Machine Learning Lab 6 - Diabetes Classification

Submitted By
Name: **Rathod Nishit Shailesh**
Register Number: **19112014**
Class: **5 BSc Data Science**

Lab Overview

Objective

- Get familiar with the problem statement, Know the dataset thoroghly, Analyse the given dataset by exploring the hidden insights with beautiful visuals and Train & Test the model for acurate classification prediction of Diabetes.

Problem Definition

- Understand the Dataset & Features.
- Perform Data Preprocessing Technique to Get Balanced Structured Data.
- Perform Statistical Data Analysis and Derive Valuable Inferences.
- Perform Exploratory Data Analysis and Derive Valuable Insights.
- Train and Test through Different Classification Models for Better Pricdiction.

Approach

This is an extension to the Problem Defnintion. Mention the process/appraoch that you have followed in order to reach out the above problem defintion.

- Step 1: Know the dataset thoroughly.
- Step 2: Perform preprocessing on data.
- Step 3: Import needfull libraries as an when you try to plot different graphs and evaluate the model.
- Step 4: Perform Statistical Data Analysis and Derive Valuable Inferences.
- Step 5: Perform Exploratory Data Analysis and Derive Valuable Insights.
- Step 6: Train and Test through Different Classification Models for Better Diabetes Prediction.
- Step 7: Help the doctors with insights for predicting if a patient is diaganose with diabetes or not.

Sections

Here, mentioned sections are defined in the below code. For this lab, the sections are -

1. Lab Overview
2. Dataset Overview
3. Data Analyst Process
4. About Different Classification Models
5. Implementation and Evaluation of Different Classification Models
6. Conclusion

References

1. https://pandas.pydata.org/
2. https://matplotlib.org/
3. https://seaborn.pydata.org/
4. https://plotly.com/
5. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
6. https://www.kaggle.com/saurabh00007/diabetescsv

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

%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Diabetes.csv to Diabetes.csv
```

In [3]:
```
df = pd.read_csv("Diabetes.csv")
```

In [4]:
```
df.shape
```

Out[4]:
```
(768, 9)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [6]:
```
df.head()
```

Out[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

In [7]:
```
df.tail()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

In [8]:
```
df.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
```

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

Out[9]:

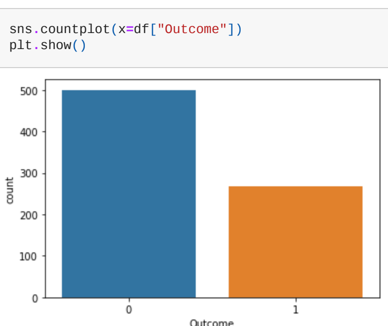
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	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

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Exploratory Data Analysis

In [11]:
```
sns.countplot(x=df["Outcome"])
plt.show()
```

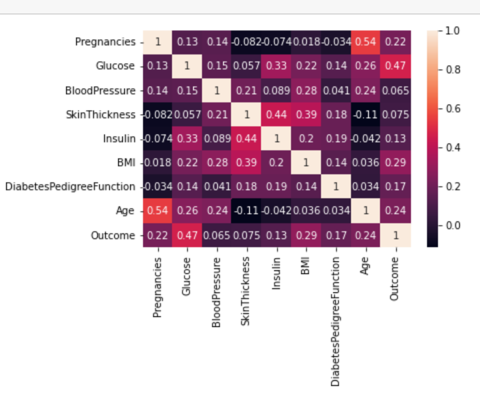


In [12]:
```
df.corr()
```

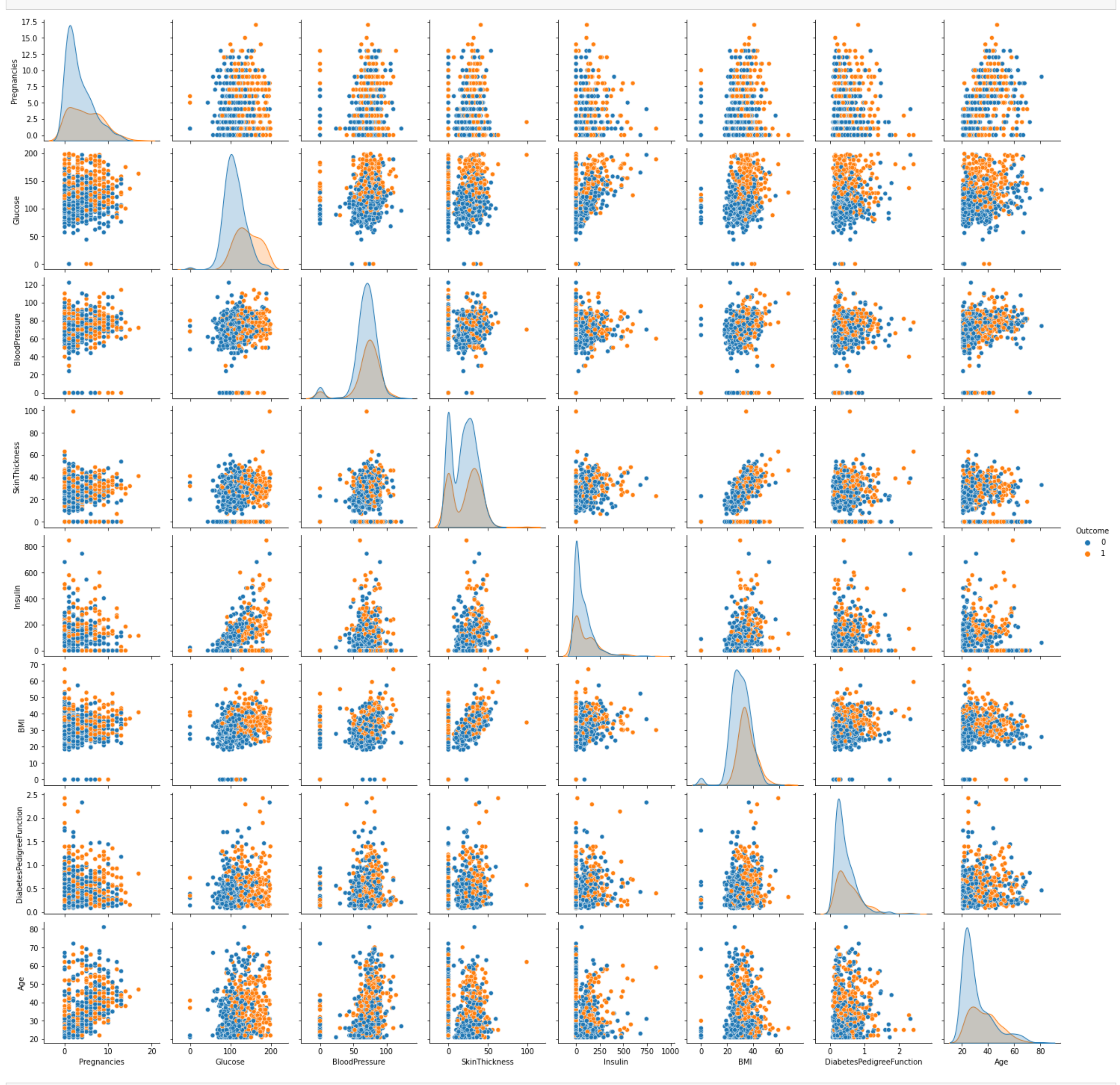
Out[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

In [13]:
```
sns.heatmap(df.corr(), annot=True)
plt.show()
```



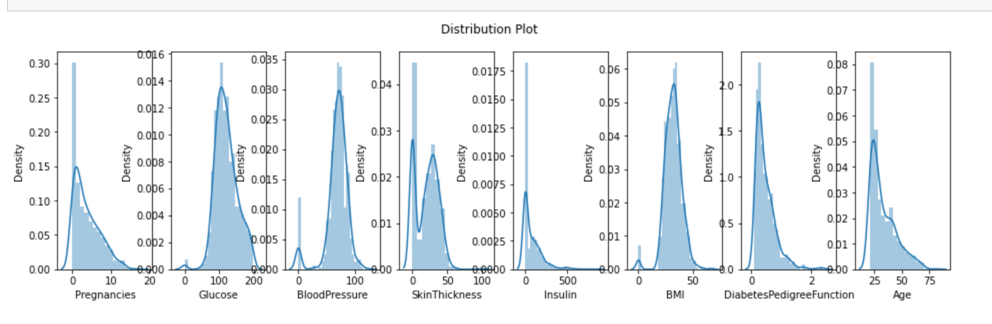
In [14]:
```
sns.pairplot(df, hue = "Outcome")
plt.show()
```



In [15]:
```
columns = list(df.columns)
columns.remove('Outcome')
fig, ax = plt.subplots(ncols = 8, figsize=(16, 4))
fig.suptitle("Distribution Plot")

for index, column in enumerate(columns):
    sns.distplot(df[column], ax=ax[index])

plt.show()
```



Implementing Classification Models

Biffurcating Classification Parameter from the Dataset.

In [16]:
```
Y = df['Outcome']
X = df.drop(['Outcome'], axis=1)
```

In [17]:
```
X.sample(5)
```

Out[17]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
721	1	114	66	36	200	38.1	0.289	21
318	3	115	66	39	140	38.1	0.150	28
634	10	92	62	0	0	25.9	0.167	31
665	1	112	80	45	132	34.8	0.217	24
762	9	89	62	0	0	22.5	0.142	33

In [18]:
```
Y.sample(5)
```

Out[18]:
```
591    0
352    0
63     0
338    1
210    0
Name: Outcome, dtype: int64
```

Train Test Split

In [19]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.30, random_state = 8)
```

In [20]:
```
X_train.sample(5)
```

Out[20]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
607	1	92	62	25	41	19.5	0.482	25
478	8	126	74	38	75	25.9	0.162	39
49	7	105	0	0	0	0.0	0.305	24
594	6	123	72	45	230	33.6	0.733	34
622	6	183	94	0	0	40.8	1.461	45

In [21]:
```
X_test.sample(5)
```

Out[21]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
370	3	173	82	48	465	38.4	2.137	25
246	10	122	68	0	0	31.2	0.258	41
549	4	189	110	31	0	28.5	0.680	37
181	0	119	64	18	92	34.9	0.725	23
757	0	123	72	0	0	36.3	0.258	52

In [22]:
```
Y_train.sample(5)
```

Out[22]:
```
195    1
394    1
212    0
333    0
379    0
Name: Outcome, dtype: int64
```

In [23]:
```
Y_test.sample(5)
```

Out[23]:
```
522    0
350    0
25     1
488    0
740    1
Name: Outcome, dtype: int64
```

In [24]:
```
from sklearn.metrics import accuracy_score
```

Support Vector Machine

Fitting and Predicting

In [25]:
```
from sklearn.svm import SVC
SupportVectorClassifier = SVC()
SupportVectorClassifier.fit(X_train,Y_train)
predA = SupportVectorClassifier.predict(X_test)
accuracy_score(predA,Y_test)
```

Out[25]:
```
0.7402597402597403
```

Classification Report

In [26]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test, predA))
```

```
              precision    recall  f1-score   support

           0       0.73      0.93      0.82       147
           1       0.77      0.40      0.53        84

    accuracy                           0.74       231
   macro avg       0.75      0.67      0.68       231
weighted avg       0.75      0.74      0.72       231
```

Confusion Matrix

In [27]:
```
from sklearn.metrics import confusion_matrix
```

In [28]:
```
CM = confusion_matrix(Y_test, predA)
CM
```

Out[28]:
```
array([[137,  10],
       [ 50,  34]])
```

In [29]:
```
Accuracy = (CM[0][0] + CM[1][1]) / (CM[0][0] + CM[1][1] + CM[0][1] + CM[1][0])
Accuracy
```

Out[29]:
```
0.7402597402597403
```

In [30]:
```
ErrorRate = (CM[0][1] + CM[1][0]) / (CM[0][0] + CM[1][1] + CM[0][1] + CM[1][0])
ErrorRate
```

Out[30]:
```
0.2597402597402597
```

In [31]:
```
Sensitivity = CM[0][0]/(CM[0][0] + CM[1][0])
Sensitivity
```

Out[31]:
```
0.732620320855615
```

In [32]:
```
Specificity = CM[1][1]/(CM[1][1] + CM[0][1])
Specificity
```

Out[32]:
```
0.7727272727272727
```

In [33]:
```
Precision = CM[0][0]/(CM[0][0] + CM[1][0])
Precision
```

Out[33]:
```
0.732620320855615
```

In [34]:
```
Recall = CM[0][0]/(CM[0][0] + CM[0][1])
Recall
```

Out[34]:
```
0.9319727891156463
```

In [35]:
```
F1Score = (2*(Precision*Recall))/(Precision + Recall)
F1Score
```

Out[35]:
```
0.8203592814371257
```

Evaluating the Effect of Parameters For Support Vector Classifier

In [36]:
```
def doSupportVectorClassifier(X, Y, test_size = 0.20, randomstate = None, c = 1.0, Kernel = "rbf"):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = test_size, random_state = randomstate)
    cls1 = SVC(C = c, kernel = Kernel)
    cls1.fit(X_train,Y_train)
    pred = cls1.predict(X_test)
    acc_score2 = accuracy_score(pred,Y_test)
    return acc_score2
```

In [37]:
```
test_size = [0.30, 0.25, 0.20, 0.10]
random_states = [8, 27, 42]

Regularization_Parameter = [1.0,50.0,100.0]
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
```

In [38]:
```
df1 = pd.DataFrame(columns = ['Test_Size','Random_States','Support_Vector_Machine_Accuracy','RegularizationParameter','Kernel'])
```

In [39]:
```
for t_size in test_size:
    for r_state in random_states:
        for RP in Regularization_Parameter:
            for ker in kernels:

                Algo1 = doSupportVectorClassifier(X, Y, t_size, r_state, RP, ker)
                SupportVectorMachine = {}
                SupportVectorMachine['Test_Size'] = t_size
                SupportVectorMachine['Random_States'] = r_state
                SupportVectorMachine['Support_Vector_Machine_Accuracy'] = Algo1
                SupportVectorMachine['RegularizationParameter'] = RP
                SupportVectorMachine['Kernel'] = ker

                df1 = df1.append(SupportVectorMachine, ignore_index = True)
```

In [40]:
```
df1.to_csv('Diabetes_SupportVectorMachine_Evalution.csv')
```

In [41]:
```
df1.head(10)
```

Out[41]:

	Test_Size	Random_States	Support_Vector_Machine_Accuracy	RegularizationParameter	Kernel
0	0.3	8	0.796537	1.0	linear
1	0.3	8	0.757576	1.0	poly
2	0.3	8	0.740260	1.0	rbf
3	0.3	8	0.519481	1.0	sigmoid
4	0.3	8	0.779221	50.0	linear
5	0.3	8	0.766234	50.0	poly
6	0.3	8	0.796537	50.0	rbf
7	0.3	8	0.437229	50.0	sigmoid
8	0.3	8	0.770563	100.0	linear
9	0.3	8	0.761905	100.0	poly

In [42]:
```
df1.tail(10)
```

Out[42]:

	Test_Size	Random_States	Support_Vector_Machine_Accuracy	RegularizationParameter	Kernel
134	0.1	42	0.766234	1.0	rbf
135	0.1	42	0.545455	1.0	sigmoid
136	0.1	42	0.688312	50.0	linear
137	0.1	42	0.727273	50.0	poly
138	0.1	42	0.753247	50.0	rbf
139	0.1	42	0.467532	50.0	sigmoid
140	0.1	42	0.688312	100.0	linear
141	0.1	42	0.714286	100.0	poly
142	0.1	42	0.714286	100.0	rbf
143	0.1	42	0.467532	100.0	sigmoid

Advantages of Support Vector Machine (SVM)

1. Regularization capabilities: SVM has L2 Regularization feature. So, it has good generalization capabilities which prevent it from over-fitting.
2. Handles non-linear data efficiently: SVM can efficiently handle non-linear data using Kernel trick.
3. Solves both Classification and Regression problems: SVM can be used to solve both classification and regression problems. SVM is used for classification problems while SVR (Support Vector Regression) is used for regression problems.
4. Stability: A small change to the data does not greatly affect the hyperplane and hence the SVM. So the SVM model is stable.

Disadvantages of Support Vector Machine (SVM)

1. Choosing an appropriate Kernel function is difficult: Choosing an appropriate Kernel function (to handle the non-linear data) is not an easy task. It could be tricky and complex. In case of using a high dimension Kernel, you might generate too many support vectors which reduce the training speed drastically.
2. Extensive memory requirement: Algorithmic complexity and memory requirements of SVM are very high. You need a lot of memory since you have to store all the support vectors in the memory and this number grows abruptly with the training dataset size.
3. Requires Feature Scaling: One must do feature scaling of variables before applying SVM.
4. Long training time: SVM takes a long training time on large datasets.
5. Difficult to interpret: SVM model is difficult to understand and interpret by human beings unlike Decision Trees.

Exploratory Data Analysis On The Results Derived From - Support Vector Machine

$! pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip

$ from pandas_profiling import ProfileReport

$ profile = ProfileReport(df2)

$ profile.to_notebook_iframe()

$ profile.to_file(output_file = "19112014_NishitRathod_EDA_Diabetes_ResultsDerivedFromAlgorithms.html")

Conclusion

In this lab, we have tried to gain the knowledge about data and its variables, further we did some preprocessing to the data in order to bring it into more analyst friendly mode, laterly we implemented various graphs using various libraries in order to get valuable insights, furthermore, we implemented and evaluated various classification models to get high accuracy in terms of predicting diabetes which can help the doctors to predict diabetic patients.