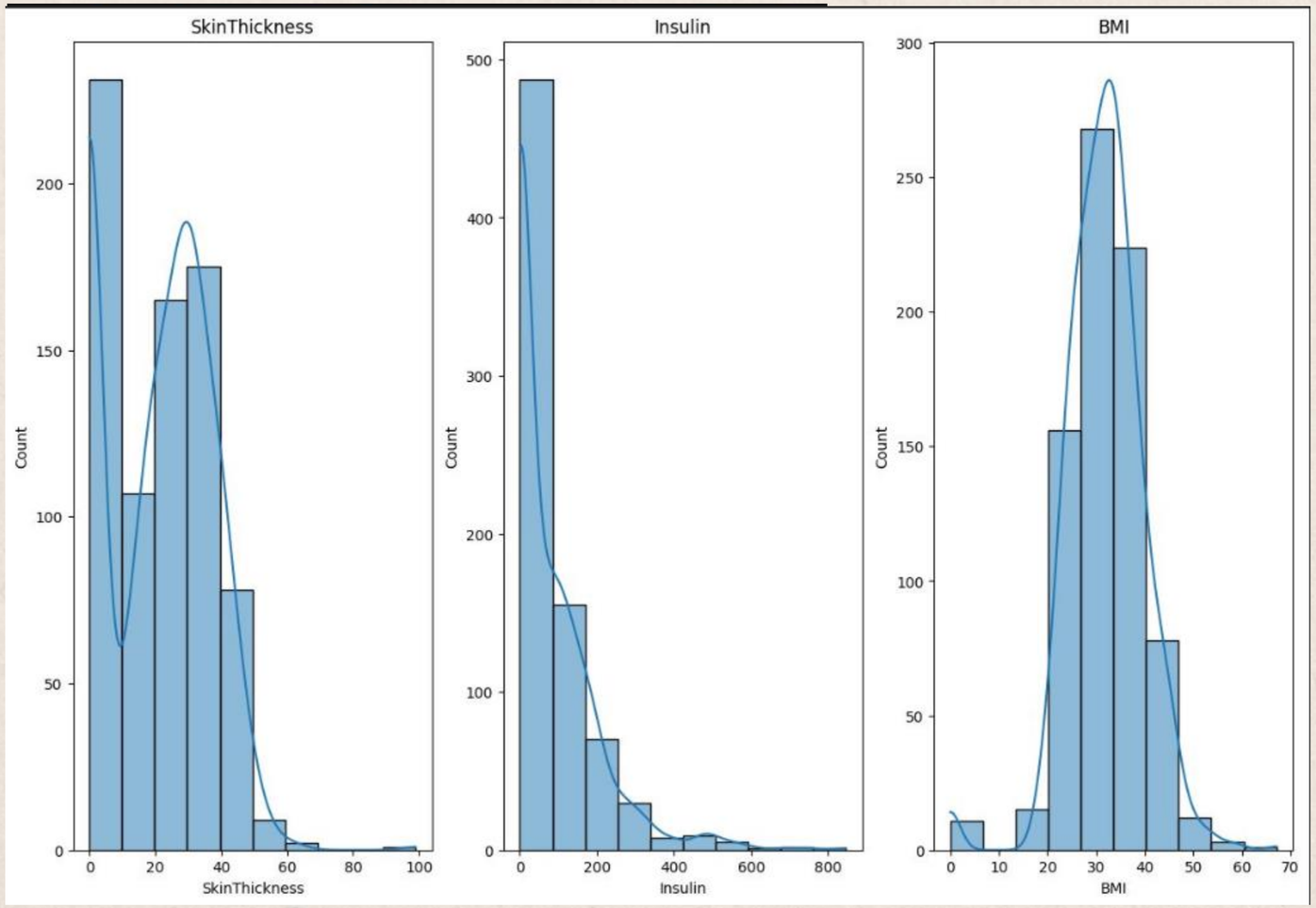
03 데이터 분석

```
fig, axes = plt.subplots(1, 3, figsize = (15, 10))
axes = axes.flatten()

sns.histplot(ax = axes[0], x = df['SkinThickness'],
             bins = 10,
             kde = True,
             cbar = True).set(title = 'SkinThickness')

sns.histplot(ax = axes[1], x = df['Insulin'],
             bins = 10,
             kde = True,
             cbar = True).set(title = 'Insulin')

sns.histplot(ax = axes[2], x = df['BMI'],
             bins = 10,
             kde = True,
             cbar = True).set(title = 'BMI')
```

```
[Text(0.5, 1.0, 'BMI')]
```

데이터프레임 중 피부 두께, 인슐린, BMI에 대한 히스토그램 시각화

03 데이터 분석

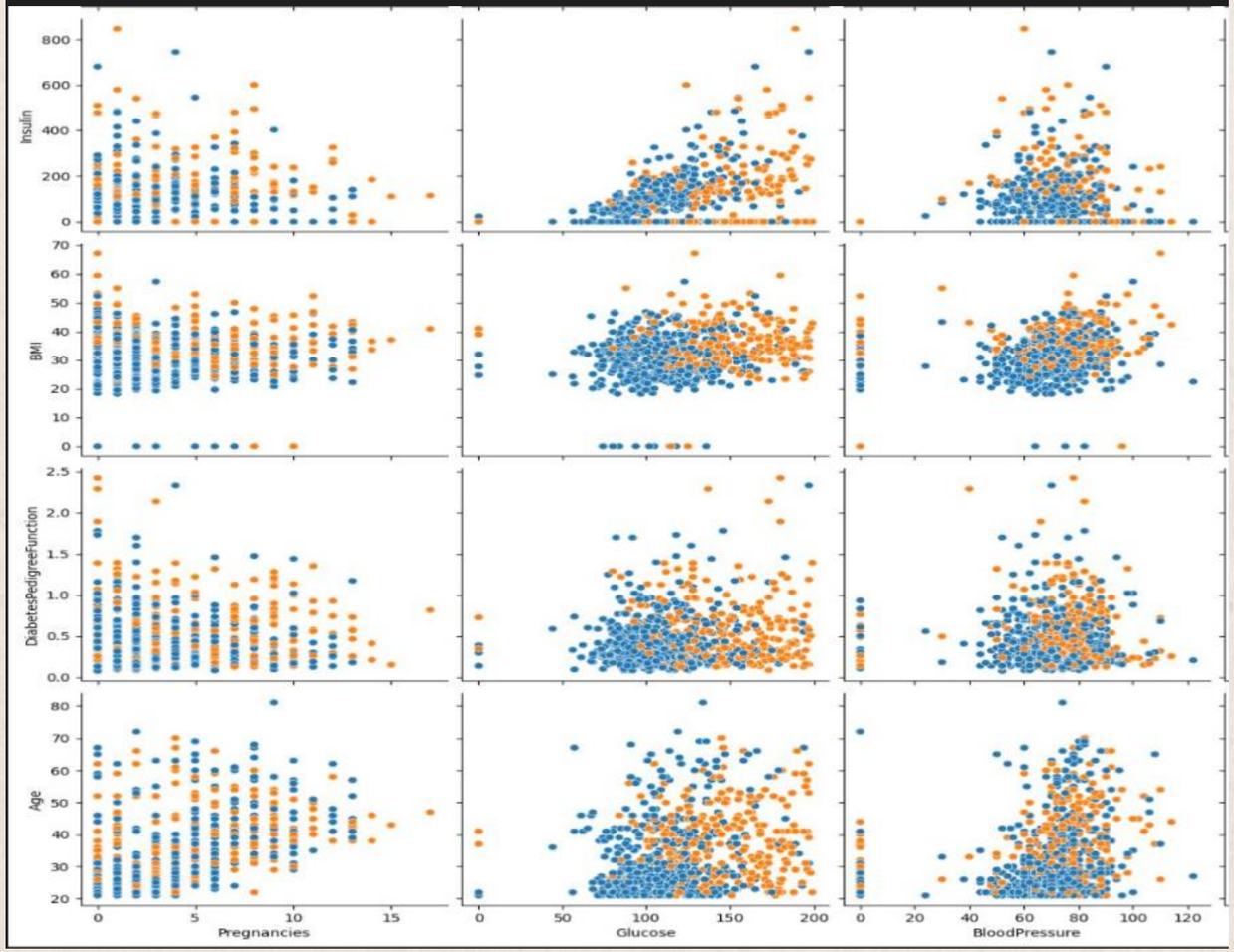


피부 두께는 얇을수록,
인슐린은 적을수록,
BMI는 중간수치에서
높은 분포를 띔

03 데이터 분석

```
sns.pairplot(df, diag_kind = 'hist', hue = 'Outcome', height = 3, aspect = 1.2, corner = True)
```

```
<seaborn.axisgrid.PairGrid at 0x7d638931a020>
```



모든 변수간의 산점도와 히스토그램을
한눈에 보여주는 플롯

Outcome 변수의 값을 기준으로
데이터를 색으로 구분하여 산점도를 시각화

모든 변수들 간의 관계와 데이터의
패턴을 쉽고 빠르게 파악 가능

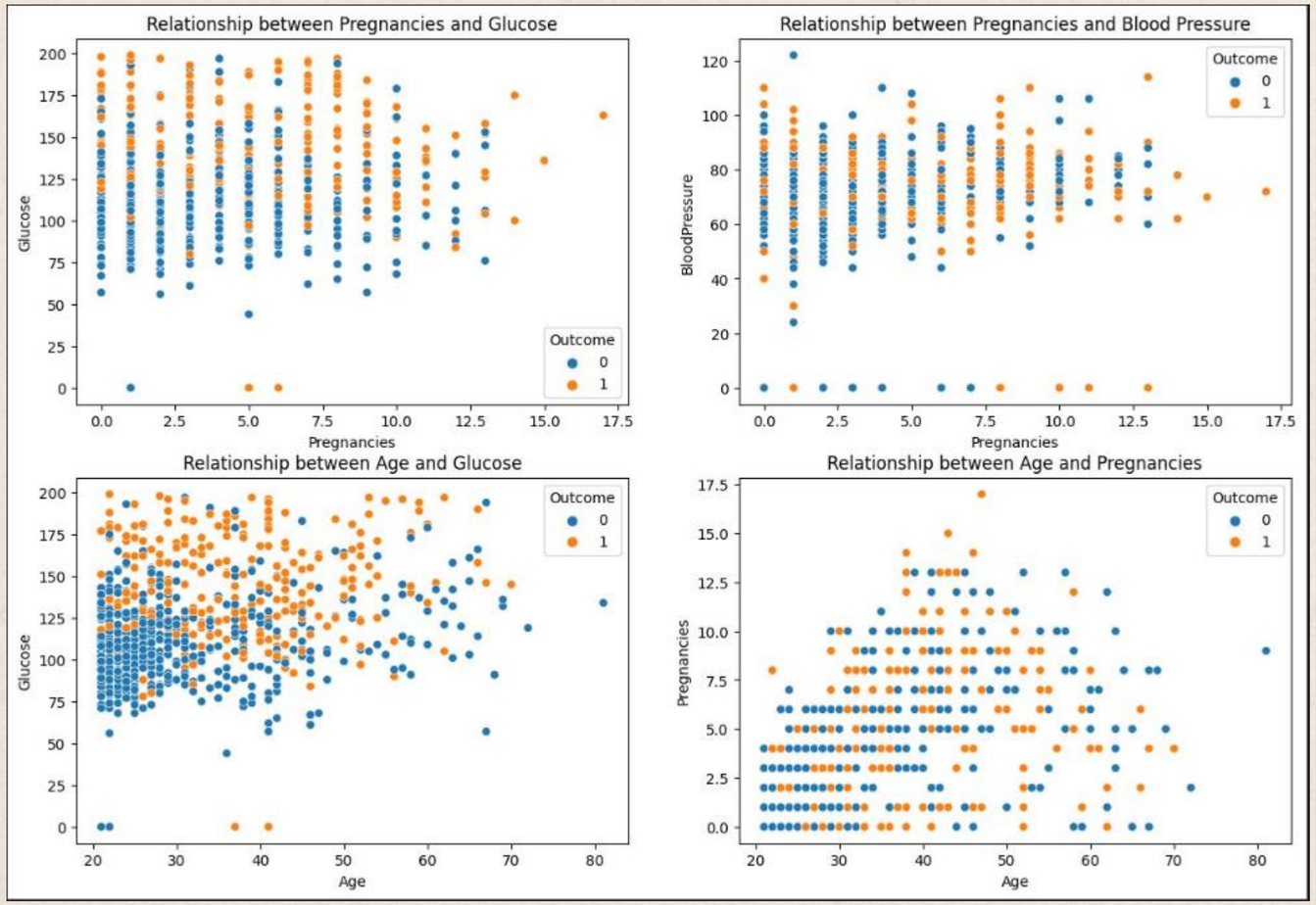
03 데이터 분석

```
fug, axes = plt.subplots(2, 2, figsize = (15, 10))
axes = axes.flatten()

sns.scatterplot(ax = axes[0],
                x = 'Pregnancies',
                y = 'Glucose',
                hue = 'Outcome',
                data = df).set(title = 'Relationship between Pregnancies and Glucose')

sns.scatterplot(ax = axes[1],
                x = 'Pregnancies',
                y = 'BloodPressure',
                hue = 'Outcome',
                data = df).set(title = 'Relationship between Pregnancies and Blood Pressure')

sns.scatterplot(ax = axes[2],
                x = 'Age',
                y = 'Glucose',
                hue = 'Outcome',
                data = df).set(title = 'Relationship between Age and Glucose')

sns.scatterplot(ax = axes[3],
                x = 'Age',
                y = 'Pregnancies',
                hue = 'Outcome',
                data = df).set(title = 'Relationship between Age and Pregnancies')
```

눈여겨볼만한 상관관계 5개를 추출 및 분석

임신 횟수와 혈당
임신 횟수와 혈압
나이와 혈당
나이와 임신 횟수
혈당과 인슐린

03 데이터 분석



1. 임신 횟수와 혈당의 상관관계

점들의 결과값이 상단으로 향하며
결과값의 차이가 명확

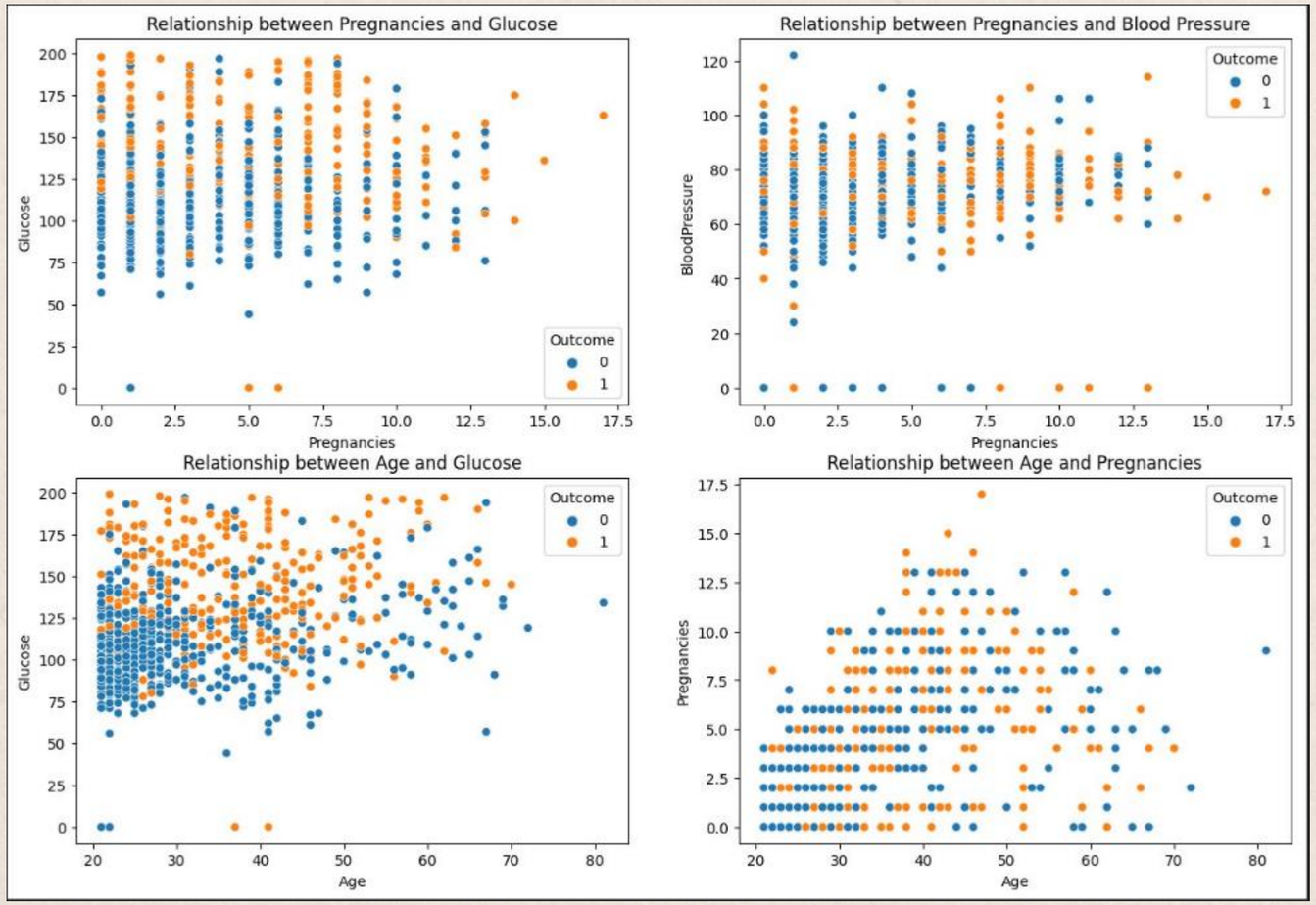
혈당이 높을수록 임신 횟수가 적고
많음에 관계없이 당뇨 발병 확률이 높음

2. 임신 횟수와 혈압의 상관관계

점들의 특별한 패턴이 없음

7.5회의 임신 횟수부터 결과값이 반전
이는 7.5회이상의 임신 횟수가
혈압을 높임으로써 당뇨 발병을 높임

03 데이터 분석



3. 나이와 혈당의 상관관계

30세 부터 1의 결과값의 분포가 증가

이는, 30세부터 혈당의 농도가 증가하며 당뇨의 발병을 높임

4. 나이와 임신 횟수의 상관관계

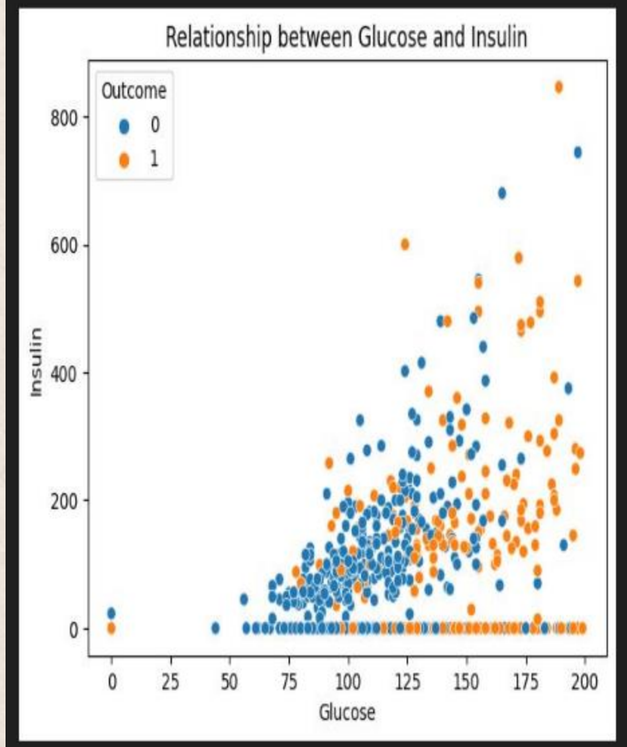
노란색 점들의 분포를 보았을때, 나이가 많아질수록 적은 임신 횟수에서의 분포가 보임

이는, 나이가 많아질수록 적은 임신 횟수 만으로도 당뇨에 취약해질 수 있음

03 데이터 분석

```
sns.scatterplot(x = 'Glucose', y = 'Insulin', hue = 'Outcome', data = df).set(title = 'Relationship between Glucose and Insulin')
```

```
[Text(0.5, 1.0, 'Relationship between Glucose and Insulin')]
```



5. 혈당과 인슐린의 상관관계

혈당이 높아질수록 인슐린의 농도는 올라감

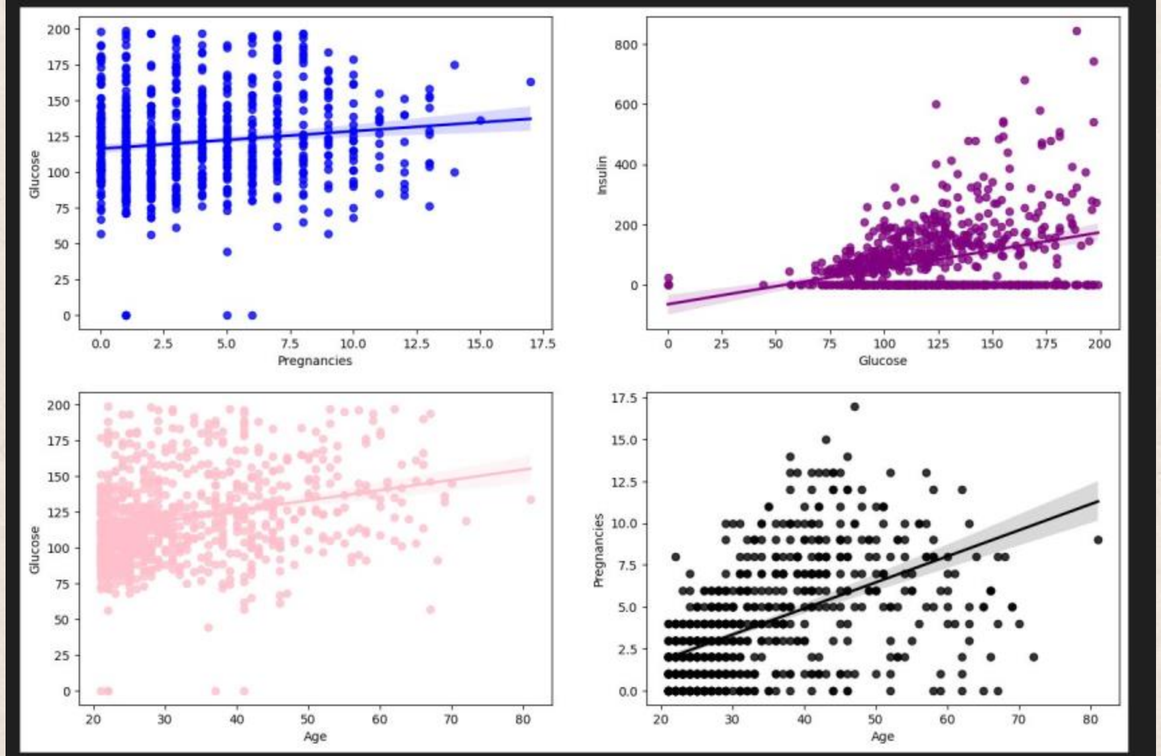
이때, 인슐린의 농도가 낮더라도 정상적인
혈당을 가졌다면 당뇨에 면역을 가질 수 있음

하지만, 혈당이 높을수록 인슐린의 농도가
높고 낮음에 관계없이 당뇨에 취약한 못습을 보임

03 데이터 분석

```
fig, axes = plt.subplots(2, 2, figsize = (15, 10))
axes = axes.flatten()

sns.regplot(ax = axes[0], x = 'Pregnancies', y = 'Glucose', data = df, color = 'blue')
sns.regplot(ax = axes[1], x = 'Glucose', y = 'Insulin', data = df, color = 'purple')
sns.regplot(ax = axes[2], x = 'Age', y = 'Glucose', data = df, color = 'pink')
sns.regplot(ax = axes[3], x = 'Age', y = 'Pregnancies', data = df, color = 'black')
```

```
<Axes: xlabel='Age', ylabel='Pregnancies'>
```



앞서 살펴본 두 변수간의 산점도를 포함한 선형 회귀 선

기울기의 의미

1. 두 변수 간의 선형 관계
2. 두 변수 간의 양의 상관 관계
3. 두 변수 간의 관계를 시각적으로 확인
4. 두 변수 간의 상호작용 확인

03 데이터 분석

```
plt.figure(figsize = [20, 10], facecolor = 'white')
sns.heatmap(df.corr(), cmap = 'magma', annot = True, linewidths = 2)
```

```
<Axes: >
```

04 모델 학습

```
X = df.drop('Outcome', axis = 1)
y = df['Outcome']
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, shuffle = True, random_state = 1)
```

모델 학습을 위해 데이터를 학습 세트와 테스트 세트로 분할

모델을 학습할 때에는 학습 세트, 학습된 모델의 성능을 평가 할때는 테스트 세트를 활용

일반화 성능 확인

04 모델 학습

```
from sklearn.preprocessing import MinMaxScaler
norm = MinMaxScaler(feature_range = (0, 1))
norm.fit(X_train)
X_train = norm.transform(X_train)
X_test = norm.transform(X_test)
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import accuracy_score
```

머신러닝 모델에 입력 데이터를 제공하기 전 데이터를 정규화

그리드 서치를 활용해 모델에 대한 매개변수 그리드를 설정, 최적의 매개변수를 찾기 위한 교차 검증을 수행

accuracy_score 함수를 통한 테스트 세트의 정확도 평가

05 모델링 로지스틱 회귀

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
prediction = model.predict(X_test)
```

```
cm = confusion_matrix(y_test, prediction)
accuracy_lr = accuracy_score(y_test, prediction)
print('Confusion Matrix:\n', cm)
print('Accuracy: ', accuracy_lr)
```

```
Confusion Matrix:
 [[90  9]
 [27 28]]
Accuracy:  0.7662337662337663
```

로지스틱 회귀 모델을 생성, 학습 데이터를 통해
모델을 훈련, 테스트 데이터를 사용해 예측을 수행

정확도 : 약 76%

05 모델링 K-최근접 이웃 분류기

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
```

```
knn_params = {'n_neighbors': np.arange(1, 100),
              'weights': ['uniform', 'distance'],
              'leaf_size': [25, 30, 25]}

knn_cv_model = GridSearchCV(knn, knn_params, cv = 10)
knn_cv_model.fit(X_train, y_train)
```

```
GridSearchCV
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'leaf_size': [25, 30, 25],
                         'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]),
                         'weights': ['uniform', 'distance']})
```

k-최근접 이웃 분류기 모델의 인스턴스를 생성

그리드 서치를 수행하여 최적의 파라미터를 찾기 위한 교차검증

05 모델링 K-최근접 이웃 분류기

```
print('Best score for train set: ', str(knn_cv_model.best_score_))
print('Best k-value: ', knn_cv_model.best_params_['n_neighbors'])
print('Best weight: ', knn_cv_model.best_params_['weights'])
print('Best leaf size: ', knn_cv_model.best_params_['leaf_size'])
```

```
Best score for train set:  0.7474352194606029
Best k-value:  49
Best weight:  uniform
Best leaf size:  25
```

그리드 서치를 통해 찾은 최적의 하이퍼파라미터와 해당 설정에서의 최상의 성능에 관한 정보 출력

```
knn_model = KNeighborsClassifier(n_neighbors = knn_cv_model.best_params_['n_neighbors'],
                                 weights = knn_cv_model.best_params_['weights'],
                                 leaf_size = knn_cv_model.best_params_['leaf_size'])

knn_model.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(leaf_size=25, n_neighbors=49)
```

모델을 초기화, 학습데이터를 사용하여 모델을 학습

05 모델링 K-최근접 이웃 분류기

```
pred = knn_model.predict(X_test)
accuracy_knn = accuracy_score(y_test, pred)
accuracy_knn
```

```
0.7662337662337663
```

정확도 : 약 76%

05 모델링 그래디언트 부스팅 분류기

```
from sklearn.ensemble import GradientBoostingClassifier
gbm = GradientBoostingClassifier()
```

```
gbm_params = {'learning_rate': [0.005, 0.008, 0.1, 0.15],
              'n_estimators': [80, 100, 150, 200],
              'max_depth': [2, 3, 4],
              'min_samples_split': [2, 3, 4]}

gbm_cv_model = GridSearchCV(gbm, gbm_params, cv = 10, n_jobs = -1)
gbm_cv_model.fit(X_train, y_train)
```

```
GridSearchCV
GridSearchCV(cv=10, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'learning_rate': [0.005, 0.008, 0.1, 0.15],
                         'max_depth': [2, 3, 4], 'min_samples_split': [2, 3, 4],
                         'n_estimators': [80, 100, 150, 200]})
```

그래디언트 부스팅 분류기 모델의 인스턴스 생성

그리드 서치를 수행하여 최적의 파라미터를 찾기 위한 교차검증

05 모델링 그래디언트 부스팅 분류기

```
print("Best score for train set: " + str(gbm_cv_model.best_score_))
print("best learning_rate value: " + str(gbm_cv_model.best_params_["learning_rate"]),
    "\nbest n_estimators value: " + str(gbm_cv_model.best_params_["n_estimators"]),
    "\nbest max_depth value: " + str(gbm_cv_model.best_params_["max_depth"]),
    "\nbest min_samples_split value: " + str(gbm_cv_model.best_params_["min_samples_split"]))
```

```
Best score for train set: 0.767160232681121
best learning_rate value: 0.1
best n_estimators value: 100
best max_depth value: 2
best min_samples_split value: 2
```

그리드 서치를 통해 찾은 최적의 하이퍼파라미터와 해당 설정에서의 최상의 성능에 관한 정보 출력

```
gbm = GradientBoostingClassifier(learning_rate = gbm_cv_model.best_params_["learning_rate"],
                                 max_depth = gbm_cv_model.best_params_["max_depth"],
                                 n_estimators = gbm_cv_model.best_params_["n_estimators"],
                                 min_samples_split = gbm_cv_model.best_params_["min_samples_split"])

gbm_model = gbm.fit(X_train, y_train)
y_pred = gbm_model.predict(X_test)
accuracy_gbm = accuracy_score(y_test, y_pred)
accuracy_gbm
```

모델을 초기화, 학습데이터를 사용하여 모델을 학습

```
0.8116883116883117
```

정확도 : 약 81%

05 모델링 LGBM 분류기

```
from lightgbm import LGBMClassifier
lgbm = LGBMClassifier()
```

```
lgbm_params = {"n_estimators": [80, 100, 120, 150, 200],
               "max_depth": [-1, 3, 4],
               "learning_rate": [0.05, 0.08, 0.1, 0.12, 0.15, 0.18],
               "min_child_samples": [15, 20, 25, 30]}
```

```
lgbm_cv_model = GridSearchCV(lgbm, lgbm_params, cv = 10, n_jobs = -1)
lgbm_cv_model.fit(X_train, y_train)
```

```
GridSearchCV
GridSearchCV(cv=10, estimator=LGBMClassifier(), n_jobs=-1,
             param_grid={'learning_rate': [0.05, 0.08, 0.1, 0.12, 0.15, 0.18],
                         'max_depth': [-1, 3, 4],
                         'min_child_samples': [15, 20, 25, 30],
                         'n_estimators': [80, 100, 120, 150, 200]})
```

LGBM 분류기 모델의 인스턴스 생성

그리드 서치를 수행하여 최적의 파라미터를 찾기 위한 교차검증

05 모델링 LGBM 분류기

```
print("Best score for train set: " + str(lgbm_cv_model.best_score_))
print("best learning_rate value: " + str(lgbm_cv_model.best_params_["learning_rate"]),
      "\nbest n_estimators value: " + str(lgbm_cv_model.best_params_["n_estimators"]),
      "\nbest max_depth value: " + str(lgbm_cv_model.best_params_["max_depth"]),
      "\nbest min_child_samples value: " + str(lgbm_cv_model.best_params_["min_child_samples"]))
```

```
Best score for train set: 0.7573506081438393
best learning_rate value: 0.05
best n_estimators value: 80
best max_depth value: 3
best min_child_samples value: 15
```

```
lgbm = LGBMClassifier(learning_rate = lgbm_cv_model.best_params_["learning_rate"],
                      max_depth = lgbm_cv_model.best_params_["max_depth"],
                      n_estimators = lgbm_cv_model.best_params_["n_estimators"],
                      min_child_samples = lgbm_cv_model.best_params_["min_child_samples"])

lgbm_model = lgbm.fit(X_train, y_train)
y_pred = lgbm_model.predict(X_test)
accuracy_lgbm = accuracy_score(y_test, y_pred)
accuracy_lgbm
```

```
0.7922077922077922
```

그리드 서치를 통해 찾은 최적의
하이퍼파라미터와 해당 설정에서의
최상의 성능에 관한 정보 출력

모델을 초기화, 학습데이터를 사용하여
모델을 학습

정확도 : 약 79%

05 모델링

```
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression',
              'Gradient Boosting Classifier', 'LGBM'],
    'Score': [accuracy_knn, accuracy_lr, accuracy_gbm, accuracy_lgbm]})

sorted_model = models.sort_values(by='Score', ascending=False)

sorted_model
```

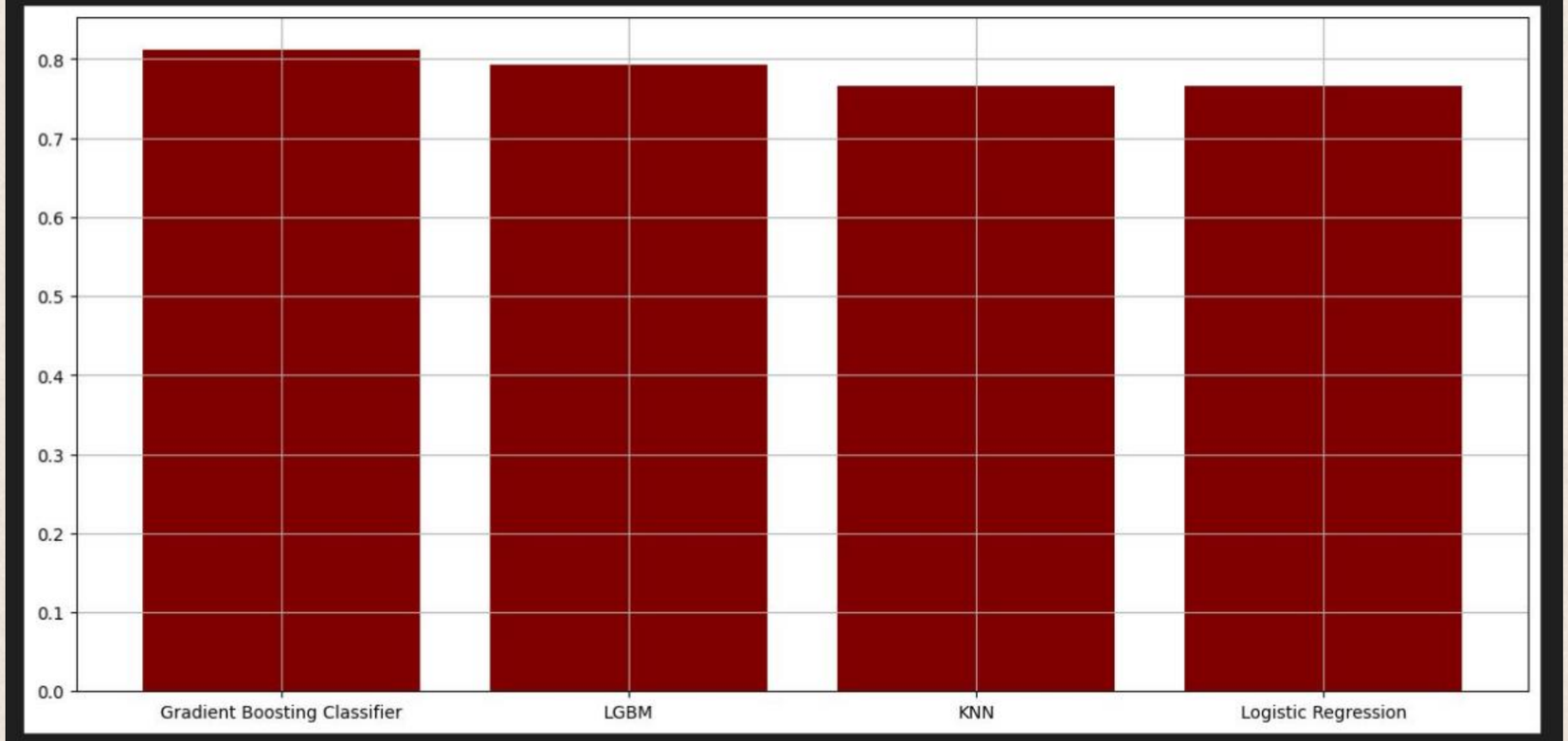
	Model	Score
2	Gradient Boosting Classifier	0.811688
3	LGBM	0.792208
0	KNN	0.766234
1	Logistic Regression	0.766234

모델의 정확도를 데이터프레임에 저장

정확도에 따라 내림차순 정렬

05 모델링

```
plt.figure(figsize=(15, 7))

fig = plt.bar(sorted_model['Model'], sorted_model['Score'],color='maroon')

plt.grid()
plt.show()
```



06 결론

다양한 모델 중 그래디언트 부스팅 모델을 이용한 예측이 가장 적합함

나이, 임신 횟수, 혈당은 당뇨 발병 확률과 많은 연관이 있음

- 임신 횟수가 많을수록 혈압을 높여 당뇨에 직접적인 연관을 줌 -

- 혈당의 농도가 높을수록 인슐린 농도에 관계없이 당뇨 발병을 높임
이는 30세 이상의 나이가 가장 취약시기 -

- 나이가 많아질수록 임신 횟수에 관계없이 당뇨에 취약함 -

06 결론

데이터 분석 과제를 통해서 다양한 변수들의 상관관계를 해석하는데
많은 어려움이 있었습니다.

앞으로 머신러닝을 통해 데이터를 해석하고 숨은 특성을 찾아내는 능력을
키우는데 많은 도움이 된 것 같고, 어떻게 앞으로 나아가야 할지
명확하게 알 수 있는 기회가 된 것 같습니다.

인공지능에 많은 흥미를 가지고 좀 더 다양한 분야에 도전해 봐야겠다는
생각도 들고 앞으로 많이 노력해야겠다는 마음가짐도 가지게 되었습니다