인공지능을 활용한 당뇨병 판별

김재현

Data Set

Pima Indians Diabetes Database

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.000000	35	30.5	33.6	0.627	50	1
1	1	85.0	66.000000	29	30.5	26.6	0.351	31	0
2	8	183.0	64.000000	23	30.5	23.3	0.672	32	1
3	1	89.0	66.000000	23	94.0	28.1	0.167	21	0
4	0	137.0	40.000000	35	168.0	43.1	2.288	33	1
5	5	116.0	74.000000	23	30.5	25.6	0.201	30	0
6	3	78.0	50.000000	32	88.0	31.0	0.248	26	1
7	10	115.0	69.105469	23	30.5	35.3	0.134	29	0
8	2	197.0	70.000000	45	543.0	30.5	0.158	53	1
9	8	125.0	96.000000	23	30.5	32.0	0.232	54	1

미국 국립 당뇨병 연구소(National Institute of Diabetes and Digestive and Kidney Diseases)에서 환자에게 당뇨병이 있는지 여부를 예측하기 위해 수집한 데이터로 수집된 데이터는 21세 이상의 피마 인디언 혈통의 여성

Data Set Colum

Columns

Column 명	내용	자료형
Pregnancies	임신 횟수	int64
Gluose	포도당 부하 검사 수치	int64
BloodPressure	혈압(mm Hg)	int64
SkinThickness	팔 삼두근 뒤쪽의 피하지방 측정값(mm)	int64
BMI	혈청 인슐린(mu U/ml)	float64
Insulin	체질량지수(체중(kg)/키(m))^2	int64
DiabetesPedigreeFunction	당뇨 내력 가중치 값	float64
Age	나이	int64
Outcome	당뇨병 여부	int64

Missing Value Analysis

isnull().sum()

```
# 결측값(Missing value) 확인
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
```

확인 결과 결측값은 없는것을 확인할 수 있다.

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

포도당 수치, 혈압, 피하지방 측정값, 인슐린 수치, BMI의 최솟값을 보면 살아있는 사람이라면 나올 수 없는 0이라는 값을 가지는 것을 확인할 수 있다.

Missing Value Analysis

데이터 분석

```
# 확인 결과 결측값은 없지만 describe()의 표를 보면 포도당 수치, 혈압, BMI등의 0이 나올 수 없는 값을 가진 칼럼이 있는 것을 확인할 수 있다.
zero_features = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
total_count = df['Glucose'].count()

for feature in zero_features:
    zero_count = df[df[feature]==0][feature].count()
    print('{0} 의 값이 0인 건수는 {1}, 전체데이터의 {2:.2f}% 입니다.'.format(feature, zero_count, 100*zero_count/total_count))
```

```
Glucose 의 값이 0인 건수는 5, 전체데이터의 0.65% 입니다.
BloodPressure 의 값이 0인 건수는 35, 전체데이터의 4.56% 입니다.
SkinThickness 의 값이 0인 건수는 227, 전체데이터의 29.56% 입니다.
Insulin 의 값이 0인 건수는 374, 전체데이터의 48.70% 입니다.
BMI 의 값이 0인 건수는 11, 전체데이터의 1.43% 입니다.
```

Missing Value Analysis

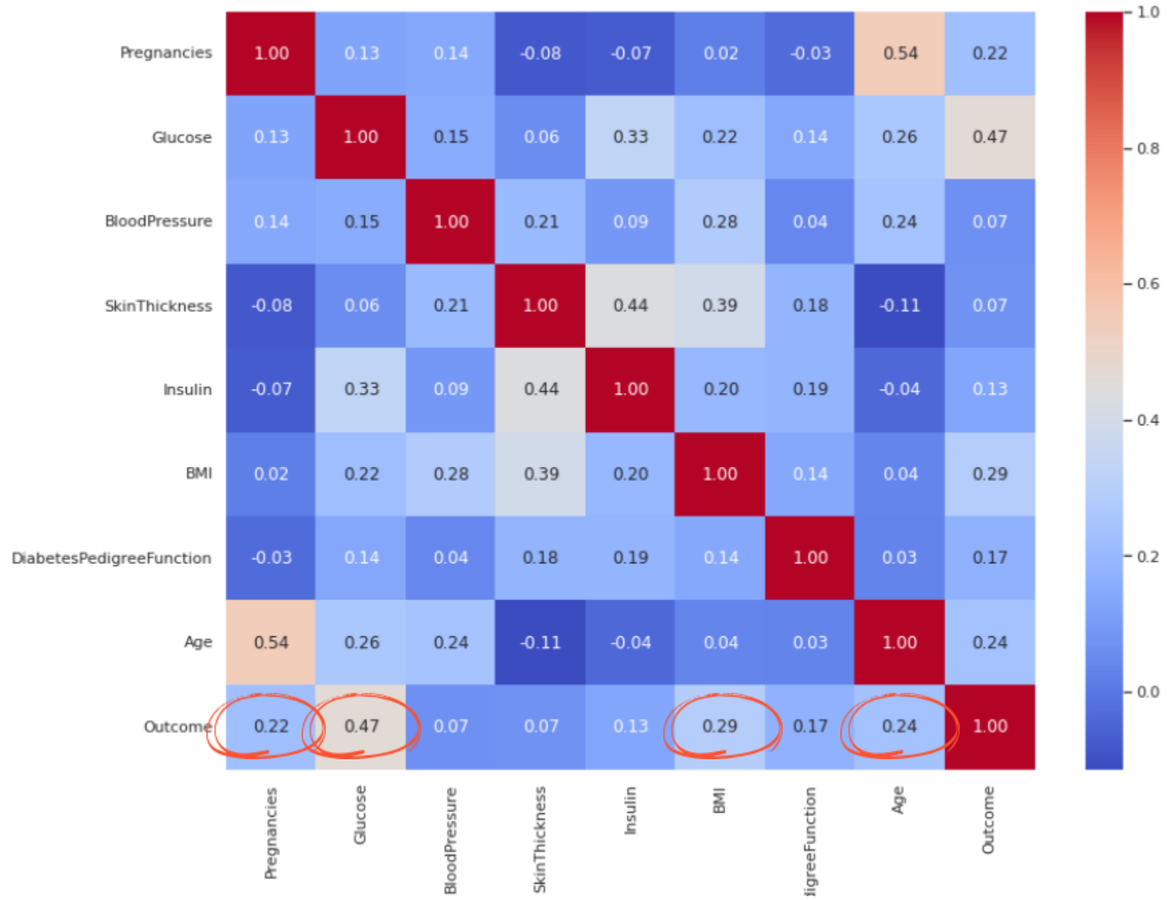
replace()

```
# 포도당 수치가 0인 값들을 평균값으로 변경
df['Glucose'] = df['Glucose'].replace(0, df['Glucose'].mean())

# 혈압 수치가 0인 값들을 평균값으로 변경
df[df['BloodPressure'] == 0]['BloodPressure'].value_counts()
df['BloodPressure'] = df['BloodPressure'].replace(0,df['BloodPressure'].mean())

# BMI수치가 0인 값들을 중앙값으로 변경
df[df['BMI'] == 0]['BMI'].value_counts()
df['BMI'] = df['BMI'].replace(0, df['BMI'].median())

# 팔 삼두근 뒤쪽의 피하지방 측정값이 0인 값들을 중앙값으로 변경
df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].median())

# 인슐린 수치가 0인 값들을 중앙값으로 변경
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())
```

 # 0인 값들을 중앙값 / 평균값으로 대치

Exploratory Data Analysis

heatmap



 # 각 변수간의 연관관계를 파악하고 어떤 특성이 결과에 얼마나 영향을 주는지 파악한다.

Modeling

데이터 스케일링

```
# 데이터 스케일링
# QuantileTransformer : sklearn에서 제공하는 스케일러, 데이터를 1000개 분위로 나눈 후 0~1 사이에 고르게 분포시키는 방식
q  = QuantileTransformer()
X = q.fit_transform(df)
transformedDF = q.transform(X)

transformedDF = pd.DataFrame(X)
transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.746728	0.812173	0.518979	0.804974	0.255890	0.593586	0.752618	0.889398	1.0
1	0.230366	0.091623	0.290576	0.645942	0.255890	0.214005	0.476440	0.556937	0.0
2	0.863220	0.956806	0.234293	0.358639	0.255890	0.077880	0.784031	0.582461	1.0
3	0.230366	0.125654	0.290576	0.358639	0.662958	0.285340	0.106675	0.000000	0.0
4	0.000000	0.723168	0.005236	0.804974	0.834424	0.929319	0.998691	0.604712	1.0

 # 학습데이터를 모델에 입력하기전 특성별로 데이터 스케일이 다르기 때문에

모든 특성의 범위를 같게 만들어준다.

Modeling

Data Splitting

```
# 학습데이터와 테스트 데이터 분리
features = df.drop(["Outcome"], axis=1)
labels = df["Outcome"]

x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.30, random_state=7)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
547	4	131.0	68.0	21	166.0	33.1	0.160	28
580	0	151.0	90.0	46	30.5	42.1	0.371	21
405	2	123.0	48.0	32	165.0	42.1	0.520	26
206	8	196.0	76.0	29	280.0	37.5	0.605	57
492	4	99.0	68.0	38	30.5	32.8	0.145	33

〈X_train〉

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
236	7	181.0	84.0	21	192.0	35.9	0.586	51
715	7	187.0	50.0	33	392.0	33.9	0.826	34
766	1	126.0	60.0	23	30.5	30.1	0.349	47
499	6	154.0	74.0	32	193.0	29.3	0.839	39
61	8	133.0	72.0	23	30.5	32.9	0.270	39

〈Y_train〉

모델을 학습하기위한 학습데이터(X_train)와
학습한 모델의 정확도를 확인하기 위한 (Y_train으로 분리)

Modeling

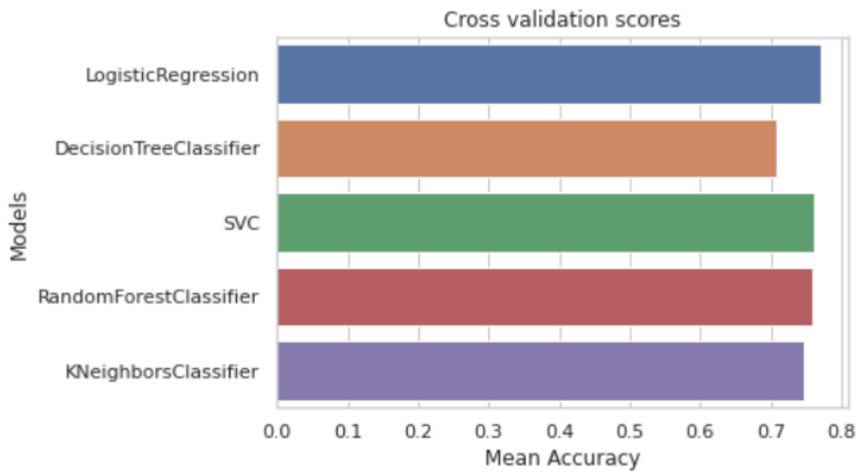
모델 성능 평가

```python
# 모델 평가
def evaluate_model(models):

  # 계층별 k-겹 교차검증을 이용한 모델 평가
    kfold = StratifiedKFold(n_splits = 10)

    result = []
    for model in models :
        result.append(cross_val_score(estimator = model, X = x_train, y = y_train, scoring = "accuracy", cv = kfold, n_jobs=4))

    cv_means = []
    cv_std = []
    for cv_result in result:
        cv_means.append(cv_result.mean())
        cv_std.append(cv_result.std())

    result_df = pd.DataFrame({
        "CrossValMeans":cv_means,
        "CrossValerrors": cv_std,
        "Models":[
            "LogisticRegression", # 로지스틱 회귀
            "DecisionTreeClassifier", # 의사결정트리
            "SVC", # SVC
            "RandomForestClassifier", # 랜덤 포레스트
            "KNeighborsClassifier" # KNN
        ]
    })

    # 차트 생성
    bar = sns.barplot(x = "CrossValMeans", y = "Models", data = result_df, orient = "h")
    bar.set_xlabel("Mean Accuracy")
    bar.set_title("Cross validation scores")
    return result_df
```

Modeling

모델 성능 평가

```
# 모델링 단계에서의 서로다른 알고리즘 테스트

random_state = 30
models = [
    LogisticRegression(random_state = random_state, solver='liblinear'),
    DecisionTreeClassifier(random_state = random_state),
    SVC(random_state = random_state),
    RandomForestClassifier(random_state = random_state),
    KNeighborsClassifier(),
]
evaluate_model(models)
```



LogisticRegression	0.770964
DecisionTreeClassifier	0.707687
SVC	0.761670
RandomForestClassifier	0.759748
KNeighborsClassifier	0.746506

Modeling

Hyperparameter Tuning

```python
def analyze_grid_result(grid_result):
    '''
    Analysis of GridCV result and predicting with test dataset
    Show classification report at last
    '''
    # Best parameters and accuracy
    print("Tuned hyperparameters: (best parameters) ", grid_result.best_params_)
    print("Accuracy :", grid_result.best_score_)

    means = grid_result.cv_results_["mean_test_score"]
    stds = grid_result.cv_results_["std_test_score"]
    for mean, std, params in zip(means, stds, grid_result.cv_results_["params"]):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
    print()
    print("")
    print("Detailed classification report:")
    y_true, y_pred = y_test, grid_result.predict(x_test)
    print(classification_report(y_true, y_pred))
    print()
```

 # 머신러닝 학습을 할 때 더 효과가 좋도록 하는 주 변수가 아닌 자동 설정 되는 변수를 의미 한다. 즉 최적의 변수을 세팅하여 학습을 한다는 것이다.

Modeling

LogisticRegression

```
model = LogisticRegression(solver='liblinear')
solvers = ['newton-cg', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]

grid = dict(solver = solvers, penalty = penalty, C = c_values)
cv = StratifiedKFold(n_splits = 50, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = grid, cv = cv, scoring = 'accuracy', error_score = 0)

logi_result = grid_search.fit(x_train, y_train)

analyze_grid_result(logi_result)
```

```
Tuned hyperparameters: (best parameters)  {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
Accuracy : 0.774909090909091
0.773 (+/-0.241) for {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.773 (+/-0.241) for {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.241) for {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.775 (+/-0.226) for {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.240) for {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.773 (+/-0.224) for {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.242) for {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722 (+/-0.226) for {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.764 (+/-0.245) for {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.687 (+/-0.256) for {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
```

Modeling

SVC

```
model = SVC()

tuned_parameters = [
    {"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},
    {"kernel": ["linear"], "C": [1, 10, 100, 1000]},
]
cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = tuned_parameters, cv = cv, scoring = 'accuracy', error_score = 0)
scv_result = grid_search.fit(x_train, y_train)

analyze_grid_result(scv_result)
```

```
Tuned hyperparameters: (best parameters)  {'C': 100, 'kernel': 'linear'}
Accuracy : 0.7765286023414526
0.715 (+/-0.[redacted] for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.747 (+/-0.023) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.670 (+/-0.020) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.737 (+/-0.003) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.663 (+/-0.005) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.728 (+/-0.036) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.648 (+/-0.002) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.680 (+/-0.031) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.762 (+/-0.001) for {'C': 1, 'kernel': 'linear'}
0.769 (+/-0.021) for {'C': 10, 'kernel': 'linear'}
0.777 (+/-0.008) for {'C': 100, 'kernel': 'linear'}
0.764 (+/-0.003) for {'C': 1000, 'kernel': 'linear'}
```

Modeling

RandomForestClassifier

```
model = RandomForestClassifier(random_state=42)
# Define grid search
tuned_parameters = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = tuned_parameters, cv = cv, scoring = 'accuracy', error_score = 0)
grid_result = grid_search.fit(x_train, y_train)

# SVC Hyperparameter Result
analyze_grid_result(grid_result)
```

```
Tuned hyperparameters: (best parameters)  {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 200}
Accuracy : 0.7765008600122066
0.765 (+/-0.031) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 200}
0.773 (+/-0.023) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 500}
0.765 (+/-0.031) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 200}
0.773 (+/-0.023) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 500}
0.775 (+/-0.019) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 200}
0.769 (+/-0.023) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 500}
0.769 (+/-0.046) for {'criterion': 'gini', 'max_depth': 5, 'max_features': 'auto', 'n_estimators': 200}
```

Modeling

최종 결과

```
y_pred = logi_result.predict(x_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.79	0.88	0.83	147
1	0.74	0.58	0.65	84
accuracy			0.77	231
macro avg	0.77	0.73	0.74	231
weighted avg	0.77	0.77	0.77	231

느낀점

1. Scikit-learn

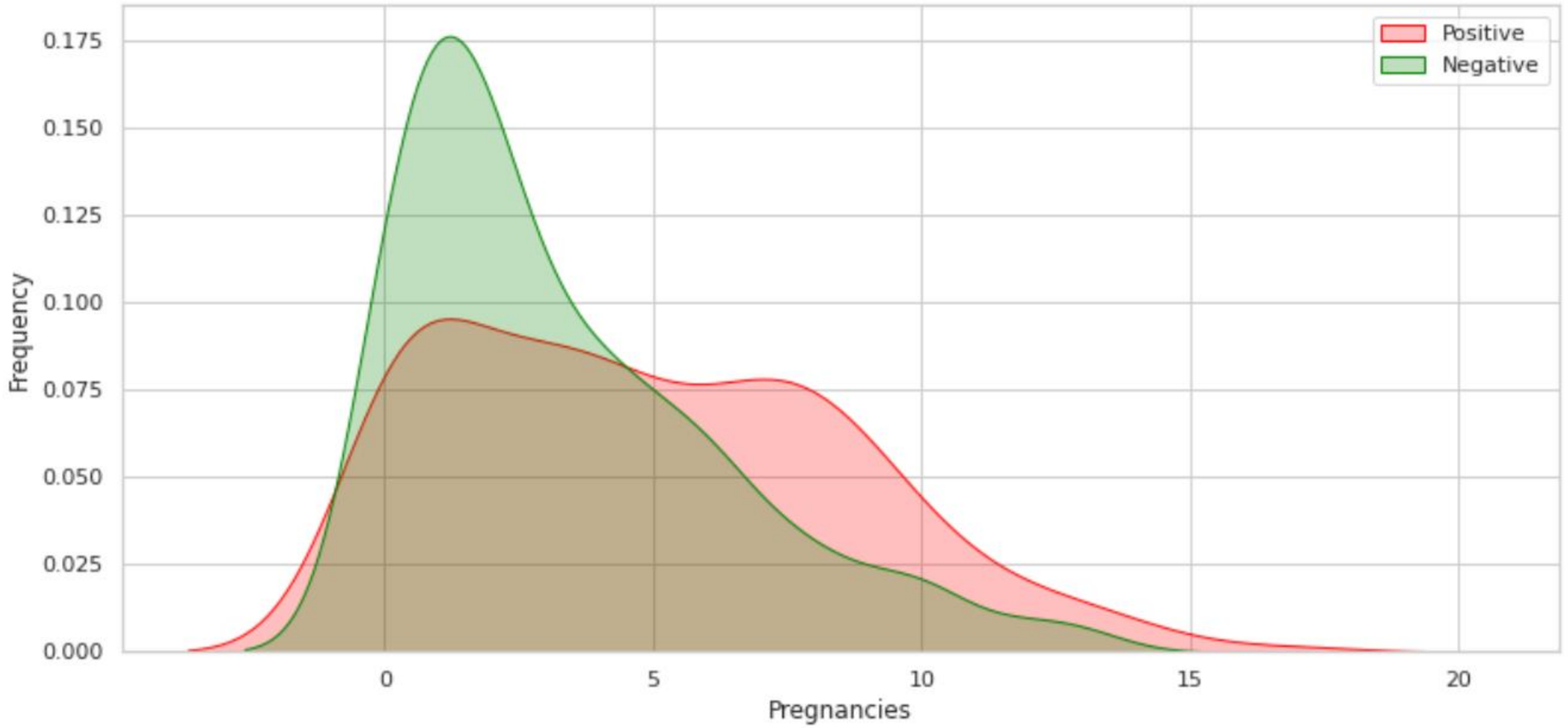
Scikit-learn은 생각했던 것 보다 훨신 더 다양한 기능을 지원하며 이 기능들을 이해하고 사용할 수 있다면 어렵지않게 훌륭한 인공지능을 만들 수 있을 것 같다.

2. 데이터 분석의 어려움

Keggle에 올라온 대부분의 오픈된 소스에서 데이터를 분석 및 전처리를 하는 부분이 대부분을 차지한다.
그만큼 머신러닝에 있어 데이터의 속성을 세밀하게 분석하는 것은 더 정확도가 높은 인공지능을 만들 수 있을 것이라고 생각이 들었지만 이번 프로젝트에서는 데이터 분석에 있어서 부족함이 있던점이 아쉬웠다.

Exploratory Data Analysis

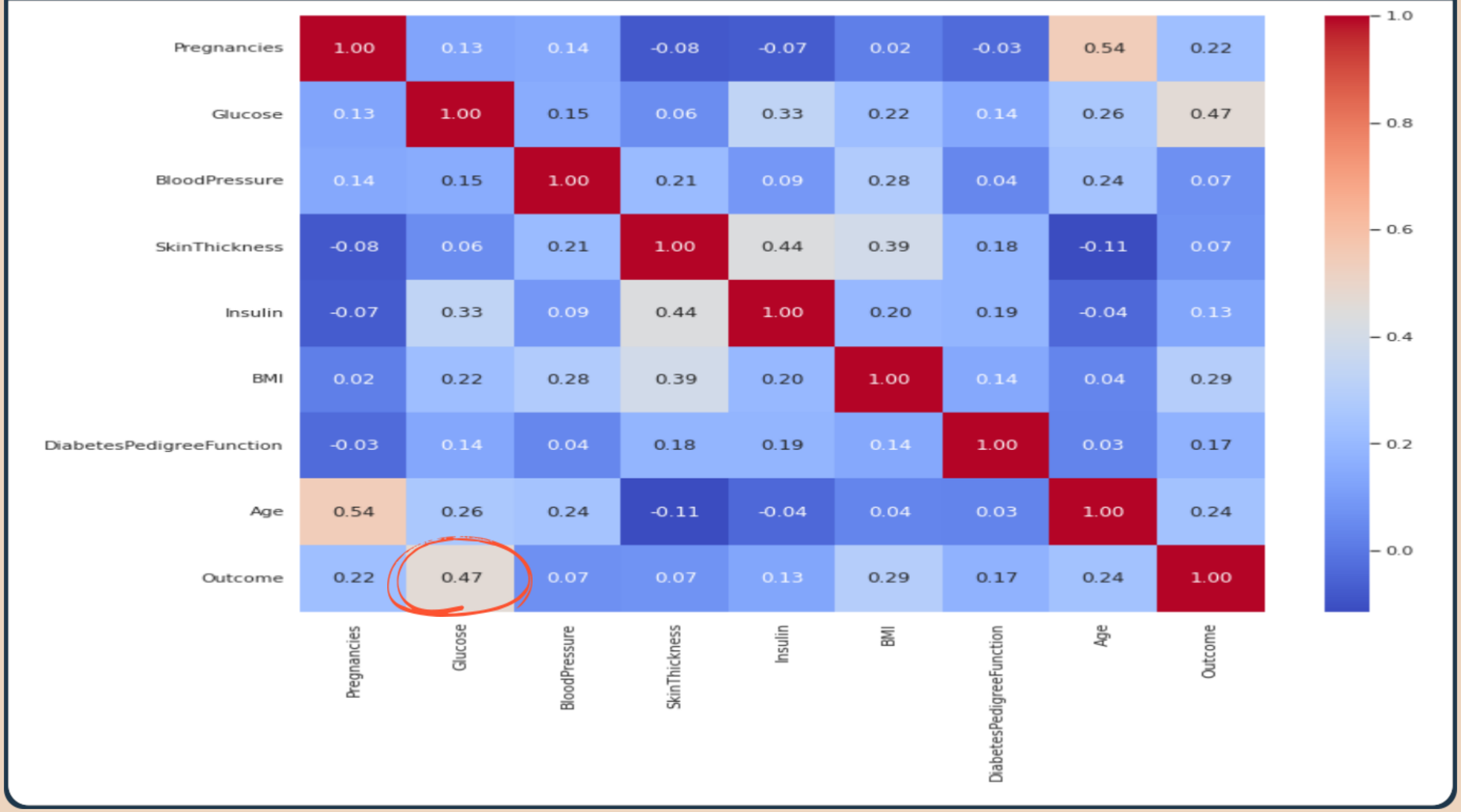
임신 횟수(Pregnancies)



임신한 횟수가 많아질수록 당뇨병에 걸린것을 확인할 수 있다.

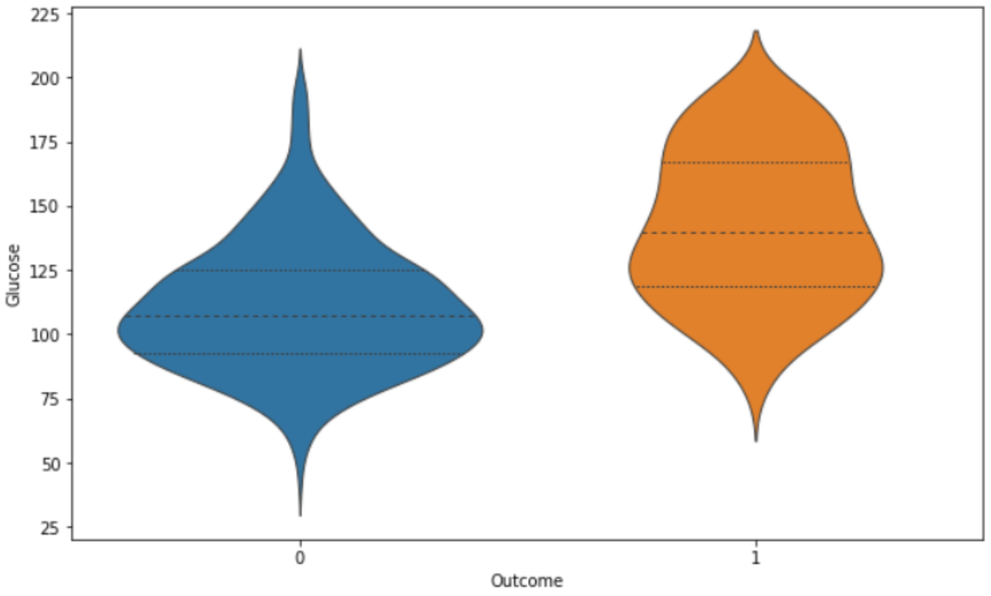
Exploratory Data Analysis

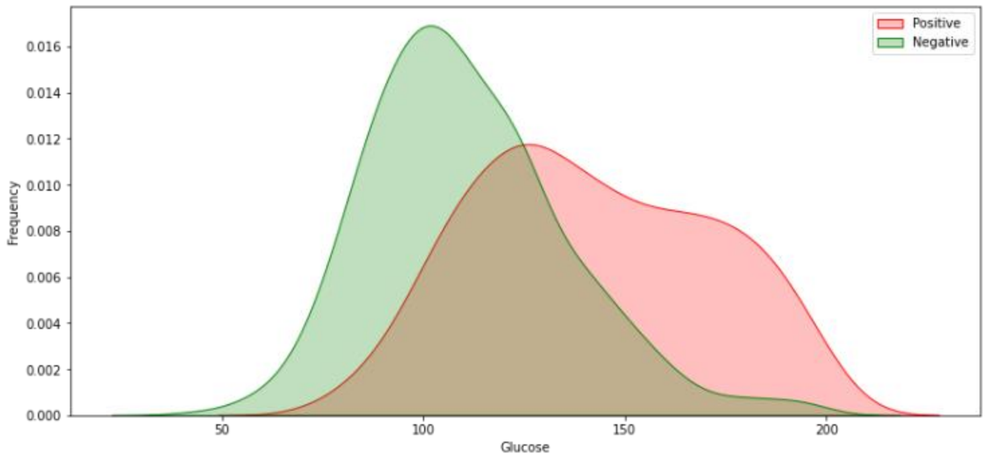
heatmap



Exploratory Data Analysis

포도당 부하 검사 수치 (Glucose)





 # 당뇨병 환자들의 포도당 수치가 걸리지 않은 사람들보다 높게 형성되어 있다.