Python for data analysis

ESILV A4 project

DIA3 : Charles DUVAL, Jeanne DEBETTE, Paul CARIOU

Our dataset

Online Shoppers Purchasing Intention Dataset

- 18 columns and 12330 entries
- 14 columns of float or int, 2 booleans and 2 objects

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

Cleaning :

1. Drop null values `dataset.dropna(inplace=True)`
2. Normalizing with MinMaxScaler() from sklearn.processing

```
scaler = MinMaxScaler()
dataset[['Administrative_Duration','Informational_Duration','ProductRelated_Duration']] =
scaler.fit_transform(dataset[['Administrative_Duration','Informational_Duration','ProductRelated_Duration']])
```

3. We change the booleans and objects value into int values

```
# replacing months with the conrresponding number
dataset['Month'] = dataset['Month'].replace(['Jan','Feb','Mar','Apr','May','June','Jul','Aug','Sep','Oct','Nov','Dec'],[1,2,3,4,5,6,7,8,9,10,11,12])
```

```
# categorization of VisitorType, weekend and Revenue
dataset['VisitorType'] = dataset['VisitorType'].replace(['Returning_Visitor','New_Visitor','Other'],[1,2,3])
dataset['Weekend'] = dataset['Weekend'].replace([True,False],[1,0])
dataset['Revenue'] = dataset['Revenue'].replace([True,False],[1,0])
```

Our dataset

4 . Dividing the dataset into features and labels

```
features = dataset.drop('Revenue', axis=1)
labels = dataset['Revenue']
labels = labels.replace([True, False], [1, 0])
labels = labels.astype('int')
np_labels = np.array(labels)
np_features = np.array(features)
np_features = np_features.astype('float')
```

After the cleaning of our dataset :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  int64
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  int64
 16  Weekend                  12330 non-null  int64
 17  Revenue                  12330 non-null  int64
dtypes: float64(7), int64(11)
memory usage: 1.7 MB
```

Our goal

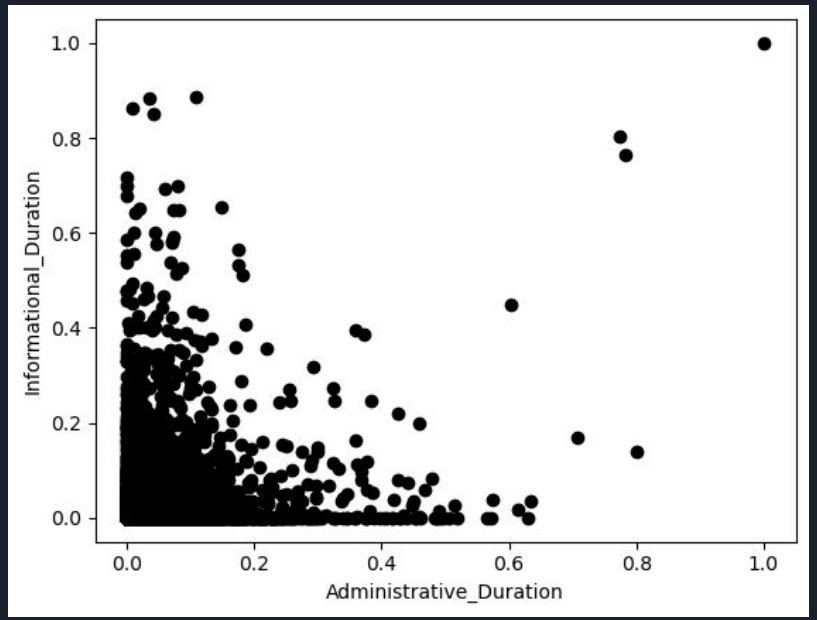
Now that our Dataset is cleaned, what do we want to know of it ?

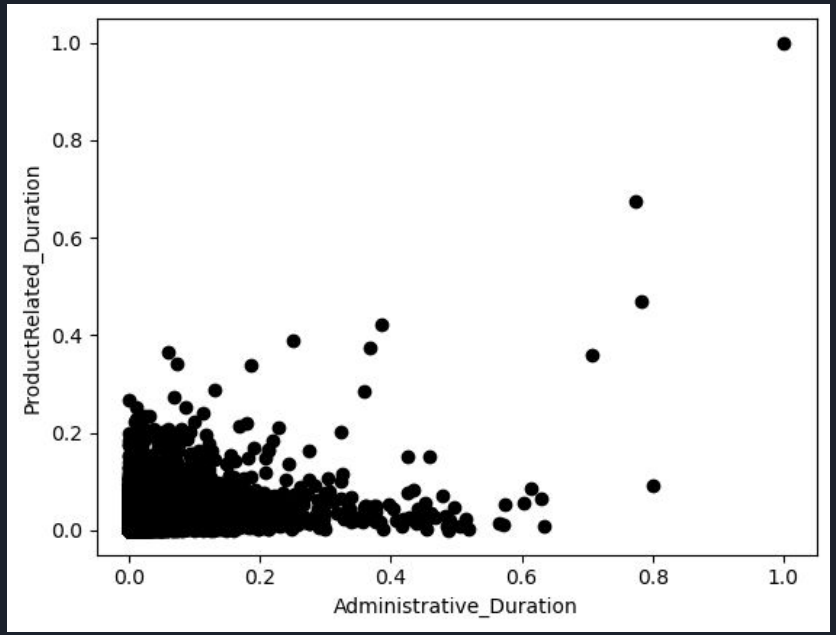
We want to analyse the behavior of customers on the internet in order to predict if a consumer is going to **buy** something or **not**.

How can we do this ?
By **clustering our data** : **Will the customer buy it**, yes or no ?
Since we want a binary answer, we want to get two clusters from our data.

An overview of our data

Plotting different features against each other

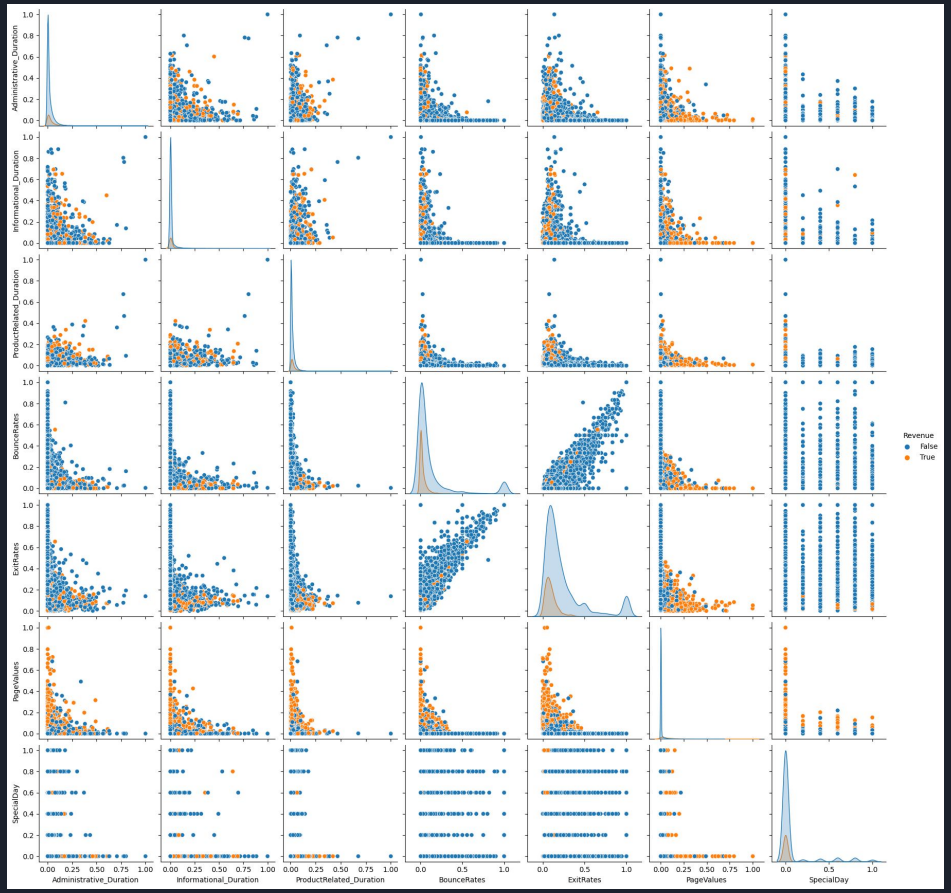




An overview of our data

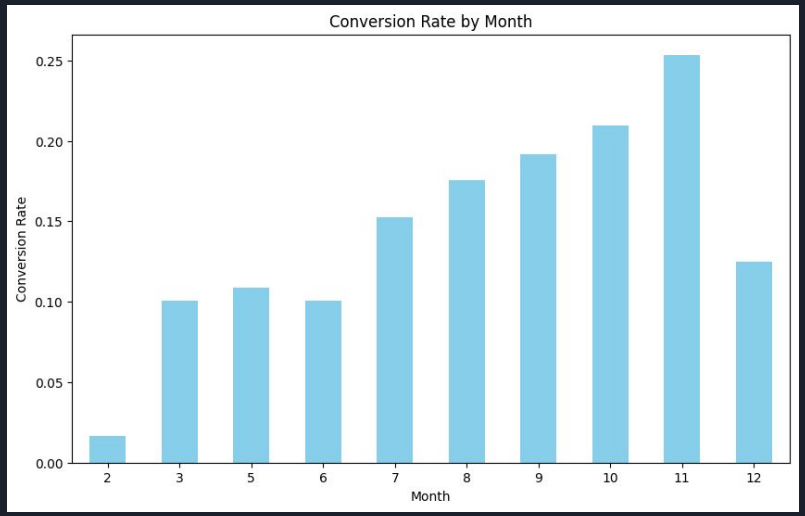
Heatmap w/

pairplot

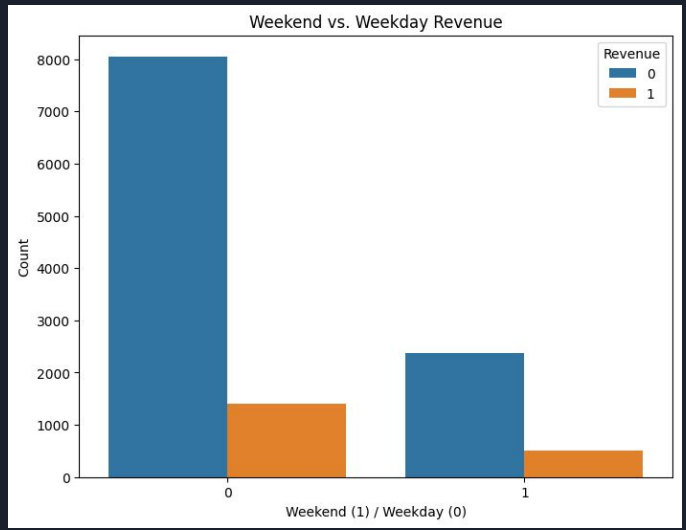


An overview of our data

```
plt.figure(figsize=(10, 6))
conversion_by_month = dataset.groupby('Month')['Revenue'].mean()
conversion_by_month.plot(kind='bar', color='skyblue')
plt.title('Conversion Rate by Month')
plt.xlabel('Month')
plt.ylabel('Conversion Rate')
plt.xticks(rotation=0)
plt.show()
```

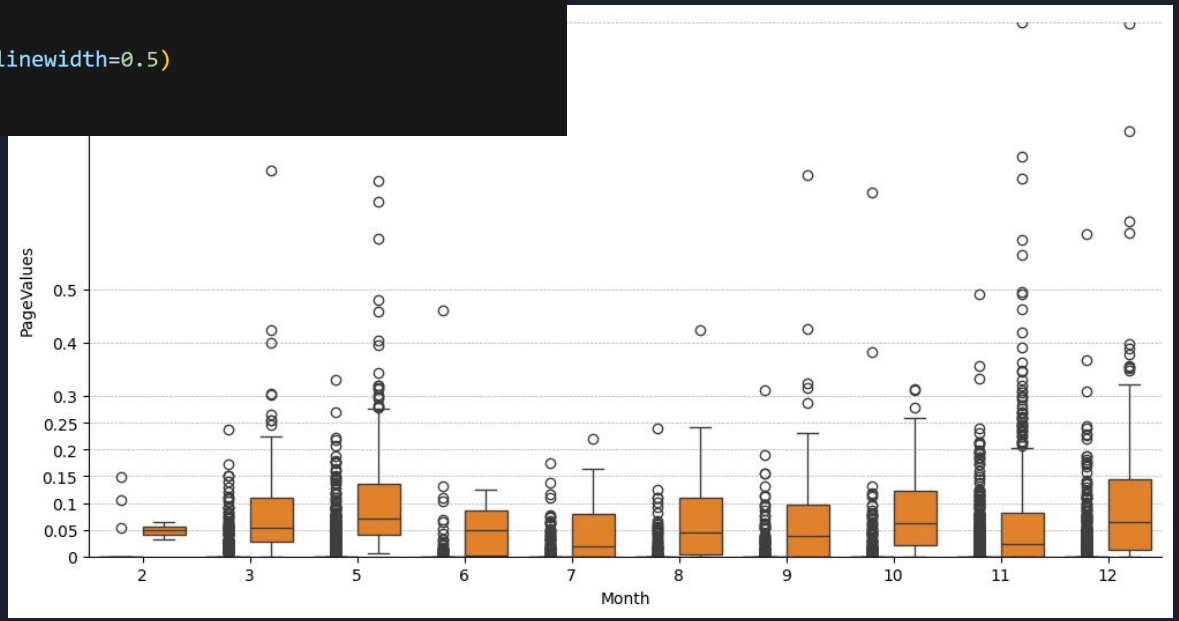



```
plt.figure(figsize=(8, 6))
sns.countplot(x='Weekend', hue='Revenue', data=dataset)
plt.title('Weekend vs. Weekday Revenue')
plt.xlabel('Weekend (1) / Weekday (0)')
plt.ylabel('Count')
plt.show()
```



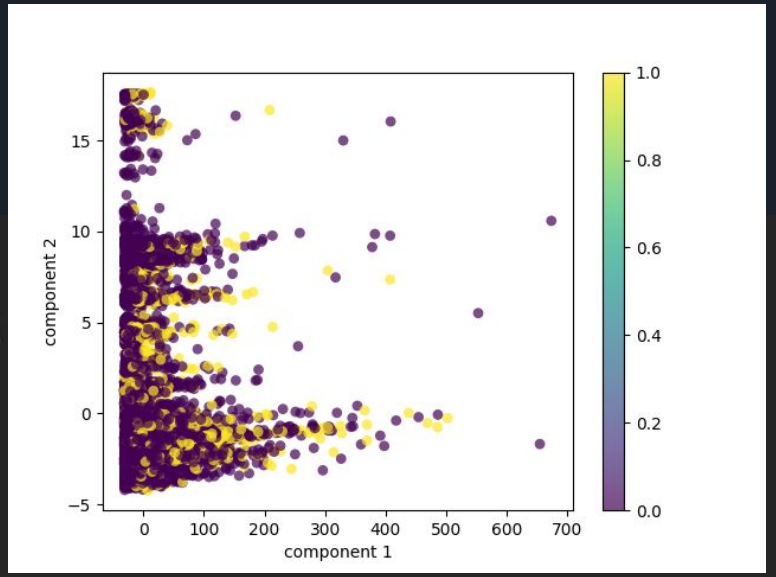
An overview of our data

```
plt.figure(figsize=(12, 6))
sns.boxplot(data=dataset, x='Month', y='PageValues', hue='Revenue')
ax = plt.gca()
ax.set_ylim(bottom=0, top=0.25)
ax.yaxis.set_ticks([0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 1])
ax.yaxis.set_ticklabels(['0', '0.05', '0.1', '0.15', '0.2', '0.25', '0.3', '0.4', '0.5', '1'])
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.yaxis.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.show()
```



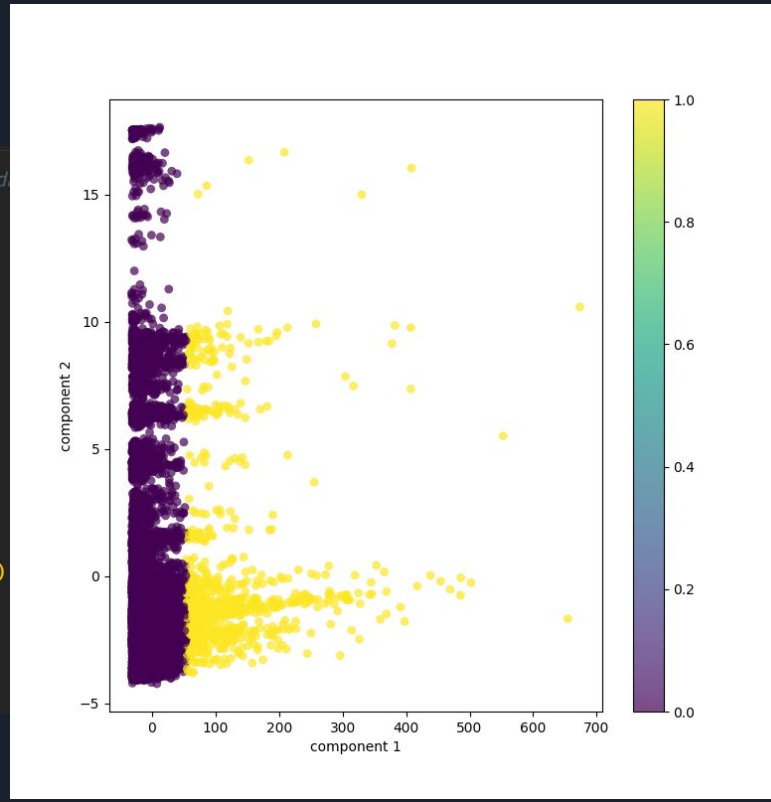
PCA : project in 2D a 18D dataset

```python
# PCA projection to 2D
from sklearn.decomposition import PCA
#create a dataset containing all the non categorical features
pca = PCA(n_components=2)
pca.fit(dataset)
X_pca = pca.transform(dataset)
print("Original shape: {}".format(str(dataset.shape)))
print("Reduced shape: {}".format(str(X_pca.shape)))

# plot first vs. second principal component, colored by class
plt.figure(figsize=(8, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=dataset['Revenue'], edgecolor='none', alpha=0.7, s=40)
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar()
plt.show()
```



KMEANS : first try to classify in two groups

```
# we are going to use dbscan and kmeans clustering algorithms to search for clusters in the d
from sklearn.cluster import DBSCAN, KMeans
dbscan = DBSCAN(eps=0.1, min_samples=5) # problem with the eps value
KMeans = KMeans(n_clusters=2)

dbscanned = dbscan.fit(dataset)
kmeaned = KMeans.fit(dataset)


from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(dataset)
# plot the dbscan clusters
plt.figure(figsize=(8, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeaned.labels_, edgecolor='none', alpha=0.7, s=40)
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar()
plt.show()
```



Another way : logistic regression

- good method for classifying **binary** data
- fast, but can be optimized by choosing a good C (the weight of each iteration)
- We put a fix iter_number in order to speed up the process (without it, it **does not** end)

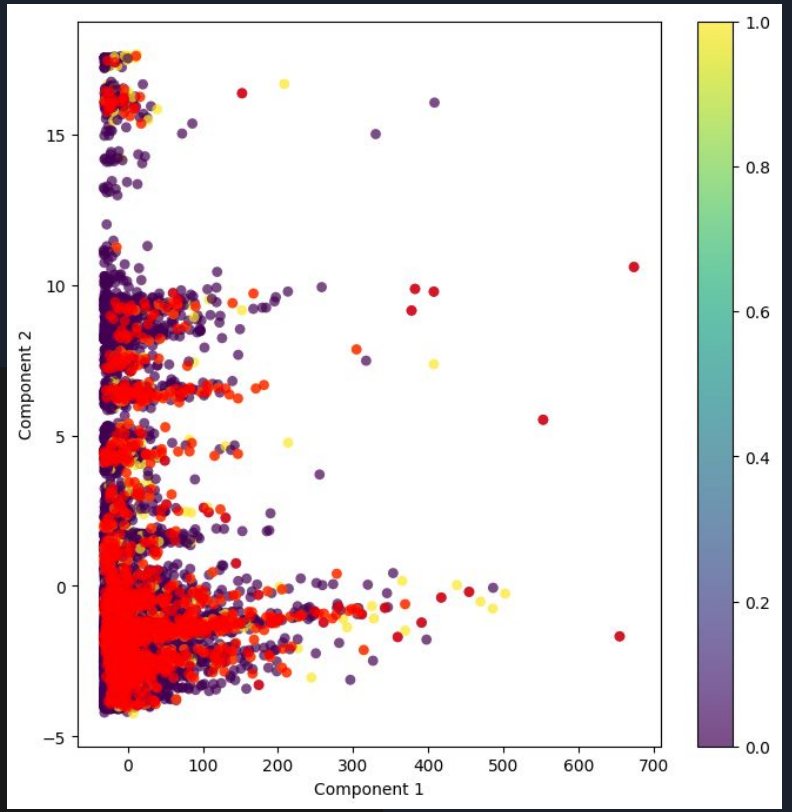
Grid Search : finding the best C for the logistic regression

```python
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Define the parameter grid for Grid Search
param_grid = {'C': [i for i in np.arange(1, 10000,10)]}

# Create the Logistic Regression model
model = LogisticRegression(max_iter=10000)

# Create the Grid Search object
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy', return_train_score=True, n_jobs=-1)

# Fit the Grid Search object to the data
grid_search.fit(np_features, np_labels)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

# Print the best parameters and best score
print("Best Parameters:", best_params)
print("Best Score:", best_score)
```

```python
best_params = {'C': 1121} # hard coded so that the grid search doesn't run every time
best_score = 0.880
```

Logistic Regression

```
# Fit the Logistic Regression model with the best parameter
best_model = LogisticRegression(max_iter=10000, C=best_params['C'])
best_model.fit(np_features, np_labels)

# Perform PCA on the dataset
pca = PCA(n_components=2)
X_pca = pca.fit_transform(np_features)

# Predict the labels using the best model
predicted_labels = best_model.predict(np_features)

# Get the indices of the points that are misclassified
misclassified_indices = np.where(predicted_labels != np_labels)[0]

# Plot the misclassified points using PCA in 2 dimensions
plt.figure(figsize=(8, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=np_labels, edgecolor='none', alpha=0.7, s=40)
plt.scatter(X_pca[misclassified_indices, 0], X_pca[misclassified_indices, 1], c='red', edgecolor='none', alpha=0.7, s=40)
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.colorbar()
plt.show()
```



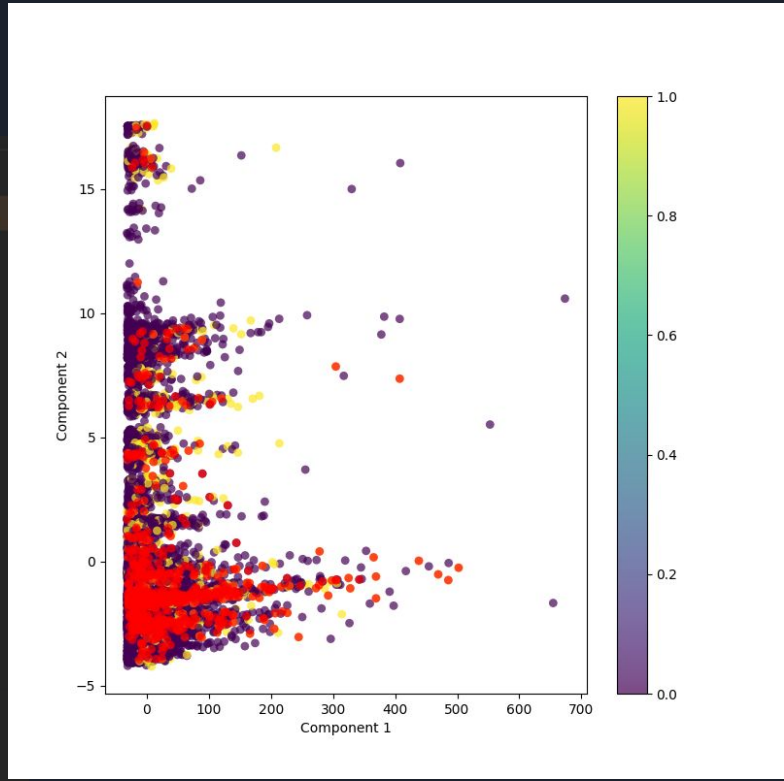
To go even further : **a neural network**

- faster for larger dataset
- works with GPU (graphic processing unit)
- is one of the best way of classifying data with a large number of sample and dimensions.
- library : TENSORFLOW
- notice : it needs to run on a gpu in order to have decent calculus time. DO NOT execute the code without having set it up properly

Tensorflow

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers    Import "tensorflow.keras" could not be resolved
from tensorflow.keras.layers.experimental import preprocessing    Import "tensorflow.keras.layers.experimental" could not be resolved

# split the dataset into train and test
from sklearn.model_selection import train_test_split

train_features, test_features, train_labels, test_labels = train_test_split(np_features, np_labels, test_size=0.2, random_state=42)

# create the model

model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=[17]),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# train the model
history = model.fit(
    train_features, train_labels,
    validation_split=0.2,
    verbose=1, epochs=1000
)
```

```python
# Perform PCA on the dataset
pca = PCA(n_components=2)
X_pca = pca.fit_transform(np_features)

# Predict the labels using the best model
predicted_labels = [1 if i > 0.5 else 0 for i in model.predict(np_features)]

# Get the indices of the points that are misclassified
misclassified_indices = np.where(predicted_labels ≠ np_labels)[0]

# Plot the misclassified points using PCA in 2 dimensions
plt.figure(figsize=(8, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=np_labels, edgecolor='none', alpha=0.7, s=40)
plt.scatter(X_pca[misclassified_indices, 0], X_pca[misclassified_indices, 1], c='red', edgecolor='none', alpha=0.7, s=40)
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.colorbar()
plt.show()
```



Our API

```python
import io
import flask
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import pandas as pd
from flask import Flask, jsonify, request
from tensorflow.keras.models import load_model    Import "tensorflow.keras.models" could not be resolved
from sklearn.linear_model import LogisticRegression
from sklearn.cluster import DBSCAN, KMeans
from sklearn.decomposition import PCA
import seaborn as sns

app = Flask(__name__)

# Tensorflow model
model = load_model('model.h5')

# Load the logistic regression model.
model_2 = LogisticRegression()
best_params = {'C': 1121}
# Load the dataset.
dataset = np.load('dataset.npz')
np_features = dataset['features']
np_labels = dataset['labels']
dataset = pd.DataFrame(np.hstack((np_features, np_labels.reshape(-1, 1))), columns = ['Administrative', 'Administrative_Duration', 'Informational',…

#test the api
@app.route('/test', methods=['GET'])
def test():…

#prediction with tensorflow
@app.route('/tf', methods=['GET'])
def predict():…


@app.route('/scikit-kmeans', methods=['GET'])
def predict_scikit_kmeans():…

@app.route('/scikit-log', methods=['GET'])
def sci_predict_log():…


@app.route('/plot', methods=['GET'])
def plot():…

@app.route('/default', methods=['GET'])
def default():…

@app.route('/', methods=['GET'])
def index():…

if __name__ == '__main__':…
```

- using flask
- Give back json or files for every commands
- Allows to generate and train models based on what you put in the url (everything is explained when launching the website)
- How it works : localhost + /”enter what you want” → /plot, /tf, /scikit-learn, …

Our conclusions

- maximum precision : 90%, with tensorflow (compared to 85% with logistic regression, and nearly 0% for Kmeans or DBscan) → **our prediction are pretty accurate**

- **problems** : tensorflow cannot classify something that is not one of the two categories we are analysing (it cannot classify has “unknown” because of the last layer)

- the dataset was already without any null values

- It might be interesting to augment data for a next time

Thank you for your attention

Do you have any question ?

DIA3 : Charles DUVAL, Jeanne DEBETTE, Paul CARIOU