Harsh Seksaria

2048011

1-MDS

Diabetes Analysis

The following dataset contains data of diabetic patients recording different measures of biological factors which help in predicting whether or not the patient is prone to be affected from diabetes.

The dataset has been downloaded from Kaggle.com

Reading File

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
fl = pd.read_csv('../diabetes.csv')
fl.sample(3)
```

...

In [3]:
```
fl.describe()
```

...

The above output generated descriptive statistics summarizing data disribution on numerical values.

In [4]:
```
fl.info(verbose=True)
```

...

The above output gives us the information about the data types, non-null count, total columns, total records and memory usage.

Lab 1

Data Distribution

In [6]:
```
p = fl.hist(figsize=(20,20))
```

...

Missing Values Imputation

Wherever we have values as 0, it doesn't make any sense will cause problems in accurately understanding data. They are just like null. So, we first convert them to *null* and then do the missing value imputation.

In [4]:
```
fl[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']] = fl[['Glucos
fl.head(5)
```

...

In [8]:
```
fl.isnull().sum()
```

...

We can clearly see that there were many 0 in our dataset, and now they have been converted to 0.

Now, we will fill these null values with appropriate methods to represent the best of data.

In [5]:
```
fl['Glucose'].fillna(fl['Glucose'].mean(), inplace=True)
fl['BloodPressure'].fillna(fl['BloodPressure'].mean(), inplace=True)
fl['SkinThickness'].fillna(fl['SkinThickness'].median(), inplace=True)
fl['Insulin'].fillna(fl['Insulin'].median(), inplace=True)
fl['BMI'].fillna(fl['BMI'].median(), inplace=True)
```

In [10]:
```
1 fl.isnull().sum()
```

...

We first imputed null values by using mean/median whichever was suitable and then checked for null values to verify that there existed none.

Let's see the histogram again.

In [12]:
```
1 p = fl.hist(figsize=(20,20))
```

...

In [13]:
```
1 fl['Outcome'].value_counts().plot(kind='bar', y=['0', '1'])
2 var = fl['Outcome'].value_counts().to_dict()
3 print('Diabetic patients: ', str(var[1.0]))
4 print('Non-Diabetic patients: ', str(var[0.0]))
```

...

We can see that there are 500 patients who don't have diabetes but 268 do suffer from it.

Scatter Plot

In [14]:
```
1 p=sns.pairplot(fl, hue = 'Outcome')
```

...

The above graphs show the relationship between two quantities - the measure of the strength of association between two variables.

Heat-map

A heat-map represents the information with the help of different colours, rather, different shades of a color.

In [15]:
```
1 plt.figure(figsize=(12,10))
2 p=sns.heatmap(fl.corr(), annot=True, cmap ='RdYlGn')
```

...

Violin Plot

A violin plot is a method of plotting numeric data. It is similar to box plot with a rotated kernel density plot on each side. Violin plots are similar to box plots, except that they also show the probability density of the data at different values (in the simplest case this could be a histogram).

In [16]:
```
df=fl
```

In [17]:
```
fig,ax = plt.subplots(nrows=4, ncols=2, figsize=(18,18))
plt.suptitle('Violin Plots',fontsize=24)
sns.violinplot(x="Pregnancies", data=df,ax=ax[0,0],palette='Set3')
sns.violinplot(x="Glucose", data=df,ax=ax[0,1],palette='Set3')
sns.violinplot (x ='BloodPressure', data=df, ax=ax[1,0], palette='Set3')
sns.violinplot(x='SkinThickness', data=df, ax=ax[1,1],palette='Set3')
sns.violinplot(x='Insulin', data=df, ax=ax[2,0], palette='Set3')
sns.violinplot(x='BMI', data=df, ax=ax[2,1],palette='Set3')
sns.violinplot(x='DiabetesPedigreeFunction', data=df, ax=ax[3,0],palette='Set3')
sns.violinplot(x='Age', data=df, ax=ax[3,1],palette='Set3')
plt.show()
```

...

In [18]:
```
plt.subplots(figsize=(20,15))
sns.boxplot(x='Outcome', y='Age', data=fl)
```

...

Conclusion

We saw the data with the help of different types of graphs which told us how the different variables behave under the influence of other variables.

Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples.

The most widely used supervised learning algorithms are:

1. Support Vector Machines

2. Linear Regression
3. Logistic Regression
4. k-nearest neighbour
5. Decision trees
6. Naive Bayes
7. Linear Discriminant Analysis

The best suited algorithm for the model is the one that yeilds highest accuracy. Upon implementing and checking the algorithms, we can know which is most accurate and hence choose that one.

Now, for the dataset selected, supervised learning will be a suitable choice as it contains labels.

Lab 2

In [7]:
```
1 fl.sample(3)
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
175	8	179	72	42	130	32.7	0.719	36	1
708	9	164	78	0	0	32.8	0.148	45	1
422	0	102	64	46	78	40.6	0.496	21	0

Implementation of the Supervised Machine Learing Algorithm following the Non-parametric Approach.

In [8]:
```
1 #Import
2 from sklearn.model_selection import train_test_split
3 from sklearn.neighbors import KNeighborsClassifier
4 from sklearn import tree
5 from sklearn.metrics import accuracy_score
```

In [22]:
```
1 fl = pd.read_csv('../diabetes.csv')
```

In [6]:
```
#Dividing dataset into predictors and target
"""Here, we split our dataset into predictors and target variables.
In x, we store predictors. In y, we store target variable.
So, we assign the outcome column to y. And drop it & assign rest dataset to x."""
y = fl.Outcome
x = fl.drop("Outcome", 1)
```

In [9]:
```
#Splitting dataset into training and testing sets
"""Here, we split the dataset into the training set and testing set.
We take 70% of our data for training and the rest 30% for testing"""
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.3, random_state=11, stratify=y)
```

KNN

In [ ]:
```

```

In [10]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, Y_train)
```

Out[10]:
```
KNeighborsClassifier()
```

In [11]:
```
predict_knn = knn.predict(X_test)
predict_knn
```

Out[11]:
```
array([0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0], dtype=int64)
```

In [12]:
```
#Accuracy Score
accuracy_score(Y_test, predict_knn)*100
```

Out[12]: 70.12987012987013

In [13]:
```
#Actual value vs. Predicted Value
compare = pd.DataFrame({"Actual Valule":Y_test, "Predicted Value":predict_knn})
compare
```

Out[13]:

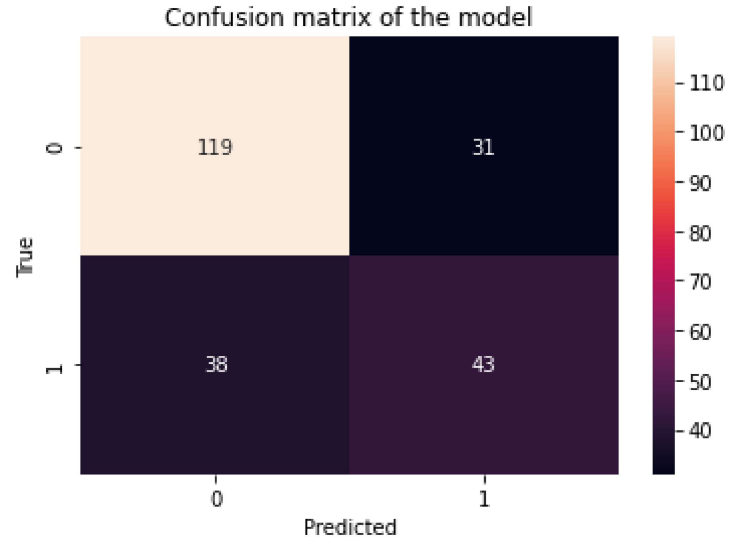
	Actual Valule	Predicted Value
102	0	0
18	0	0
672	0	0
182	0	0
75	0	0
...	...	...
493	1	1
728	0	1
301	1	0
652	0	0
526	0	0

231 rows × 2 columns

In [14]:
```
#Exporting the comparison
compare.to_csv("compare.csv")
```

In [15]:
```
#Confusion matrix and Classification Report
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
matrix = confusion_matrix(Y_test, predict_knn)
matrix

sns.heatmap(matrix, annot=True, fmt='d')
plt.title('Confusion matrix of the model')
plt.xlabel('Predicted')
plt.ylabel('True')

print(classification_report(Y_test, predict_knn))
```

```
              precision    recall  f1-score   support

           0       0.76      0.79      0.78       150
           1       0.58      0.53      0.55        81

    accuracy                           0.70       231
   macro avg       0.67      0.66      0.67       231
weighted avg       0.70      0.70      0.70       231
```



Conclusion

We used KNN algorithm to predict values. We used **70% of the dataset to train and 30% to test**. After prediction, on checking the accuracy score we got approx **70**. Which means the model was able to successfully predict correct outcome of the 70% of the test dataset.

To test the robustness of the model, we can use validation techniques. Through this technique we keep some records from the dataset for testing and use the rest for training. Some methods of validation are:

1. k-folds Cross Validation
2. Leave-One-Out Cross Validation
3. Stratified 1. k-folds Cross Validation