diabetes-predictive-system-1

November 25, 2024

[3]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
 ↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
 ↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
 ↪all files under the input directory
```

1 Import required libraries

[4]:
```
#To avoid unnecessary warnings messages
import warnings
warnings.filterwarnings('ignore')

#Visualisation libraries
import seaborn as sns
import matplotlib.pyplot as plt

#To scale our data
from sklearn.preprocessing import StandardScaler

#To split the data into training and testing set
from sklearn.model_selection import train_test_split

#Classification model- Support Vector Machine
from sklearn.svm import SVC

#Classification model performance metrics
from sklearn.metrics import␣
 ↪accuracy_score,confusion_matrix,classification_report,roc_auc_score
```

[6]:
```
df=pd.read_csv(r'C:\Users\VISHESH S\Downloads\diabetes.csv')
df.head()
```

[6]:

	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

2 Number of Rows and columns

[7]:
```
df.shape
```

[7]:
```
(768, 9)
```

3 Dataset Information

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

4 Data Description

[9]:
```
df.describe()
```

[9]:

	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	

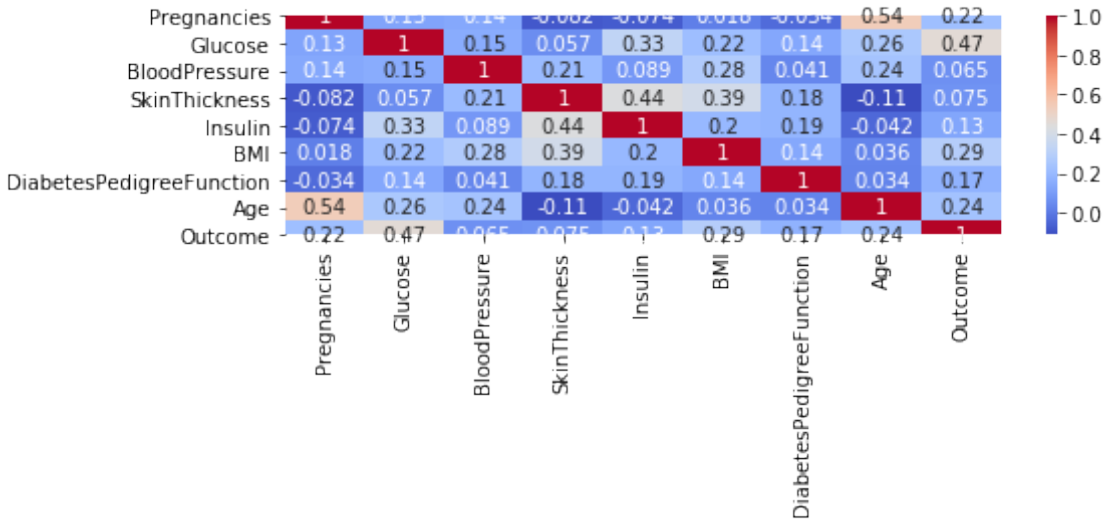
	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%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

[10]:
```
sns.pairplot(df,hue='Outcome')
```

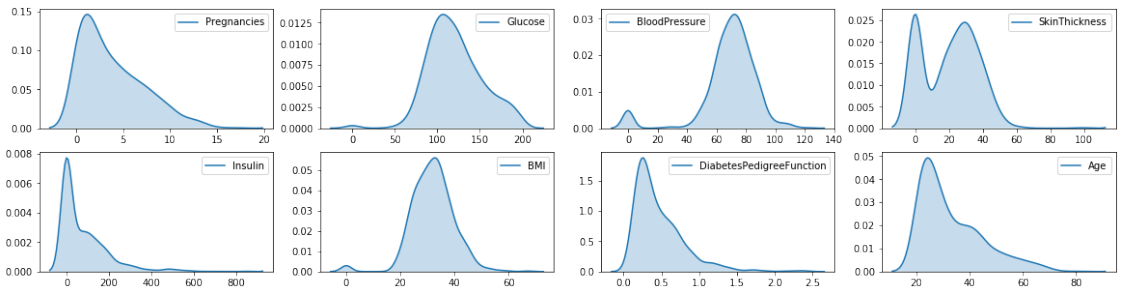
[10]:
```
<seaborn.axisgrid.PairGrid at 0x255b5e0f6c8>
```

5 Data Correlation

[11]:
```
plt.figure(figsize=(8,2))
sns.heatmap(df.corr(),annot=True,cmap='coolwarm')
plt.show()
```



[12]:
```
plt.figure(figsize=(20,5))
for i,col in enumerate(df.iloc[:,:-1]):
    plt.subplot(2,4,i+1)
    sns.kdeplot(df[col],shade=True)
plt.show()
```



6 Target Grouped data mean

[13]:
```
df.groupby('Outcome').mean()
```

[13]:
```
         Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
Outcome
0           3.298000  109.980000      68.184000      19.664000   68.792000
1           4.865672  141.257463      70.824627      22.164179  100.335821

                BMI  DiabetesPedigreeFunction        Age
Outcome
```

```
0         30.304200                  0.429734  31.190000
1         35.142537                  0.550500  37.067164
```

7 Target Value counts- Imbalanced data

[14]:
```
df['Outcome'].value_counts()
```

[14]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

8 Splitting dataset into features and target

[15]:
```
x=df.drop('Outcome',axis=1)
y=df['Outcome']
```

9 Scaling the data

[16]:
```
ss=StandardScaler()
ss.fit(x)
```

[16]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

[17]:
```
x=ss.transform(x)
```

[18]:
```
x
```

[18]:
```
array([[ 0.63994726,  0.84832379,  0.14964075, …,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, …, -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, …, -1.10325546,
         0.60439732, -0.10558415],
       …,
       [ 0.3429808 ,  0.00330087,  0.14964075, …, -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, …, -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, …, -0.20212881,
        -0.47378505, -0.87137393]])
```

10 Splitting dataset further into training and testing data

[19]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=200,test_size=0.
↪2,stratify=y)
```

11 Model Evaluation

[20]:
```
model=SVC(kernel='linear',C=10.0)

model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
ac=accuracy_score(ytest,ypred)
cm=confusion_matrix(ytest,ypred)
cr=classification_report(ytest,ypred)
train=model.score(xtrain,ytrain)
test=model.score(xtest,ytest)

print(f'{model} Accuracy:{ac}\n{cm}\n{cr}\nTraining Accuracy: {train}\nTesting␣
↪Accuracy: {test}')
```

```
SVC(C=10.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False) Accuracy:0.8051948051948052
[[89 11]
 [19 35]]
              precision    recall  f1-score   support

           0       0.82      0.89      0.86       100
           1       0.76      0.65      0.70        54

    accuracy                           0.81       154
   macro avg       0.79      0.77      0.78       154
weighted avg       0.80      0.81      0.80       154

Training Accuracy: 0.7687296416938111
Testing Accuracy: 0.8051948051948052
```

12 Model Deployment- Predictive System

[21]:
```
def prediction(input_data=()):
    #Converting to numpy array
    array_input_data=np.asarray(input_data)

    #Reshaping to tell the model we want prediction for 1 instance
    x=array_input_data.reshape(1,-1)
```

```
    #Standardising data
    standard_x=ss.transform(x)

    #Predicting
    p=model.predict(standard_x)

    #Returning the prediction
    if p==0:
        print('Patient is not Diabetic')
    else:
        print('Patient is Diabetic')

    return p
```

[22]:
```
prediction([4,110,92,0,0,37.6,0.191,30])
```

```
Patient is not Diabetic
```

[22]:
```
array([0], dtype=int64)
```

[23]:
```
prediction([6,110,92,0,0,86.6,0.191,60])
```

```
Patient is Diabetic
```

[23]:
```
array([1], dtype=int64)
```

[32]:
```
def predictions(input_data=()):
    #Converting to numpy array
    array_input_data=np.asarray(input_data)



    #Reshaping to tell the model we want prediction for 1 instance
    x=array_input_data.reshape(1,-1)



    #Predicting
    p=model.predict(x)

    #Returning the prediction
    if p==0:
        print('Patient is not Diabetic')
    else:
        print('Patient is Diabetic')
```

```
    return p
```

[33]:
```
predictions([6,110,92,0,0,86.6,0.191,60])
```

```
Patient is Diabetic
```

[33]:
```
array([1], dtype=int64)
```

[ ]:
```

```