KNN Algorithm

- KNN is K-Nearrest Neighbor model which is a supervised classification algorithm used to predict discrete class labels/categories. Here the target variable is a categorical value. It could be a binary class or multi class.
- KNN works based on the majority of the K-nearest neighbors/K-nearest obersvations.
- KNN Algorithm, 1. Picks a value for K. 2. Calculates the distance from the unknown data point to k neighbors. 3. Select all the k-neighbors or the k-observation in the training data set that are nearest to the unknown data point. 4. Predict the response of the unknown data point usig the most popular response value from the k-nearest neighbors.

Note: 1. K value is given by the user but the best k-value can be determined by trail and error method of checking the accuracy score. 2. Distance is calulated using the Euclidean's distance formulae.

```python
# Importing necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Onboarding data onto colab

from google.colab import files
rawdata=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving diabetes (1).csv to diabetes (1).csv
```

```python
# DataFrame

df=pd.read_csv('diabetes (1).csv')
df
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  
\
0              6      148             72             35        0  33.6

1              1       85             66             29        0  26.6

2              8      183             64              0        0  23.3

3              1       89             66             23       94  28.1

4              0      137             40             35      168  43.1

..           ...      ...            ...            ...      ...   ...
```

```
763           10      101            76           48       180  32.9

764            2      122            70           27         0  36.8

765            5      121            72           23       112  26.2

766            1      126            60            0         0  30.1

767            1       93            70           31         0  30.4


     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]

# Shallow copy

df_copy=df.copy()
```

Exploratory Data Analysis

```
df.head()

   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1


   DiabetesPedigreeFunction  Age  Outcome
```

```
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

```
df.tail()
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
\
763           10      101             76             48      180  32.9

764            2      122             70             27        0  36.8

765            5      121             72             23      112  26.2

766            1      126             60              0        0  30.1

767            1       93             70             31        0  30.4


     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

```
df.shape
```

```
(768, 9)
```

```
# Technical report
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
```

```
# Statistical repoprt

df.describe()
```

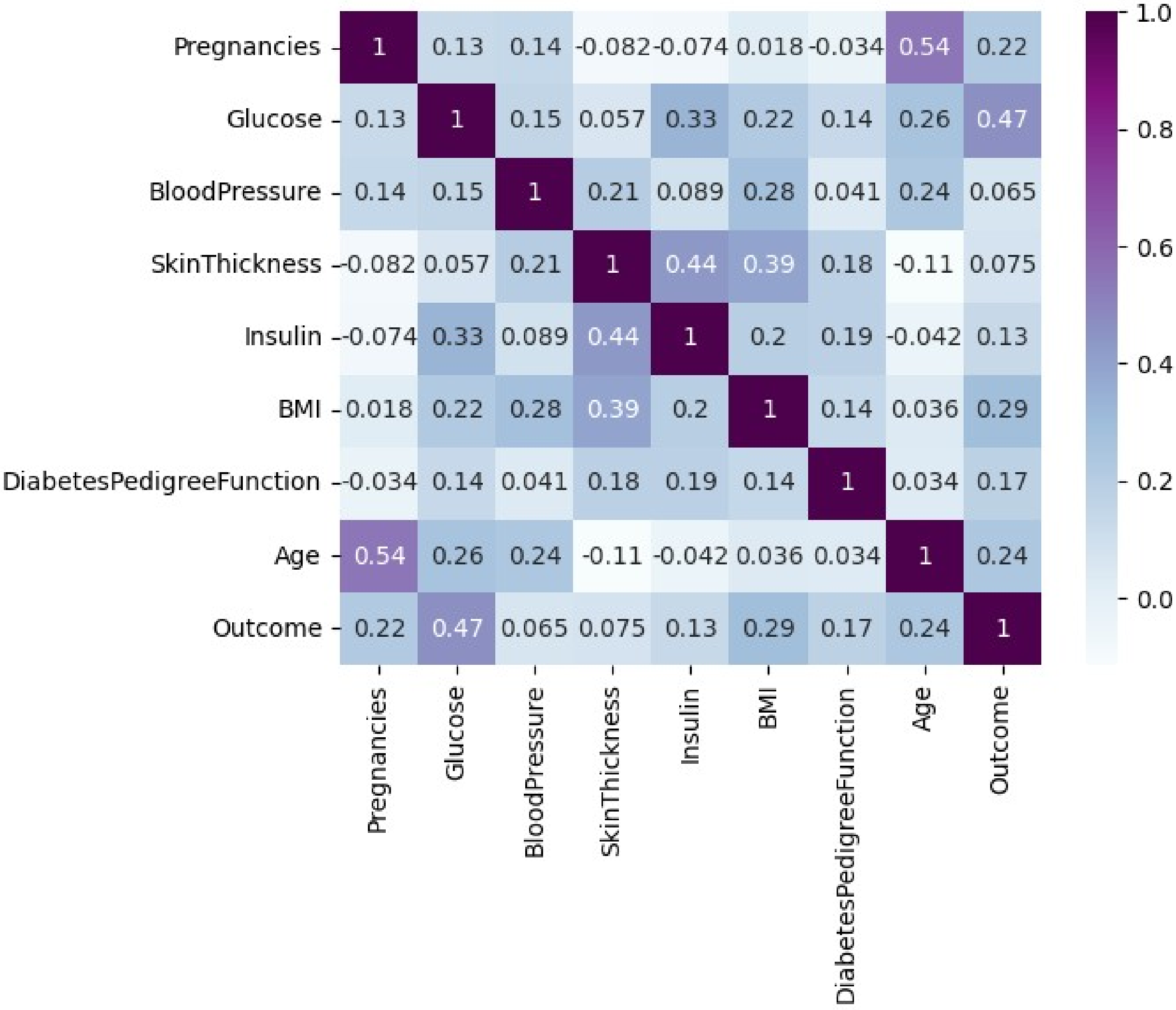
```
       Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
count   768.000000  768.000000     768.000000     768.000000
768.000000
mean      3.845052  120.894531      69.105469      20.536458
79.799479
std       3.369578   31.972618      19.355807      15.952218
115.244002
min       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
```

```
df.corr()
```

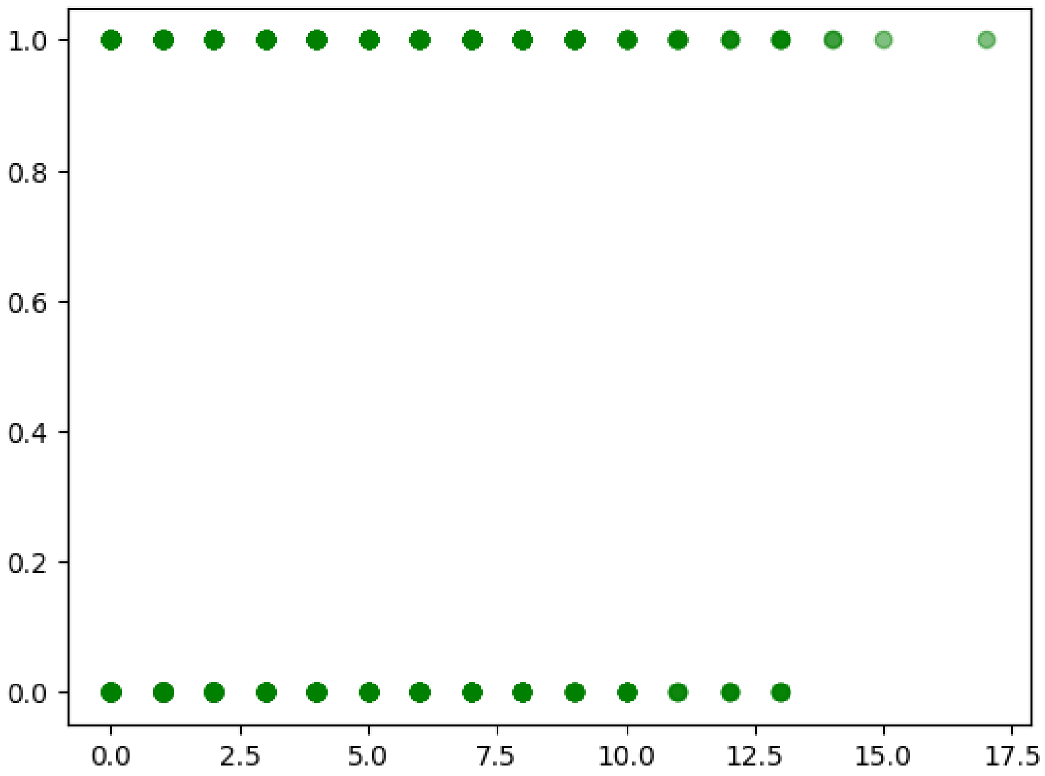
```
                          Pregnancies   Glucose  BloodPressure
SkinThickness  \
Pregnancies                  1.000000  0.129459       0.141282       -
0.081672
Glucose                      0.129459  1.000000       0.152590
0.057328
BloodPressure                0.141282  0.152590       1.000000
0.207371
SkinThickness               -0.081672  0.057328       0.207371
1.000000
Insulin                     -0.073535  0.331357       0.088933
0.436783
BMI                          0.017683  0.221071       0.281805
0.392573
DiabetesPedigreeFunction    -0.033523  0.137337       0.041265
```

```
0.183928
Age                          0.544341  0.263514       0.239528      -
0.113970
Outcome                      0.221898  0.466581       0.065068
0.074752

                           Insulin       BMI  DiabetesPedigreeFunction
\
Pregnancies              -0.073535  0.017683                 -0.033523

Glucose                   0.331357  0.221071                  0.137337

BloodPressure             0.088933  0.281805                  0.041265

SkinThickness             0.436783  0.392573                  0.183928

Insulin                   1.000000  0.197859                  0.185071

BMI                       0.197859  1.000000                  0.140647

DiabetesPedigreeFunction  0.185071  0.140647                  1.000000

Age                      -0.042163  0.036242                  0.033561

Outcome                   0.130548  0.292695                  0.173844


                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.263514  0.466581
BloodPressure             0.239528  0.065068
SkinThickness            -0.113970  0.074752
Insulin                  -0.042163  0.130548
BMI                       0.036242  0.292695
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000

# Check for null value

df.isna().sum()/len(df)*100

Pregnancies                 0.0
Glucose                     0.0
BloodPressure               0.0
SkinThickness               0.0
Insulin                     0.0
BMI                         0.0
DiabetesPedigreeFunction    0.0
Age                         0.0
```

```
Outcome                     0.0
dtype: float64

# check for duplicate value

df.duplicated().sum()

0

sns.heatmap(df.corr(),annot=True,cmap='BuPu')
plt.show()
```

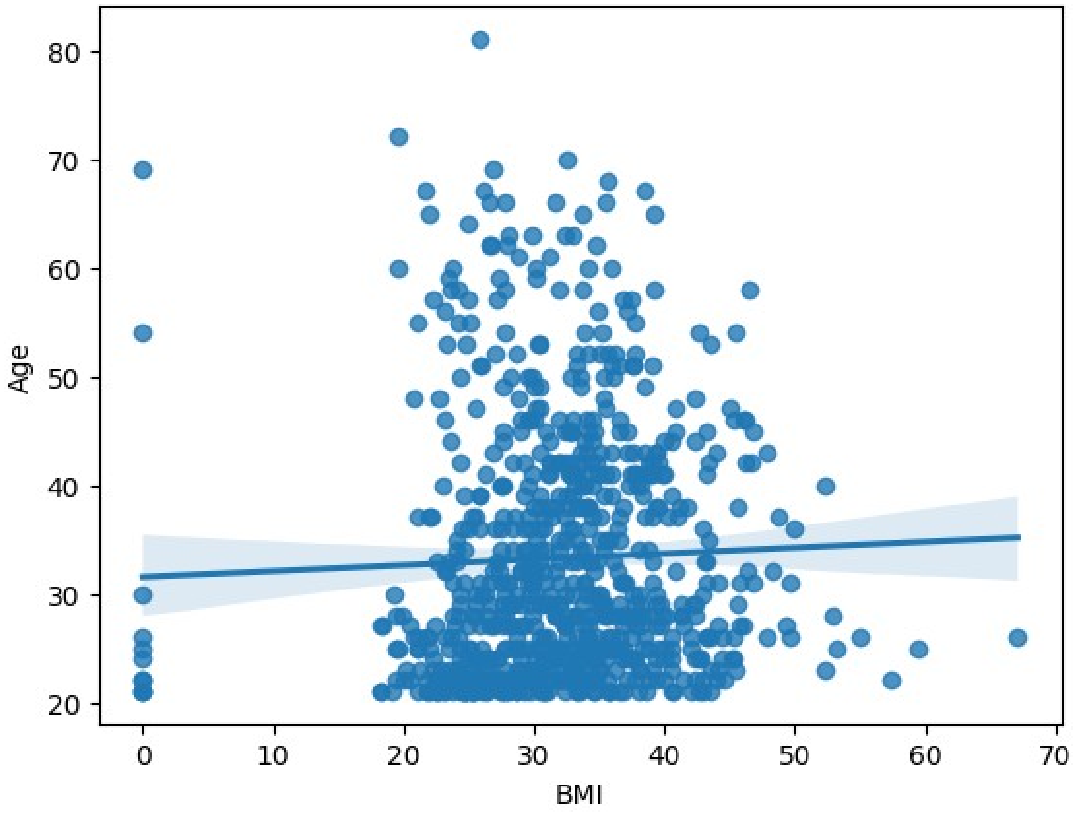


```
plt.scatter(df['Pregnancies'],df['Outcome'],alpha=0.5,color='green')
plt.show()
```



Multi-variate Analysis

```
sns.regplot(x='BMI',y='Age',data=df)
plt.show()
```



Data cleaning

Since there is no null values or duplicate values, skipping this data cleaning step.

Data encoding

Since all the features have numerical value in it, skipping this data encoding step.

Standardization

```
X=df.iloc[:,:-1]
Y=df.iloc[:,[-1]]

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()
X_sc=sc.fit_transform(X)
X_sc
```

```
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]])
```

Train Test Split

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(X_sc,Y,test_size=0.2,ra
ndom_state=7)
```

Model Building

```
from sklearn.neighbors import KNeighborsClassifier

knc=KNeighborsClassifier(n_neighbors=7,p=2)
knc.fit(x_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/
_classification.py:215: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return self._fit(X, y)
```

```
KNeighborsClassifier(n_neighbors=7)
```

```
y_predict=knc.predict(x_test)
```

```
y_predict
```

```
array([0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
0,
       1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
1,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
0])
```

```
y_test
```

```
     Outcome
353        0
236        1
323        1
98         0
701        1
..       ...
153        0
392        0
308        1
70         1
513        0

[154 rows x 1 columns]
```

Model Evaluation

```
from sklearn.metrics import
confusion_matrix,accuracy_score,recall_score,f1_score

cm=confusion_matrix(y_test,y_predict)
acs=accuracy_score(y_test,y_predict)
rs=recall_score(y_test,y_predict)
f1=f1_score(y_test,y_predict)
print('Confusion_matrix',cm)
print('Accuracy_score',acs)
print("Recall_score",rs)
print('F1_score',f1)
```

```
Confusion_matrix [[82 15]
 [29 28]]
Accuracy_score 0.7142857142857143
Recall_score 0.49122807017543857
F1_score 0.5599999999999999
```

F1_score reveals that the model's accuracy or learning capability is 0.5, (i.e) 50% of the data is being predited correctly by the model.