Unsupervised Learning Diabetes

August 16, 2022

1 Section 1: Project Topic

I wanted to explore a dataset about Diabates in the Pima Indian Community. Specifically, I wanted to see if there was any correlation between any of the factors that lead to diabetes, and see relationship between these inputs as a whole.

2 Section 2: Data Source

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The purpose of the dataset is to diagnostically predict whether or not a patient has diabetes, based on various biological/health factors of a person. Several constraints were placed on the selection of these instances from a larger database- specifically, that all patients here are females at least 21 years old of Pima Indian heritage.

This dataset contains 8 medical predictor variables, and one target variable: Outcome.

Reference:

Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., & Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In Proceedings of the Symposium on Computer Applications and Medical Care (pp. 261–265). IEEE Computer Society Press.

Dataset:

https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

3 Section 3: Data Description

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import scipy as sp
import os
import scipy.stats as stats
import math
import sklearn
```

```
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.cluster import AgglomerativeClustering
from sklearn import metrics
from sklearn.model_selection import GridSearchCV,cross_val_score
```

[2]:
```
diabetes=pd.read_csv("diabetes.csv")
diabetes.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
```

[3]:
```
print(diabetes.shape)
print('Number of rows: ', diabetes.shape[0])
print('Number of independent Variables: ', diabetes.shape[1])
```

```
(768, 9)
Number of rows:  768
Number of independent Variables:  9
```

Description of all the variables:

1) Pregnancies: # of pregnancies
2) Glucose: level from 2 hour glucose tolerance test
3) Blood Pressure:Diastolic blood pressure (mm Hg)
4) Skin Thickness: Triceps skin fold thickness (mm)
5) Insulin: 2-Hour serum insulin (mu U/ml)
6) BMI: Body mass index (weight in kg/(height in m)^2)

7) Diabetes Pedigree Function: indicates the function which scores likelihood of diabetes based on family history.

8) Age: age in years

9) Outcome: 0 or 1 (target dependent variable)

[4]:
```
diabetes.describe()
```

[4]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

4 Section 4: Data Cleaning

[5]:
```
diabetes.isnull().sum()
```

[5]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Based off of this, we can see that there no null values present. In addition, all of the medical predictors seem valuable to the dataset. Therefore, I decided not to drop any of the columns from the dataset, and went ahead with my exploratory data analysis:

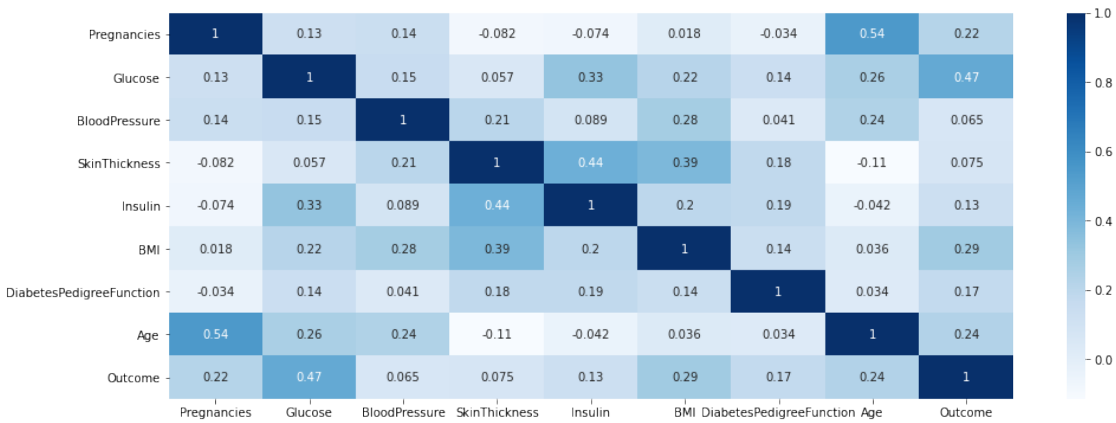
5 Section 5: Exploratory Data Analysis(EDA)

I started by creating a pairplot and a correlation heatmap in order to see the relationship between the different variables and see how closely correlated any of them are:

[6]:
```
x=sns.pairplot(diabetes, diag_kind="kde", hue="Outcome")
x.fig.set_size_inches(15,15)
```

[7]:
```
plt.figure(figsize=(16,6))
sns.heatmap(diabetes.corr(), annot = True, cmap="Blues")
```

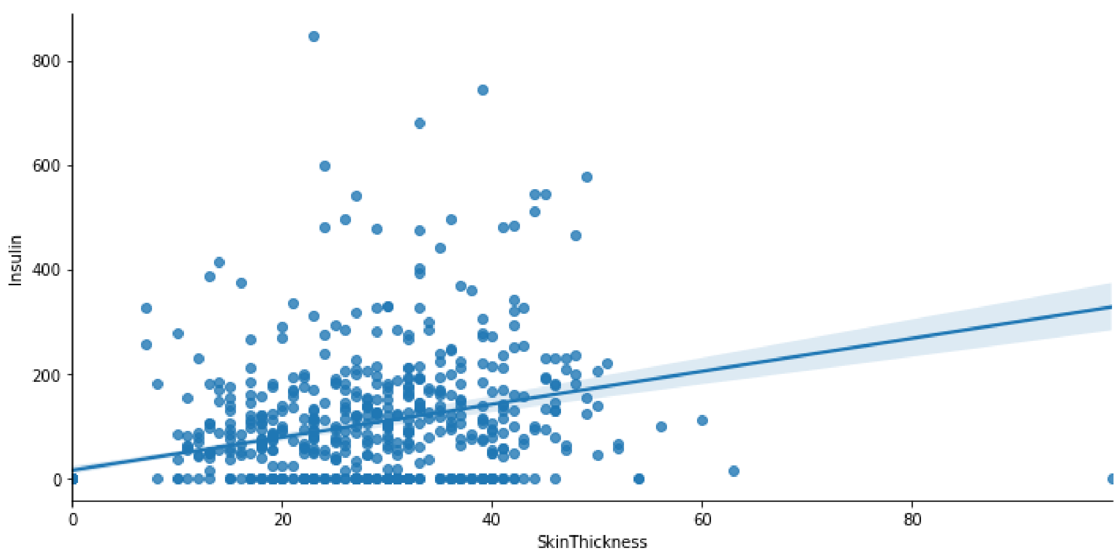
[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c7abda390>



From both visualizations, is is apparent that none of the medical predictors have a particularly strong correlation with eachother. Even though they had the strongest correlation, I thought exploring the relationship between "Age and Pregnancies" would be too unrelated to diabetes, so I explored the relationship between "skin thickness" and insulin" instead, since they had the next highest correlation:

[8]:
```
sns.lmplot(x='SkinThickness', y='Insulin', data=diabetes, aspect=2)
```

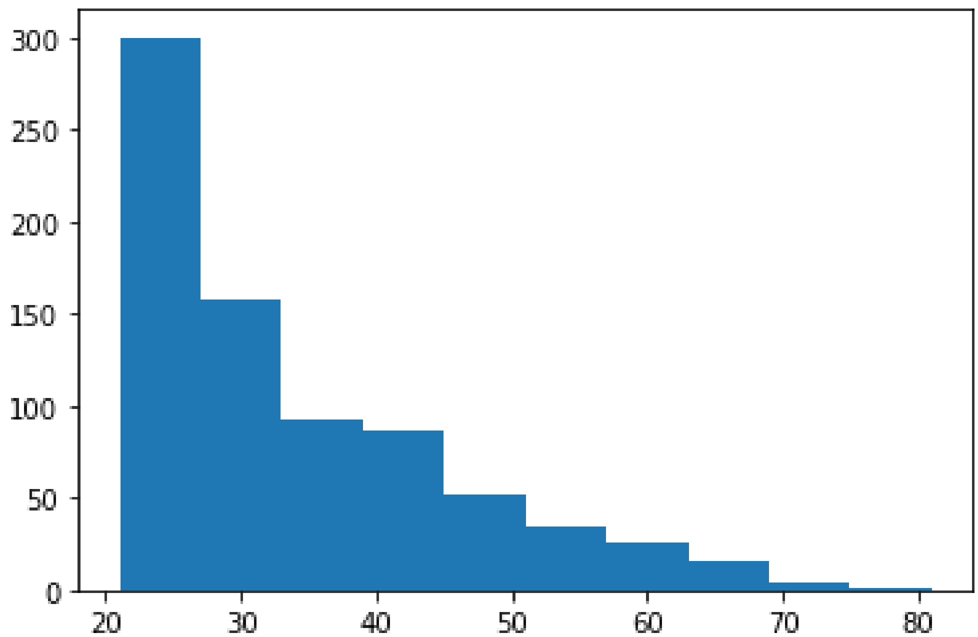
[8]: <seaborn.axisgrid.FacetGrid at 0x7f8c797e19d0>



[9]:
```
plt.hist(diabetes['Age'])
```

[9]:
```
(array([300., 157.,  92.,  86.,  52.,  34.,  26.,  16.,   4.,   1.]),
 array([21., 27., 33., 39., 45., 51., 57., 63., 69., 75., 81.]),
```

<a list of 10 Patch objects>)



6 Section 6: Model Building

Before making my models, I have to split up the data. Since the csv file does have labels, I can go ahead and use the train_test_split function to split the data into training and test sets- I will do this based off of the output variable: Outcome.

[10]:
```
columns=set(diabetes.columns)
columns.remove('Outcome')
x_reduced=diabetes[columns]
y=diabetes['Outcome']
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_reduced,y,test_size=0.
↪33,random_state=58)
```

6.1 1) K Means Clustering Model:

[11]:
```
from sklearn.cluster import KMeans
km_model=KMeans(n_clusters=2)
km_model.fit(x_train)
km_pred=km_model.predict(x_test)
if metrics.accuracy_score(km_pred,y_test)<0.5:
```

```
    zeros=np.where(km_pred==0)
    ones=np.where(km_pred==1)
    km_pred[zeros]=1
    km_pred[ones]=0
metrics.accuracy_score(km_pred,y_test)
```

[11]: 0.6417322834645669

Unfortunately this accuracy score is not great. So, I proceeded to conduct some hyperparameter tuning to see if that would increase the accuracy of the model:

[12]:
```
km_params={'algorithm':["auto", "full"],'max_iter':
 ↪[100,200,300,400,500,600],'init':['k-means++','random']}
grid=GridSearchCV(KMeans(n_clusters=2,random_state=12),km_params,scoring='accuracy',cv=3)
grid.fit(x_train,y_train)
grid.best_params_
```

[12]: {'algorithm': 'auto', 'init': 'k-means++', 'max_iter': 100}

[13]:
```
km_model=KMeans(n_clusters=2,init=grid.best_params_['init'],algorithm=grid.
 ↪best_params_['algorithm'],max_iter=grid.best_params_['max_iter'])
km_model.fit(x_train)
km_pred=km_model.predict(x_test)
if metrics.accuracy_score(km_pred,y_test)<0.5:
    zeros=np.where(km_pred==0)
    ones=np.where(km_pred==1)
    km_pred[zeros]=1
    km_pred[ones]=0
metrics.accuracy_score(km_pred,y_test)
```

[13]: 0.6417322834645669

Given that even after the hyperparameter tuning, the accuracy score was not that great, I decided to construct another unsupervised model- an Agglomerative Clustering Model:

6.2 2) Agglomerative Clustering Model:

[14]:
```
model = AgglomerativeClustering(n_clusters=2, affinity='euclidean',␣
 ↪linkage='ward')
model.fit_predict(x_test)
labels = model.labels_

sklearn.metrics.davies_bouldin_score(x_test,labels)
```

[14]: 0.7649096788847299

Instead of an accuracy score, I decided to calculate a Davies-Bouldin Score, which is commonly used with clustering models. This score has a minimum of 0, and the lower the score, the indication of better clustering. This Davies-Bouldin Score is considerably better than for the K-Means Model- however since the accuracy score didnt seem to change even with hyperparameter tuning- I decided to leave this model as is, and compare it to a supervised model instead- I went ahead and used a SUpport Vector Machine Model:

6.3 Supervised Model Comparison: SVM Model:

[19]:
```python
from sklearn.svm import SVC
svm = SVC()
svm.fit(x_train,y_train)

svm_acc= metrics.accuracy_score(y_test,svm.predict(x_test))
print(svm_acc)
```

```
0.7519685039370079
```

7 Section 7: Results/Conclusion

1) The Agglomerative Clustering Model appeared to yield better results
2) Models as a Whole: Supervised Learning Model was more accurate
3) I was not fully happy with how the models I constructed turned out- if I were to spend more time on this project, I would conduct more exploratory analysis, and see if I could reconstruct the models based off another variable, not just “Outcome”