```
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
```

```
feature_labels=['pregnancies', 'Glucose', 'Bloodpressure', 'SkinThickness', 'BMI', 'Di
```

```
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
```

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 c

```
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
```

```
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
```

In KNN we need to scale the features but this is not needed when we deal with Random Forrests so we comm

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

Now we are ready to use the Random Forests algorithm - only parameter that we need to select is n_estimato

```
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train,y_train)
```

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

```
y_pred=model.predict(X_test)
confusion_matrix(y_test,y_pred)
```

Classifier not so good: true positives=32, true negatives=85 false positives=15 and false negatives=22. Recal
TP/(TP+FP)=68% and is too low as well

Now we will estimate the feature importance

```
from sklearn.feature_selection import SelectFromModel
sfm = SelectFromModel(model, threshold=0.2)
sfm.fit(X_train, y_train)
```

```
for feature_list_index in sfm.get_support(indices=True):
    print(feature_labels[feature_list_index])
```