modelisation

February 13, 2023

[66]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import tensorflow as tf
from sklearn.model_selection import GridSearchCV
```

[67]:
```python
df = pd.read_csv("data_prep.csv")
df = df.drop(columns="Unnamed: 0")
```

[68]:
```python
df.head()
```

[68]:
```
        Age  Sortie Positif  temps_psp  temps_pe  temps_fin  Civilité_Madame  \
0  0.632653               0   0.095385  0.260073   0.480740              1.0
1  0.224490               0   0.043077  0.113553   0.546995              1.0
2  0.918367               0   0.046154  0.194139   0.408320              1.0
3  0.816327               1   0.064615  0.102564   0.303544              0.0
4  0.795918               1   0.052308  0.391941   0.543914              1.0

   Civilité_Monsieur  Type 1er RDV_Entretien individuel  Type 1er RDV_Webcam  \
0                0.0                                1.0                  0.0
1                0.0                                1.0                  0.0
2                0.0                                1.0                  0.0
3                1.0                                1.0                  0.0
4                0.0                                1.0                  0.0

   Taille dernière entreprise :_500 salariés et plus  …  \
0                                               0.0  …
1                                               0.0  …
2                                               0.0  …
3                                               0.0  …
4                                               0.0  …
```

```
   Code Prescripteur_68  Code Prescripteur_69  Code Prescripteur_73  \
0                   0.0                   0.0                   0.0
1                   0.0                   0.0                   0.0
2                   0.0                   0.0                   0.0
3                   0.0                   0.0                   0.0
4                   0.0                   0.0                   0.0

   Code Prescripteur_75  Code Prescripteur_76  Code Prescripteur_78  \
0                   0.0                   0.0                   0.0
1                   0.0                   0.0                   0.0
2                   0.0                   0.0                   0.0
3                   0.0                   0.0                   0.0
4                   0.0                   0.0                   0.0

   Code Prescripteur_92  Code Prescripteur_93  Code Prescripteur_94  \
0                   0.0                   0.0                   0.0
1                   0.0                   0.0                   0.0
2                   0.0                   0.0                   0.0
3                   0.0                   0.0                   0.0
4                   0.0                   0.0                   0.0

   Code Prescripteur_95
0                   0.0
1                   0.0
2                   0.0
3                   0.0
4                   0.0

[5 rows x 48 columns]
```

On choisie les features et le target

[69]:
```
X = df.drop('Sortie Positif', axis=1)
y=df["Sortie Positif"]
```

[70]:
```
print(X.shape)
print(y.shape)
```

```
(5162, 47)
(5162,)
```

[71]:
```
X_train, X_test,y_train,y_test = train_test_split(X,y,test_size=0.
  ↪2,random_state=42)
```

[72]:
```
X_train.describe()
```

```
[72]:                 Age    temps_psp     temps_pe    temps_fin  Civilité_Madame  \
count  4129.000000  4129.000000  4129.000000  4129.000000      4129.000000
mean      0.536499     0.107066     0.156632     0.481519         0.508598
std       0.226685     0.064071     0.099831     0.094687         0.499987
min       0.040816     0.000000     0.073260     0.064715         0.000000
25%       0.346939     0.064615     0.091575     0.446841         0.000000
50%       0.530612     0.095385     0.124542     0.500770         1.000000
75%       0.734694     0.132308     0.179487     0.534669         1.000000
max       1.000000     0.883077     1.000000     1.000000         1.000000

       Civilité_Monsieur  Type 1er RDV_Entretien individuel  \
count        4129.000000                        4129.000000
mean            0.491402                           0.901429
std             0.499987                           0.298121
min             0.000000                           0.000000
25%             0.000000                           1.000000
50%             0.000000                           1.000000
75%             1.000000                           1.000000
max             1.000000                           1.000000

       Type 1er RDV_Webcam  Taille dernière entreprise :_500 salariés et plus  \
count          4129.000000                                        4129.000000
mean              0.098571                                           0.017438
std               0.298121                                           0.130911
min               0.000000                                           0.000000
25%               0.000000                                           0.000000
50%               0.000000                                           0.000000
75%               0.000000                                           0.000000
max               1.000000                                           1.000000

       Taille dernière entreprise :_De 10 à 49 salariés  …  \
count                                       4129.000000  …
mean                                           0.370308  …
std                                            0.482946  …
min                                            0.000000  …
25%                                            0.000000  …
50%                                            0.000000  …
75%                                            1.000000  …
max                                            1.000000  …

       Code Prescripteur_68  Code Prescripteur_69  Code Prescripteur_73  \
count           4129.000000           4129.000000           4129.000000
mean               0.015016              0.043352              0.029789
std                0.121630              0.203673              0.170026
min                0.000000              0.000000              0.000000
25%                0.000000              0.000000              0.000000
50%                0.000000              0.000000              0.000000
```

```
75%                0.000000                0.000000                0.000000
max                1.000000                1.000000                1.000000

       Code Prescripteur_75  Code Prescripteur_76  Code Prescripteur_78  \
count           4129.000000           4129.000000           4129.000000
mean               0.127876              0.049407              0.101720
std                0.333992              0.216742              0.302316
min                0.000000              0.000000              0.000000
25%                0.000000              0.000000              0.000000
50%                0.000000              0.000000              0.000000
75%                0.000000              0.000000              0.000000
max                1.000000              1.000000              1.000000

       Code Prescripteur_92  Code Prescripteur_93  Code Prescripteur_94  \
count           4129.000000           4129.000000           4129.000000
mean               0.040446              0.110681              0.123032
std                0.197026              0.313774              0.328514
min                0.000000              0.000000              0.000000
25%                0.000000              0.000000              0.000000
50%                0.000000              0.000000              0.000000
75%                0.000000              0.000000              0.000000
max                1.000000              1.000000              1.000000

       Code Prescripteur_95
count           4129.000000
mean               0.061274
std                0.239861
min                0.000000
25%                0.000000
50%                0.000000
75%                0.000000
max                1.000000

[8 rows x 47 columns]
```

[73]:
```
X_test.describe()
```

[73]:
```
               Age    temps_psp     temps_pe    temps_fin  Civilité_Madame  \
count  1033.000000  1033.000000  1033.000000  1033.000000      1033.000000
mean      0.520695     0.103775     0.155768     0.478991         0.515973
std       0.229108     0.061339     0.099752     0.098526         0.499987
min       0.000000     0.000000     0.000000     0.000000         0.000000
25%       0.326531     0.064615     0.095238     0.442219         0.000000
50%       0.530612     0.095385     0.124542     0.500770         1.000000
75%       0.714286     0.132308     0.179487     0.533128         1.000000
max       1.000000     1.000000     0.802198     0.755008         1.000000
```

```
       Civilité_Monsieur  Type 1er RDV_Entretien individuel  \
count        1033.000000                        1033.000000   
mean            0.484027                           0.904163   
std             0.499987                           0.294511   
min             0.000000                           0.000000   
25%             0.000000                           1.000000   
50%             0.000000                           1.000000   
75%             1.000000                           1.000000   
max             1.000000                           1.000000   

       Type 1er RDV_Webcam  Taille dernière entreprise :_500 salariés et plus  \
count          1033.000000                                        1033.000000   
mean              0.095837                                           0.027106   
std               0.294511                                           0.162470   
min               0.000000                                           0.000000   
25%               0.000000                                           0.000000   
50%               0.000000                                           0.000000   
75%               0.000000                                           0.000000   
max               1.000000                                           1.000000   

       Taille dernière entreprise :_De 10 à 49 salariés  …  \
count                                       1033.000000  …   
mean                                           0.333979  …   
std                                            0.471861  …   
min                                            0.000000  …   
25%                                            0.000000  …   
50%                                            0.000000  …   
75%                                            1.000000  …   
max                                            1.000000  …   

       Code Prescripteur_68  Code Prescripteur_69  Code Prescripteur_73  \
count           1033.000000           1033.000000           1033.000000   
mean               0.013553              0.046467              0.033882   
std                0.115681              0.210595              0.181013   
min                0.000000              0.000000              0.000000   
25%                0.000000              0.000000              0.000000   
50%                0.000000              0.000000              0.000000   
75%                0.000000              0.000000              0.000000   
max                1.000000              1.000000              1.000000   

       Code Prescripteur_75  Code Prescripteur_76  Code Prescripteur_78  \
count           1033.000000           1033.000000           1033.000000   
mean               0.136496              0.052275              0.093901   
std                0.343480              0.222689              0.291833   
min                0.000000              0.000000              0.000000   
25%                0.000000              0.000000              0.000000   
50%                0.000000              0.000000              0.000000   
```

```
75%                 0.000000               0.000000               0.000000
max                 1.000000               1.000000               1.000000

       Code Prescripteur_92  Code Prescripteur_93  Code Prescripteur_94  \
count           1033.000000           1033.000000           1033.000000
mean               0.037754              0.127783              0.097773
std                0.190693              0.334010              0.297152
min                0.000000              0.000000              0.000000
25%                0.000000              0.000000              0.000000
50%                0.000000              0.000000              0.000000
75%                0.000000              0.000000              0.000000
max                1.000000              1.000000              1.000000

       Code Prescripteur_95
count           1033.000000
mean               0.053243
std                0.224626
min                0.000000
25%                0.000000
50%                0.000000
75%                0.000000
max                1.000000

[8 rows x 47 columns]
```

[74]:
```python
print(y_test.describe(), y_train.describe())
```

```
count    1033.000000
mean        0.246854
std         0.431390
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000
Name: Sortie Positif, dtype: float64 count    4129.000000
mean        0.245338
std         0.430339
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000
Name: Sortie Positif, dtype: float64
```

Test des différents model de machine learning pour trouver celui avec le meilleure score.

Logistic Regression:

[75]:
```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
log_pred = logreg.predict(X_test)
logreg_accuracy = accuracy_score(y_test, log_pred)
print("Accuracy:",logreg_accuracy)
```

```
Accuracy: 0.8489835430784124
```

Random Forest

[76]:
```
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
clf_pred = clf.predict(X_test)
clf_accuracy = accuracy_score(y_test, clf_pred)
print("Accuracy:", clf_accuracy)
```

```
Accuracy: 0.8557599225556631
```

Neural Network

[77]:
```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(64, activation='relu', input_shape=(47, )))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(16, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

[78]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])
```

[79]:
```
neu_net = model.fit(X_train, y_train, epochs=100, batch_size=32,␣
↪validation_data=(X_test, y_test))
```

```
Epoch 1/100
130/130 [==============================] - 1s 3ms/step - loss: 0.5696 -
accuracy: 0.7338 - val_loss: 0.5398 - val_accuracy: 0.7531
Epoch 2/100
130/130 [==============================] - 0s 2ms/step - loss: 0.5336 -
accuracy: 0.7547 - val_loss: 0.5266 - val_accuracy: 0.7531
Epoch 3/100
130/130 [==============================] - 0s 2ms/step - loss: 0.5057 -
accuracy: 0.7605 - val_loss: 0.4860 - val_accuracy: 0.7812
Epoch 4/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4623 -
accuracy: 0.8002 - val_loss: 0.4349 - val_accuracy: 0.8064
Epoch 5/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4121 -
accuracy: 0.8336 - val_loss: 0.4104 - val_accuracy: 0.8219
Epoch 6/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3899 -
```

```
accuracy: 0.8428 - val_loss: 0.4328 - val_accuracy: 0.8267
Epoch 7/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3758 -
accuracy: 0.8489 - val_loss: 0.4203 - val_accuracy: 0.8287
Epoch 8/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3649 -
accuracy: 0.8576 - val_loss: 0.3950 - val_accuracy: 0.8470
Epoch 9/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3525 -
accuracy: 0.8627 - val_loss: 0.4047 - val_accuracy: 0.8403
Epoch 10/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3530 -
accuracy: 0.8607 - val_loss: 0.4007 - val_accuracy: 0.8441
Epoch 11/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3471 -
accuracy: 0.8641 - val_loss: 0.4221 - val_accuracy: 0.8296
Epoch 12/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3465 -
accuracy: 0.8617 - val_loss: 0.4174 - val_accuracy: 0.8345
Epoch 13/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3388 -
accuracy: 0.8704 - val_loss: 0.3994 - val_accuracy: 0.8422
Epoch 14/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3324 -
accuracy: 0.8690 - val_loss: 0.4166 - val_accuracy: 0.8441
Epoch 15/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3329 -
accuracy: 0.8678 - val_loss: 0.4182 - val_accuracy: 0.8412
Epoch 16/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3288 -
accuracy: 0.8724 - val_loss: 0.4045 - val_accuracy: 0.8480
Epoch 17/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3215 -
accuracy: 0.8750 - val_loss: 0.4068 - val_accuracy: 0.8470
Epoch 18/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3168 -
accuracy: 0.8775 - val_loss: 0.4201 - val_accuracy: 0.8374
Epoch 19/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3179 -
accuracy: 0.8791 - val_loss: 0.4236 - val_accuracy: 0.8364
Epoch 20/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3143 -
accuracy: 0.8804 - val_loss: 0.4217 - val_accuracy: 0.8364
Epoch 21/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3154 -
accuracy: 0.8762 - val_loss: 0.4348 - val_accuracy: 0.8219
Epoch 22/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3068 -
```

```
accuracy: 0.8762 - val_loss: 0.4235 - val_accuracy: 0.8335
Epoch 23/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3034 -
accuracy: 0.8789 - val_loss: 0.4244 - val_accuracy: 0.8364
Epoch 24/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3026 -
accuracy: 0.8806 - val_loss: 0.4419 - val_accuracy: 0.8306
Epoch 25/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3011 -
accuracy: 0.8837 - val_loss: 0.4364 - val_accuracy: 0.8267
Epoch 26/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2934 -
accuracy: 0.8879 - val_loss: 0.4814 - val_accuracy: 0.8083
Epoch 27/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3123 -
accuracy: 0.8779 - val_loss: 0.4373 - val_accuracy: 0.8267
Epoch 28/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2908 -
accuracy: 0.8886 - val_loss: 0.4530 - val_accuracy: 0.8190
Epoch 29/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3033 -
accuracy: 0.8787 - val_loss: 0.4520 - val_accuracy: 0.8306
Epoch 30/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2843 -
accuracy: 0.8903 - val_loss: 0.4654 - val_accuracy: 0.8199
Epoch 31/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2832 -
accuracy: 0.8869 - val_loss: 0.4515 - val_accuracy: 0.8296
Epoch 32/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2789 -
accuracy: 0.8910 - val_loss: 0.4593 - val_accuracy: 0.8248
Epoch 33/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2947 -
accuracy: 0.8847 - val_loss: 0.4416 - val_accuracy: 0.8335
Epoch 34/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2713 -
accuracy: 0.8976 - val_loss: 0.4700 - val_accuracy: 0.8248
Epoch 35/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2864 -
accuracy: 0.8869 - val_loss: 0.4479 - val_accuracy: 0.8316
Epoch 36/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2684 -
accuracy: 0.8985 - val_loss: 0.4678 - val_accuracy: 0.8228
Epoch 37/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2692 -
accuracy: 0.8915 - val_loss: 0.4633 - val_accuracy: 0.8364
Epoch 38/100
130/130 [==============================] - 0s 3ms/step - loss: 0.2635 -
```

```
accuracy: 0.8961 - val_loss: 0.4556 - val_accuracy: 0.8325
Epoch 39/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2620 -
accuracy: 0.8980 - val_loss: 0.4829 - val_accuracy: 0.8170
Epoch 40/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2708 -
accuracy: 0.8903 - val_loss: 0.4896 - val_accuracy: 0.8141
Epoch 41/100
130/130 [==============================] - 0s 3ms/step - loss: 0.2647 -
accuracy: 0.8939 - val_loss: 0.4836 - val_accuracy: 0.8170
Epoch 42/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2532 -
accuracy: 0.9009 - val_loss: 0.4728 - val_accuracy: 0.8325
Epoch 43/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2527 -
accuracy: 0.8988 - val_loss: 0.4845 - val_accuracy: 0.8238
Epoch 44/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2503 -
accuracy: 0.8983 - val_loss: 0.5010 - val_accuracy: 0.8209
Epoch 45/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2414 -
accuracy: 0.9077 - val_loss: 0.5096 - val_accuracy: 0.8238
Epoch 46/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2479 -
accuracy: 0.9031 - val_loss: 0.5233 - val_accuracy: 0.8122
Epoch 47/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2366 -
accuracy: 0.9051 - val_loss: 0.5178 - val_accuracy: 0.8296
Epoch 48/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2343 -
accuracy: 0.9041 - val_loss: 0.5568 - val_accuracy: 0.8103
Epoch 49/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2379 -
accuracy: 0.9029 - val_loss: 0.4958 - val_accuracy: 0.8219
Epoch 50/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2319 -
accuracy: 0.9089 - val_loss: 0.5086 - val_accuracy: 0.8296
Epoch 51/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2274 -
accuracy: 0.9089 - val_loss: 0.5360 - val_accuracy: 0.8277
Epoch 52/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2295 -
accuracy: 0.9097 - val_loss: 0.5323 - val_accuracy: 0.8209
Epoch 53/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2538 -
accuracy: 0.8997 - val_loss: 0.5205 - val_accuracy: 0.8316
Epoch 54/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2247 -
```

```
accuracy: 0.9089 - val_loss: 0.5454 - val_accuracy: 0.8228
Epoch 55/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2203 -
accuracy: 0.9097 - val_loss: 0.5347 - val_accuracy: 0.8238
Epoch 56/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2180 -
accuracy: 0.9085 - val_loss: 0.5903 - val_accuracy: 0.7986
Epoch 57/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2221 -
accuracy: 0.9094 - val_loss: 0.5846 - val_accuracy: 0.8035
Epoch 58/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2153 -
accuracy: 0.9138 - val_loss: 0.5535 - val_accuracy: 0.8170
Epoch 59/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2087 -
accuracy: 0.9128 - val_loss: 0.5901 - val_accuracy: 0.8296
Epoch 60/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2124 -
accuracy: 0.9138 - val_loss: 0.5599 - val_accuracy: 0.8209
Epoch 61/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2083 -
accuracy: 0.9135 - val_loss: 0.5832 - val_accuracy: 0.8219
Epoch 62/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2119 -
accuracy: 0.9104 - val_loss: 0.6157 - val_accuracy: 0.8248
Epoch 63/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2077 -
accuracy: 0.9162 - val_loss: 0.5736 - val_accuracy: 0.8074
Epoch 64/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2137 -
accuracy: 0.9104 - val_loss: 0.5792 - val_accuracy: 0.8199
Epoch 65/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2017 -
accuracy: 0.9150 - val_loss: 0.5733 - val_accuracy: 0.8228
Epoch 66/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2036 -
accuracy: 0.9138 - val_loss: 0.5946 - val_accuracy: 0.7996
Epoch 67/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2012 -
accuracy: 0.9152 - val_loss: 0.6219 - val_accuracy: 0.8064
Epoch 68/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1988 -
accuracy: 0.9174 - val_loss: 0.6003 - val_accuracy: 0.8170
Epoch 69/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2007 -
accuracy: 0.9116 - val_loss: 0.5948 - val_accuracy: 0.8219
Epoch 70/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2041 -
```

```
accuracy: 0.9145 - val_loss: 0.6906 - val_accuracy: 0.7754
Epoch 71/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1968 -
accuracy: 0.9174 - val_loss: 0.6080 - val_accuracy: 0.8151
Epoch 72/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1923 -
accuracy: 0.9203 - val_loss: 0.6120 - val_accuracy: 0.8170
Epoch 73/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1877 -
accuracy: 0.9198 - val_loss: 0.6736 - val_accuracy: 0.7899
Epoch 74/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1887 -
accuracy: 0.9208 - val_loss: 0.6588 - val_accuracy: 0.7909
Epoch 75/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1884 -
accuracy: 0.9227 - val_loss: 0.6479 - val_accuracy: 0.8006
Epoch 76/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1884 -
accuracy: 0.9225 - val_loss: 0.6494 - val_accuracy: 0.8161
Epoch 77/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1887 -
accuracy: 0.9164 - val_loss: 0.7197 - val_accuracy: 0.7986
Epoch 78/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1850 -
accuracy: 0.9227 - val_loss: 0.6554 - val_accuracy: 0.7977
Epoch 79/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1814 -
accuracy: 0.9210 - val_loss: 0.6519 - val_accuracy: 0.7986
Epoch 80/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1850 -
accuracy: 0.9213 - val_loss: 0.6245 - val_accuracy: 0.8170
Epoch 81/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1830 -
accuracy: 0.9256 - val_loss: 0.7092 - val_accuracy: 0.7812
Epoch 82/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1949 -
accuracy: 0.9152 - val_loss: 0.6443 - val_accuracy: 0.8093
Epoch 83/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1813 -
accuracy: 0.9259 - val_loss: 0.7121 - val_accuracy: 0.7890
Epoch 84/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1777 -
accuracy: 0.9259 - val_loss: 0.6892 - val_accuracy: 0.8190
Epoch 85/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1781 -
accuracy: 0.9230 - val_loss: 0.6628 - val_accuracy: 0.7996
Epoch 86/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1937 -
```

```
accuracy: 0.9179 - val_loss: 0.6831 - val_accuracy: 0.7880
Epoch 87/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1710 -
accuracy: 0.9298 - val_loss: 0.6877 - val_accuracy: 0.8045
Epoch 88/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1709 -
accuracy: 0.9290 - val_loss: 0.6877 - val_accuracy: 0.8170
Epoch 89/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1710 -
accuracy: 0.9254 - val_loss: 0.7343 - val_accuracy: 0.8015
Epoch 90/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1773 -
accuracy: 0.9266 - val_loss: 0.6999 - val_accuracy: 0.8064
Epoch 91/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1803 -
accuracy: 0.9235 - val_loss: 0.6888 - val_accuracy: 0.8180
Epoch 92/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1653 -
accuracy: 0.9298 - val_loss: 0.7638 - val_accuracy: 0.7725
Epoch 93/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1800 -
accuracy: 0.9247 - val_loss: 0.6962 - val_accuracy: 0.8180
Epoch 94/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1643 -
accuracy: 0.9312 - val_loss: 0.7263 - val_accuracy: 0.8180
Epoch 95/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1623 -
accuracy: 0.9315 - val_loss: 0.7349 - val_accuracy: 0.8103
Epoch 96/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1615 -
accuracy: 0.9324 - val_loss: 0.6774 - val_accuracy: 0.8141
Epoch 97/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1632 -
accuracy: 0.9317 - val_loss: 0.7184 - val_accuracy: 0.8190
Epoch 98/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1566 -
accuracy: 0.9334 - val_loss: 0.7694 - val_accuracy: 0.8006
Epoch 99/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1554 -
accuracy: 0.9361 - val_loss: 0.8006 - val_accuracy: 0.7909
Epoch 100/100
130/130 [==============================] - 0s 2ms/step - loss: 0.1553 -
accuracy: 0.9346 - val_loss: 0.8448 - val_accuracy: 0.7890
```

[80]:

```python
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print('Test Accuracy:', test_accuracy)
```

```
33/33 [==============================] - 0s 1ms/step - loss: 0.8448 - accuracy:
```

```
0.7890
Test Accuracy: 0.7889641523361206
```

Nous pouvons voir que le modèle avec le meilleure score c'est le "Random Forest Classifier", nous allons maintenant essayer d'optimiser le modèle pour avoir un meilleure score.

[81]:
```
clf = RandomForestClassifier(random_state=42)
param_grid = {
    "n_estimators": [10,50,100,200,300],
    "max_depth": [None, 5, 10, 20],
    "min_samples_split": [2,5,10],
    "min_samples_leaf": [1,2,4]
}
```

[82]:
```
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

[82]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [None, 5, 10, 20],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [10, 50, 100, 200, 300]},
             scoring='accuracy')
```

[83]:
```
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_
print("Best parameters:", best_params)
print("Best accuracy:", best_accuracy)
```

```
Best parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split':
5, 'n_estimators': 300}
Best accuracy: 0.8542020691173233
```

[84]:
```
clf = RandomForestClassifier(**best_params)
clf.fit(X_train, y_train)
clf_predict = clf.predict(X_test)
clf_accuracy = accuracy_score(y_test, clf_pred)
print("Accuracy:", clf_accuracy)
```

```
Accuracy: 0.8557599225556631
```

[85]:
```
cm = confusion_matrix(y_test, clf_predict)
print("Confusion Matrix:\n", cm)
```

```
Confusion Matrix:
 [[734  44]
 [103 152]]
```

[86]:
```
(730 + 153) / (730 + 48 + 102 + 153)
```

[86]: 0.8547918683446273

[87]:
```
cr = classification_report(y_test, clf_predict)
print("Classification Report:\n", cm)
```

```
Classification Report:
 [[734  44]
 [103 152]]
```

Feature Importance

[88]:
```
importances = clf.feature_importances_
importances_df = pd.DataFrame(importances, index=X_train.columns,␣
 ↪columns=["importance"])
importances_df.sort_values(by="importance", ascending=False, inplace=True)
print(importances_df.head(10))
```

```
                                                   importance
temps_fin                                            0.481955
Age                                                  0.095971
temps_psp                                            0.082739
temps_pe                                             0.080236
Taille dernière entreprise :_Moins de 10 salariés    0.011465
Taille dernière entreprise :_De 10 à 49 salariés     0.011398
Secteur_SUPPORT A L'ENTREPRISE                       0.010875
Taille dernière entreprise :_De 50 à 499 salariés    0.010841
Secteur_COMMERCE, VENTE ET GRANDE DISTRIBUTION       0.010295
Civilité_Monsieur                                    0.009004
```

[89]:
```
XX_train = X_train.drop(columns="temps_fin")
XX_test = X_test.drop(columns="temps_fin")
```

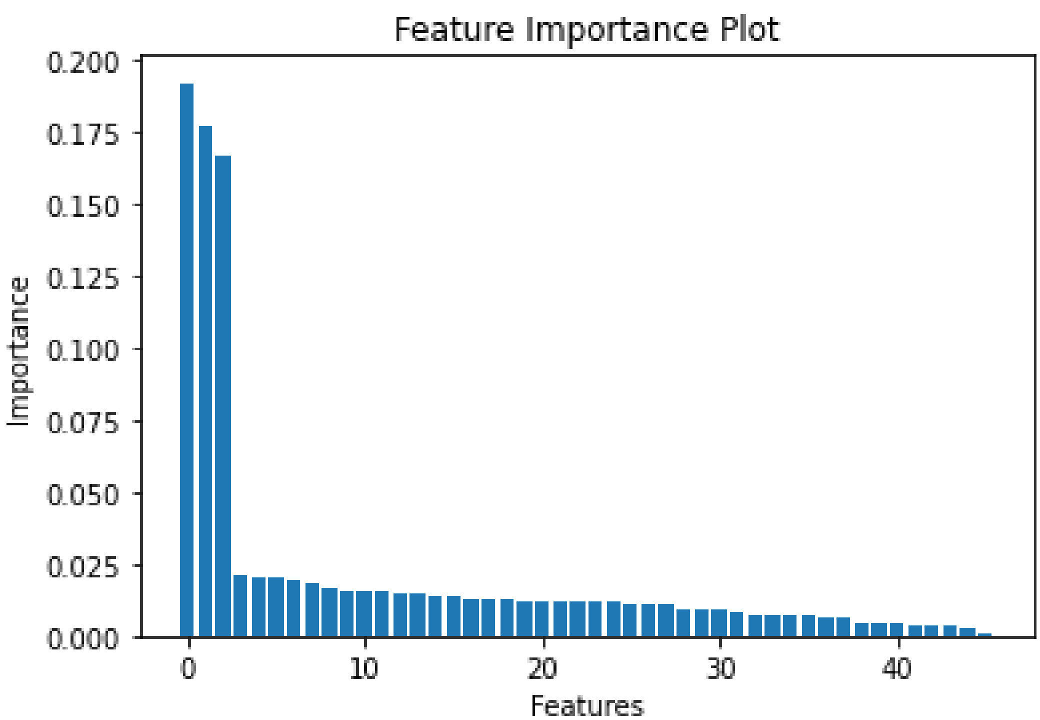
[90]:
```
clf = RandomForestClassifier(**best_params)
clf.fit(XX_train, y_train)
clf_predict = clf.predict(XX_test)
clf_accuracy = accuracy_score(y_test, clf_pred)
print("Accuracy:", clf_accuracy)
```

```
Accuracy: 0.8557599225556631
```

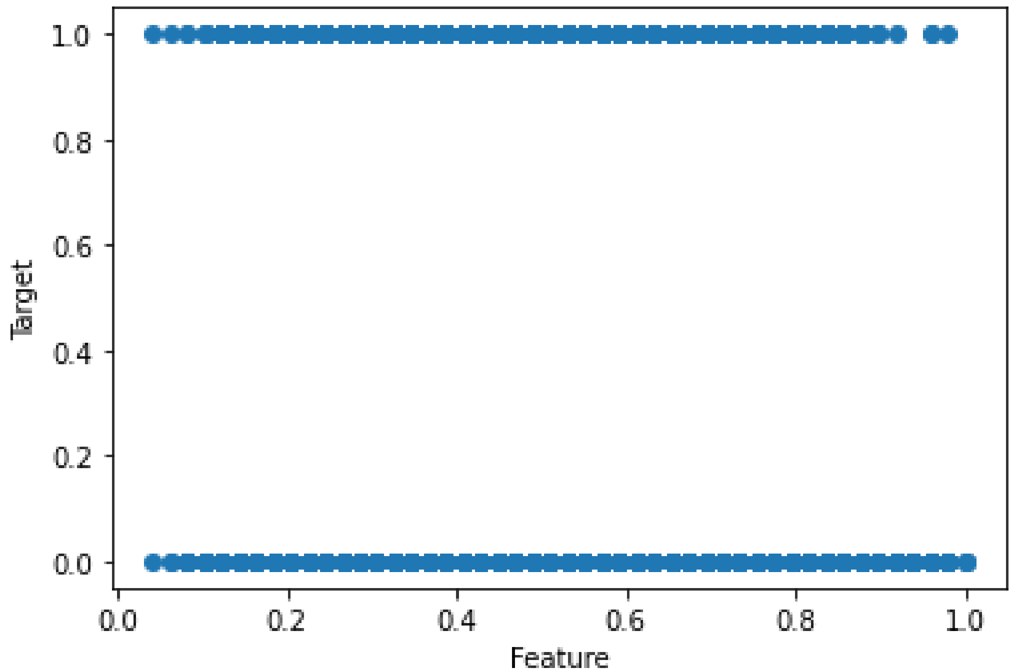
[91]:
```
importances = clf.feature_importances_
importances_df = pd.DataFrame(importances, index=XX_train.columns,␣
 ↪columns=["importance"])
importances_df.sort_values(by="importance", ascending=False, inplace=True)
print(importances_df.head(10))
```

```
                                                    importance
Age                                                   0.191761
temps_psp                                             0.176529
temps_pe                                              0.166905
Taille dernière entreprise :_De 10 à 49 salariés      0.021496
Taille dernière entreprise :_Moins de 10 salariés     0.020301
Taille dernière entreprise :_De 50 à 499 salariés     0.019963
Secteur_SUPPORT A L'ENTREPRISE                        0.019315
Secteur_COMMERCE, VENTE ET GRANDE DISTRIBUTION        0.018308
Civilité_Madame                                       0.016132
Secteur_HÔTELLERIE- RESTAURATION TOURISME LOISI…    0.015759
```

[92]:
```
sorted_importances = sorted(importances, reverse=True)
plt.bar(range(XX_train.shape[1]), sorted_importances)
plt.xlabel("Features")
plt.ylabel("Importance")
plt.title("Feature Importance Plot")
plt.show()
```



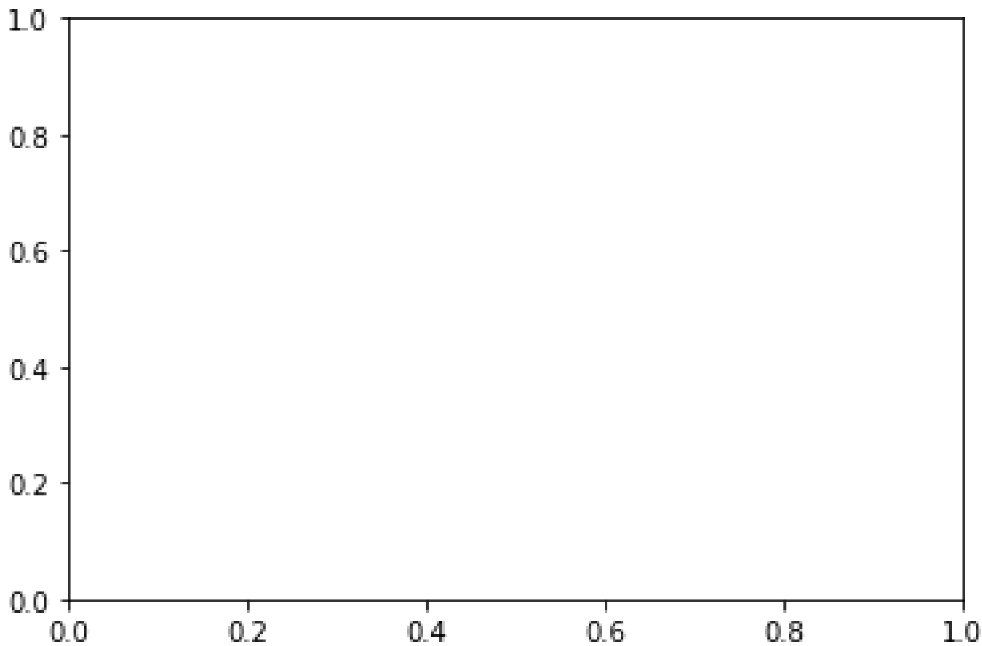
[93]:
```
plt.scatter(XX_train['Age'], y_train)
plt.xlabel("Feature")
plt.ylabel("Target")
plt.show()
```



[77]:
```
for target in [0,1]:
    plt.scatter(XX_train[y_train==target], y_train[y_train==target],␣
 ↪label=target)
plt.xlabel("Age")
plt.ylabel("Target (0 or 1)")
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
c:\Users\NicolasFUENTES\Documents\Python\Memoire\Modelisation.ipynb Cell 35 in␣
 ↪<cell line: 1>()
      <a href='vscode-notebook-cell:/c%3A/Users/NicolasFUENTES/Documents/Python/
 ↪Memoire/Modelisation.ipynb#X53sZmlsZQ%3D%3D?line=0'>1</a> for target in [0,1]
----> <a href='vscode-notebook-cell:/c%3A/Users/NicolasFUENTES/Documents/Python/
 ↪Memoire/Modelisation.ipynb#X53sZmlsZQ%3D%3D?line=1'>2</a>     plt.
 ↪scatter(XX_train[y_train==target], y_train[y_train==target], label=target)
      <a href='vscode-notebook-cell:/c%3A/Users/NicolasFUENTES/Documents/Python/
 ↪Memoire/Modelisation.ipynb#X53sZmlsZQ%3D%3D?line=2'>3</a> plt.xlabel("Age")
      <a href='vscode-notebook-cell:/c%3A/Users/NicolasFUENTES/Documents/Python/
 ↪Memoire/Modelisation.ipynb#X53sZmlsZQ%3D%3D?line=3'>4</a> plt.ylabel("Target␣
 ↪(0 or 1)")

File c:
 ↪\Users\NicolasFUENTES\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\
 ↪py:2807, in scatter(x, y, s, c, marker, cmap, norm, vmin, vmax, alpha,␣
 ↪linewidths, edgecolors, plotnonfinite, data, **kwargs)
```

```
   2802 @_copy_docstring_and_deprecators(Axes.scatter)
   2803 def scatter(
   2804         x, y, s=None, c=None, marker=None, cmap=None, norm=None,
   2805         vmin=None, vmax=None, alpha=None, linewidths=None, *,
   2806         edgecolors=None, plotnonfinite=False, data=None, **kwargs):
-> 2807     __ret = gca().scatter(
   2808         x, y, s=s, c=c, marker=marker, cmap=cmap, norm=norm,
   2809         vmin=vmin, vmax=vmax, alpha=alpha, linewidths=linewidths,
   2810         edgecolors=edgecolors, plotnonfinite=plotnonfinite,
   2811         **({"data": data} if data is not None else {}), **kwargs)
   2812     sci(__ret)
   2813     return __ret

File c:
 ↪\Users\NicolasFUENTES\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\
 ↪py:1412, in _preprocess_data.<locals>.inner(ax, data, *args, **kwargs)
   1409 @functools.wraps(func)
   1410 def inner(ax, *args, data=None, **kwargs):
   1411     if data is None:
-> 1412         return func(ax, *map(sanitize_sequence, args), **kwargs)
   1414     bound = new_sig.bind(ax, *args, **kwargs)
   1415     auto_label = (bound.arguments.get(label_namer)
   1416                   or bound.kwargs.get(label_namer))

File c:
 ↪\Users\NicolasFUENTES\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\
 ↪py:4369, in Axes.scatter(self, x, y, s, c, marker, cmap, norm, vmin, vmax,␣
 ↪alpha, linewidths, edgecolors, plotnonfinite, **kwargs)
   4367 y = np.ma.ravel(y)
   4368 if x.size != y.size:
-> 4369     raise ValueError("x and y must be the same size")
   4371 if s is None:
   4372     s = (20 if rcParams['_internal.classic_mode'] else
   4373          rcParams['lines.markersize'] ** 2.0)

ValueError: x and y must be the same size
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0