```
import numpy as np #linear algebra library of Python
import pandas as pd # build on top of numpy for data analysis, data manipulation and d
import matplotlib.pyplot as plt #plotting library of Python
```

Now let's mount Google drive so that we can upload the diabetes.csv file. You can find the code in the 'Code s

```
from google.colab import drive
drive.mount('/content/gdrive')
```

First thing that we do is take a look at the shape of the dataframe (df.shape) and take a look at first 5 lines th

```
df=pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/diabetes.csv') #import file fi
df.head() #shows first 5 lines including column namesdf.shape # number of rows and col
```

```
df.shape # provides # rows and # columns of the dataframe df - 768 rows and 9 columns
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, ca
```

Now we will assess if the dataset has the same proportion of diabetes vs. non-diabetes cases. At the same t
dataset we note that woman #2 has a skin thickness of zero and this is not realistic. It leads us to believe tha
was available. This does not apply to columns columns 1 and 9 for obvious reasons.

We use a trick to count the non-zero values of the columns. We convert the data type of the dataframe df to t
values to false=0 and all other entries to true=1 . We subsequently add up all True entries per column.

```
df.astype(bool).sum(axis=0) # counts the number of non-zeros for each column while act
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
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

The dataframe is unbalanced as we have 268 ones (diabetes) and thus 500 zeros (no diabetes).

The easiest option could be to eliminate all those patients with zero values, but in this way we would eliminat

Another option is to calculate the median value for a specific column and substitute the zero values for the co

```
median_BMI=df['BMI'].median()
df['BMI']=df['BMI'].replace(to_replace=0, value=median_BMI)

median_BloodPressure=df['BloodPressure'].median()
df['BloodPressure']=df['BloodPressure'].replace(to_replace=0, value=median_BloodPressu

median_Glucose=df['Glucose'].median()
df['Glucose']=df['Glucose'].replace(to_replace=0, value=median_Glucose)

median_SkinThickness=df['SkinThickness'].median()
df['SkinThickness']=df['SkinThickness'].replace(to_replace=0, value=median_SkinThickne

median_Insulin=df['Insulin'].median()
df['Insulin']=df['Insulin'].replace(to_replace=0, value=median_Insulin)
```

```
df.head() #shows first 5 lines including column names
```

The skin thickness of woman #2 is now 23 (median of that column)

Let's create numpy arrays, one for the features (X) and one for the label (y)

```
X=df.drop('Outcome', 1).values #drop 'Outcome' column but you keep the index column
y=df['Outcome'].values
```

We import the train_test_split function from sklearn to split the arrays or matrices into random train and test s

Parameters:
test_size : in our case 20% (default=0.25)

random_state: is basically used for reproducing your problem the same every time it is run. If you do not use a

[illegible]

the split you might get a different set of train and test data points and will not help you in debugging in case y
number does not matter

stratify : array-like or None (default=None) If the number of values belonging to each class are unbalanced, us
basically asking the model to take the training and test set such that the class proportion is same as of the w

```
from sklearn.model_selection import train_test_split #method to split training and tes
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, random_state=42
```

No feature scaling needed when working with Trees

```
#from sklearn.preprocessing import StandardScaler
#sc=StandardScaler()
#X_train=sc.fit_transform(X_train)
#X_test=sc.transform(X_test)
```

Now we are ready to use the GBM algorithm and import de XGBClassifier from the xgboost library from sklea

```
pip install bayesian-optimization
```

```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
import bayes_opt
from bayes_opt import BayesianOptimization
from sklearn.model_selection import cross_val_score
```

'subsample' introduces a parameter that has an analogy with bagging. You dont use the whole dataset for eve
reduce variance. 'colsample_bytree' parameter has an analogy with Random Forests as we do not use all feat
level

```
pbounds = {'n_estimators': (50, 1000), 'eta': (0.01, 3), 'max_depth': (1,32), 'gamma':
model_tune = XGBClassifier(n_jobs=-1) #n_jobs=-1 means that all CPUs are used
def xgboostcv(eta, n_estimators, max_depth, min_child_weight, gamma, subsample, colsam
    return np.mean(cross_val_score(model_tune, X_train, y_train, cv=5, scoring='accura

optimizer = BayesianOptimization(
    f=xgboostcv,
    pbounds=pbounds,
    random_state=1)

optimizer.maximize(
    init_points=2,
    n_iter=5)
print(optimizer.max)
```

```
model = XGBClassifier(eta=2.16, n_estimators=138, max_depth=10, min_child_weight=4, ga
```

```
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Now we will look at other classification KPIs that we discussed in our lessons: Confusion Matrix, ROC, AUC, F

```
from sklearn.metrics import confusion_matrix
y_pred=model.predict(X_test)
confusion_matrix(y_test,y_pred)
```

Classifier not better compared to other classifiers that we tried on same dataset. Using Gridsearch the result