
# USKUDAR UNIVERSITY

# FACULTY OF ENGINEERING AND NATURAL SCIENCES

# COMPUTER ENGINEERING

# 2021-2022 FALL

# FINAL PROJECT REPORT

**STUDENT ID:** 180201043

**NAME LASTNAME:** AHMET AKDEMİR

**PROJECT TITLE:** DIABETES DATASET

# 1. INTRODUCTION

I searched datasets on the internet and i select a famous data set which is

Pima Indians Diabetes Dataset for my project. In that database, there is information about 768 patients' data. In that dataset I am going to try to predict a person diabet or not, for this process my base values will be other columns.

I will do that prediction via using Python libraries there are lots of thing about machine learning and data analysis there.

# 2. MATERIALS AND METHODS

# DATASET

Pima Indians Diabetes Dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian. The data set consist information of the patients such as Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function values and also their ages and number of times pregnant if they had.

Web link of my dataset:

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

## TOOLS THAT I USED

### Python

### Keras

**Numpy**

**Pandas**

**Matplotlib**

**Seaborn**

**Sklearn**

## 3. CODES

First I import the libraries.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import keras
from keras.models import Sequential # Models Architecture import
from keras.layers import Dense # Layers of Model import
import seaborn as sns
```

12.2s

Then We read the csv file on our program. Used pandas and csv libraries.

```
# READ DATASET AND ANALYZE
diabetes_df = pd.read_csv('diabetes.csv')
result = diabetes_df
result.head()
# result = diabetes_df.columns
# result = diabetes_df.info()
# result = diabetes_df.describe()
# result = diabetes_df.describe().T
# result = diabetes_df.isnull().sum()
```

0.6s

In that image, we can see first 5 rows quickly by Pandas.

| | 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 |

## Handling Missing Datas On The Dataset

There was a few values empty like 0 or NaN on our dataset.

```
#making a copy of our dataframe beacuse we don't want to break our main dataset
# #and replace NaN values to 0 in order to calculate number of NaN values
diabetes_df_copy = diabetes_df.copy()
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]=
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)
print(diabetes_df_copy.isnull().sum())
diabetes_df_copy.head()
```

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 6 | 148.0 | 72.0 | 35.0 | NaN | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85.0 | 66.0 | 29.0 | NaN | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183.0 | 64.0 | NaN | NaN | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137.0 | 40.0 | 35.0 | 168.0 | 43.1 | 2.288 | 33 | 1 |

I changed the NaN values to 0 in order to delete them because when there is some missing values on the data set, our results from algorithms and methods could be wrong.

First, i replace the NaN values to 0. Before do that process, i took a copy of that csv file because we don’t want to break the original dataset right now.

I will work on the copy version. After taking the copy version, I changed the NaN values to 0.

```
#we changed Nan values to 0 and now we will turn that values to their own columns's median values on our copy dataframe
# because we will make predict we don't want to zero values on rows
diabetes_df_copy['Glucose'].fillna(diabetes_df_copy['Glucose'].mean(), inplace = True)
diabetes_df_copy['BloodPressure'].fillna(diabetes_df_copy['BloodPressure'].mean(), inplace = True)
diabetes_df_copy['SkinThickness'].fillna(diabetes_df_copy['SkinThickness'].mean(), inplace = True)
diabetes_df_copy['Insulin'].fillna(diabetes_df_copy['Insulin'].mean(), inplace = True)
diabetes_df_copy['BMI'].fillna(diabetes_df_copy['BMI'].median(), inplace = True)
print(diabetes_df_copy.isnull().sum())
diabetes_df_copy.head()
```

[5] ✓ 0.6s Python

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

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 6 | 148.0 | 72.0 | 35.00000 | 155.548223 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85.0 | 66.0 | 29.00000 | 155.548223 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183.0 | 64.0 | 29.15342 | 155.548223 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89.0 | 66.0 | 23.00000 | 94.000000 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137.0 | 40.0 | 35.00000 | 168.000000 | 43.1 | 2.288 | 33 | 1 |

Then i changed float numbers to integer on our copy dataset

```
diabetes_df_copy[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction']] =
diabetes_df_copy[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction']].apply(np.int64)
diabetes_df_copy
# i changed the float numbers to integer
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 6 | 148 | 72 | 35 | 155 | 33 | 0 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 155 | 26 | 0 | 31 | 0 |
| 2 | 8 | 183 | 64 | 29 | 155 | 23 | 0 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28 | 0 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43 | 2 | 33 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 763 | 10 | 101 | 76 | 48 | 180 | 32 | 0 | 63 | 0 |
| 764 | 2 | 122 | 70 | 27 | 155 | 36 | 0 | 27 | 0 |
| 765 | 5 | 121 | 72 | 23 | 112 | 26 | 0 | 30 | 0 |
| 766 | 1 | 126 | 60 | 29 | 155 | 30 | 0 | 47 | 1 |
| 767 | 1 | 93 | 70 | 31 | 155 | 30 | 0 | 23 | 0 |

768 rows × 9 columns

İ converted our copy of dataset to an array

```
# converted to an array
dataset = diabetes_df_copy.values
dataset
```

✓ 0.4s

```
array([[  6, 148,  72, ...,   0,  50,   1],
       [  1,  85,  66, ...,   0,  31,   0],
       [  8, 183,  64, ...,   0,  32,   1],
       ...,
       [  5, 121,  72, ...,   0,  30,   0],
       [  1, 126,  60, ...,   0,  47,   1],
       [  1,  93,  70, ...,   0,  23,   0]], dtype=int64)
```

Y is target dataset

```
#Get all of the rows from the first nine columns of the data set
X = dataset[:,0:8]
y = dataset[:,8] # target dataset
y
```

✓ 0.7s

I process our dataset and normalize between 0 and 1.

```
# Processing The Data
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
X_scale = min_max_scaler.fit_transform(X)
X_scale
```

✓ 0.6s

```
array([[0.35294118, 0.67096774, 0.48979592, ..., 0.30612245, 0.
        0.48333333],
       [0.05882353, 0.26451613, 0.42857143, ..., 0.16326531, 0.
        0.16666667],
       [0.47058824, 0.89677419, 0.40816327, ..., 0.10204082, 0.
        0.18333333],
       ...,
       [0.29411765, 0.49677419, 0.48979592, ..., 0.16326531, 0.
        0.15       ],
       [0.05882353, 0.52903226, 0.36734694, ..., 0.24489796, 0.
        0.43333333],
       [0.05882353, 0.31612903, 0.46938776, ..., 0.24489796, 0.
        0.03333333]])
```

Then i split the datas into training and testing

```
#
X_train , X_test , y_train, y_test = train_test_split(X_scale,y,test_size=0.2,random_state=4)
```

0.4s

Building ANN Model

```
# Build the model

#ANN MODEL
model = Sequential([
    Dense(8,activation='relu',input_shape=(8,)),
    Dense(4,activation='relu'),
    Dense(1,activation='sigmoid')
])
```

✓ 0.1s

```
model.compile(
    optimizer = 'sgd',
    loss = 'binary_crossentropy',
    metrics=['accuracy']
)
```

[18] ✓ 0.6s

Putting the model into training

```
hist = model.fit(X_train,y_train,batch_size=57,epochs=1000,validation_split=0.2)
```

✓ 41.4s

# 4. RESULTS

## VISUALIZING AND PREDICTION

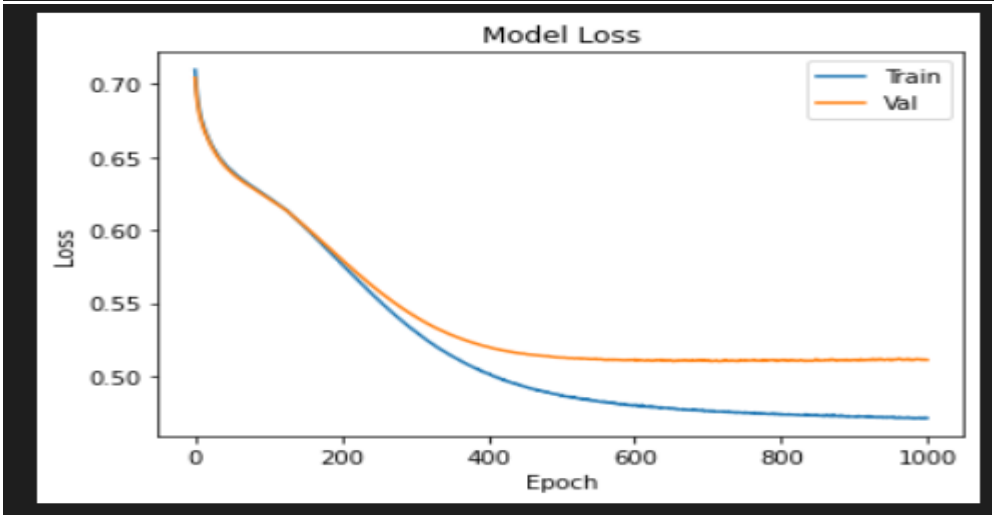
### Train the model

```
9/9 [==============================] - 0s 4ms/step - loss: 0.6758 - accuracy: 0.6721 - val_loss: 0.6739 - val_accuracy: 0.6667
Epoch 11/1000
9/9 [==============================] - 0s 4ms/step - loss: 0.6740 - accuracy: 0.6619 - val_loss: 0.6723 - val_accuracy: 0.6667
Epoch 12/1000
9/9 [==============================] - 0s 4ms/step - loss: 0.6724 - accuracy: 0.6538 - val_loss: 0.6707 - val_accuracy: 0.6585
Epoch 13/1000
...
Epoch 999/1000
9/9 [==============================] - 0s 5ms/step - loss: 0.4714 - accuracy: 0.7760 - val_loss: 0.5115 - val_accuracy: 0.7398
Epoch 1000/1000
9/9 [==============================] - 0s 7ms/step - loss: 0.4715 - accuracy: 0.7739 - val_loss: 0.5113 - val_accuracy: 0.7398
```

### Model Loss

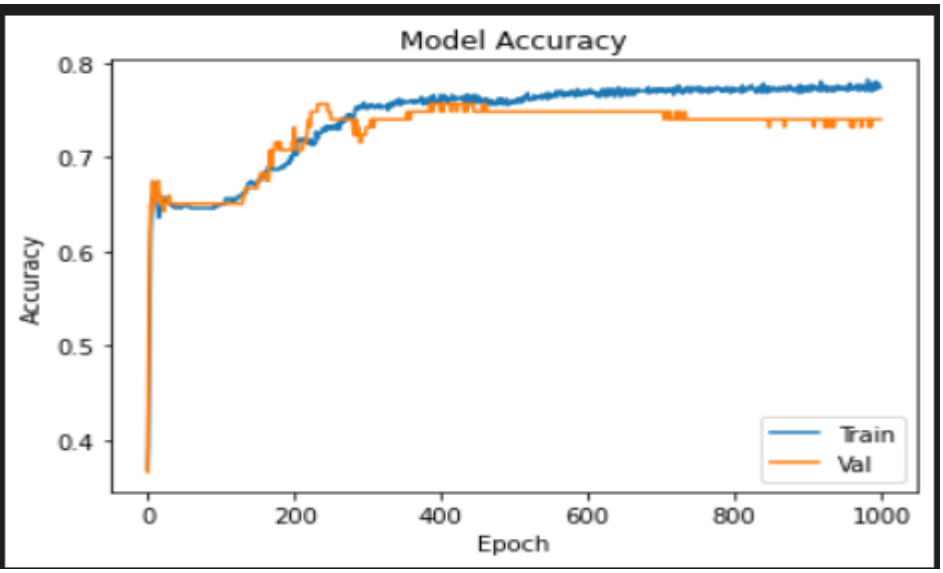
```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Val'], loc = 'upper right')
plt.show()
```

✓ 0.1s



# Model Accuracy

```
# Visualize the training accuracy and the validation accuracy to see if the model is overfitting
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Val'], loc = 'lower right')
plt.show()
```




# PREDICTION

```
prediction = model.predict(X_test)
prediction  # prediction for diabetes
```

```
Output exceeds the size limit. Open the full output data in a text editor
array([[0.17623295],
       [0.08536828],
       [0.0815507 ],
       [0.4228792 ],
       [0.27584407],
       [0.7407669 ],
       [0.6545072 ],
       [0.22940831],
       [0.8730784 ],
       [0.1768477 ],
       [0.41567364],
       [0.08154981],
       [0.15328121],
       [0.08154981],
```

MAKING PREDICTION WITH SHOWING ORIGINAL VALUES

```
# Making a prediction and also showing original values
prediction = model.predict(X_test)
prediction = [1 if y > 0.5 else 0 for y in prediction]
print(prediction)
print(y_test)
```

✓ 0.9s

```
[0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
[0 0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 0 0 0
 0 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0
 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1 1 0 0
 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1 1 0
 0 0 0 0 1 1]
```

EVALUATE THE MODEL

```
#Evaluate the model the test data set
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

pred = model.predict(X_test)
pred = [1 if y > 0.5 else 0 for y in pred]
print(classification_report(y_test,pred))
print('Confusion Matrix : \n',confusion_matrix(y_test,pred))
print('Accuracy : ', accuracy_score(y_test,pred))
```

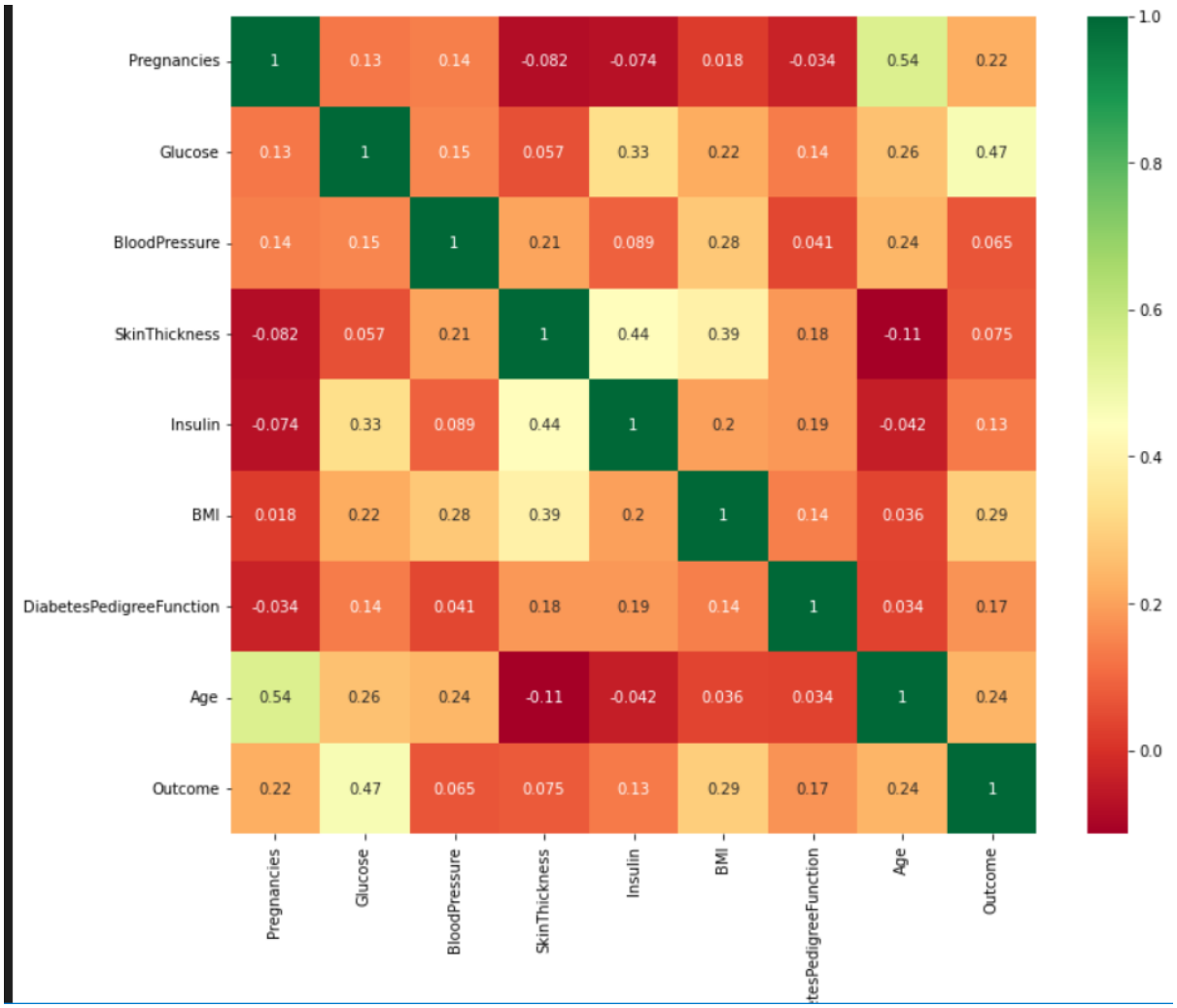
✓ 0.9s

```
5/5 [==============================] - 0s 924us/step
              precision    recall  f1-score   support

           0       0.84      0.88      0.86       102
           1       0.74      0.67      0.71        52

    accuracy                           0.81       154
   macro avg       0.79      0.78      0.78       154
weighted avg       0.81      0.81      0.81       154

Confusion Matrix :
 [[90 12]
 [17 35]]
Accuracy :  0.8116883116883117
```

# HEATMAP

```
# used seaborn for heatmap
plt.figure(figsize=(12,10))
p = sns.heatmap(diabetes_df.corr(), annot=True,cmap ='RdYlGn')
```

✓ 0.5s



# 5.CONCLUSIONS

An artificial Neural Network for prediction diabet person After using all the patient records from our Pima Indians Diabetes Dataset we made a machine learning model to predict patients in the dataset have diabetes or not. We get 0.81 Accurate this shows us the algorithm which is we did above, works fine.

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

https://www.analyticsvidhya.com/blog/2022/01/diabetes-prediction-using-machine-learning/