```
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
```

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/creditcard.csv') #import file
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
```


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

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

```
random_state = np.random.RandomState(42)
```

```
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=ra
```

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

```
#x_test=sc.transform(x_test)
```

Now we are ready to use AD algorithm

```
pip install pyOD
```



```
from pyod.models.lof import LOF   # LOF detector
from pyod.models.ocsvm import OCSVM # One Class SVM
from pyod.models.iforest import IForest #Isolated Forests
from pyod.utils.data import evaluate_print
```

```
Classifiers = {'Local Outlier Factor (LOF)': LOF(n_neighbors=20, algorithm='auto', met
               'OneClassSVM': OCSVM (kernel='poly', degree=3 , nu=0.1, max_iter=-1, co
               'Isolation Forests': IForest(n_estimators=100, contamination=0.002, n_j
```

```
for i, (clf_name, clf) in enumerate(Classifiers.items()):
        print()
        print(i + 1, 'fitting', clf_name)

        # fit the data and tag outliers
        clf.fit(X_train) # fit model on training data

        y_train_pred = clf.labels_ # the binary labels of the training data: 0 for inl
        y_train_scores = clf.decision_scores_ # the outlier scores of the training dat

        y_test_pred = clf.predict(X_test) # predict if a particular sample is an outli
```

```
        y_test_scores = clf.decision_function(X_test) # predict raw anomaly score of t
        y_test_proba=clf.predict_proba(X_test) # predict the probability of a sample b

        evaluate_print(clf_name, y_test, y_test_scores)
```