diabetes

September 24, 2024

[1]:
```
import pandas as pd
import numpy as np
```

1 Importation des données à partir d'un fichier csv présent dans le dossier

[2]:
```
data = pd.read_csv("diabetes.csv")
data.head()
```

[2]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

1.1 Identification des caractéristiques et de la variable cible

[3]:
```
features_names = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness",␣
↪"Insulin", "BMI", "DiabetesPedigreeFunction", "Age"]
X = data[features_names]
y = data["Outcome"]
```

2 Exploration

[4]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
dtypes: float64(2), int64(6)
memory usage: 48.1 KB
```

Il n'y a pas de valeur manquante, tous les types des données sont numériques.

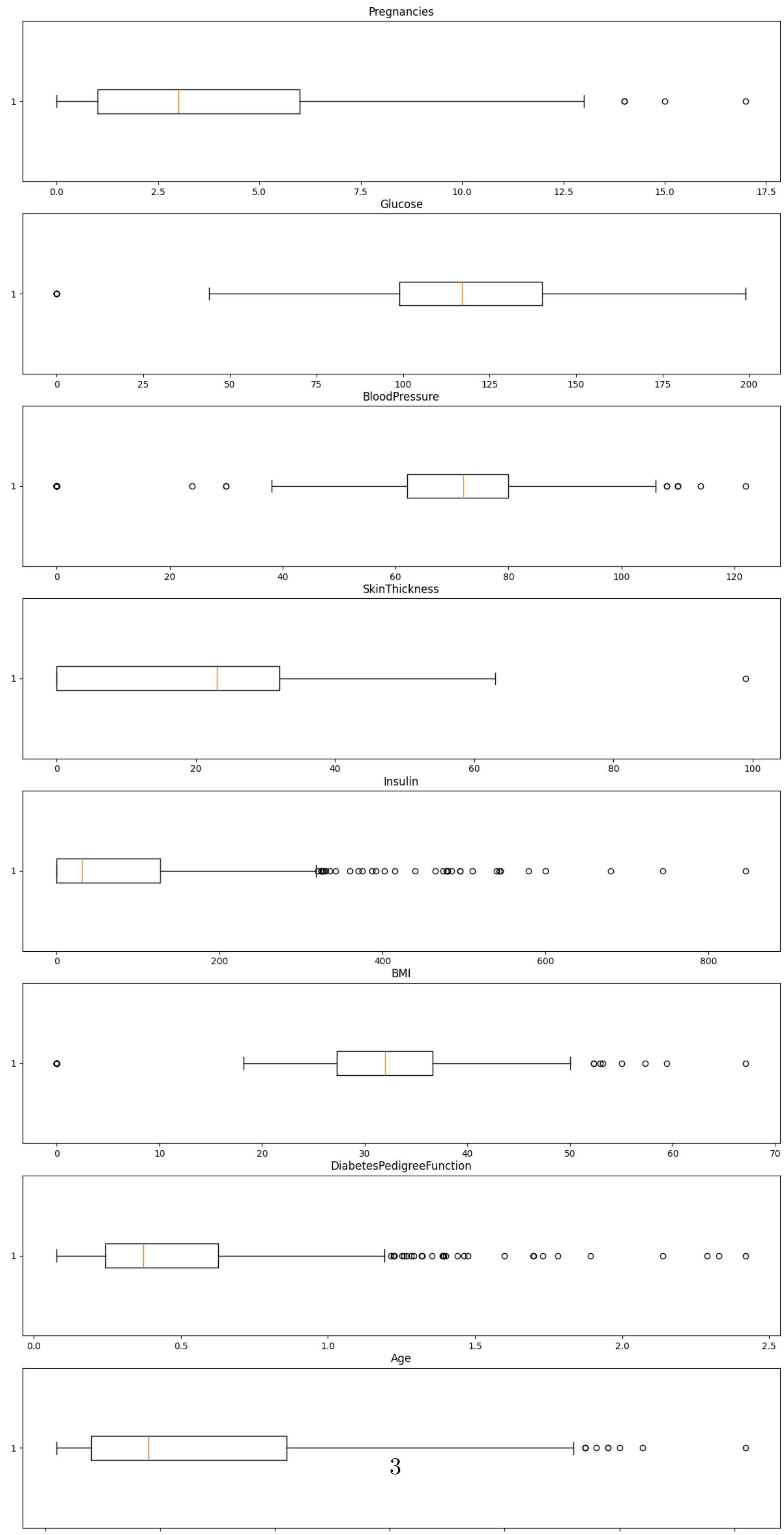
[5]:
```python
X.describe()
```

[5]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age
count  768.000000                768.000000  768.000000
mean    31.992578                  0.471876   33.240885
std      7.884160                  0.331329   11.760232
min      0.000000                  0.078000   21.000000
25%     27.300000                  0.243750   24.000000
50%     32.000000                  0.372500   29.000000
75%     36.600000                  0.626250   41.000000
max     67.100000                  2.420000   81.000000
```

[6]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(15,30))
for i, name in enumerate(features_names):
    plt.subplot(len(features_names), 1, i+1)
    plt.title(name)
    plt.boxplot(data[name], vert=False)
```


Pregnancies
1
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Glucose
1
0 25 50 75 100 125 150 175 200
BloodPressure
1
0 20 40 60 80 100 120
SkinThickness
1
0 20 40 60 80 100
Insulin
1
0 200 400 600 800
BMI
1
0 10 20 30 40 50 60 70
DiabetesPedigreeFunction
1
0.0 0.5 1.0 1.5 2.0 2.5
Age
1
20 30 40 50 60 70 80

[7]:
```
data.loc[data["Outcome"] == 1, "Outcome"].count()
```

[7]: 268

[8]:
```
data.loc[data["Outcome"] == 0, "Outcome"].count()
```

[8]: 500

Il y a deux fois moins de données pour les cas de diabète par rapport aux cas négatifs ce qui peux biaser le résultat. La solution est d'enrichir la classe minoritaire ou d'appauvrir la majoritaire. Ici au choisi le 1er cas pour ne pas perdre de données.

2.0.1 Séparation en jeu d'entrainement et de test et création de nouveaux exemples synthétiques pour la classe minoritaire (ici outcome=1 donc cas diabétiques)

[9]:
```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

X_train, X_test, y_train, y_test = train_test_split(data[features_names],␣
 ↪data["Outcome"], test_size=0.2, random_state=24 )
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

2.0.2 Pour que la classification se passe bien nous allons normaliser et standardiser les données.

[10]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_resampled_scaled = scaler.fit_transform(X_resampled)
X_test_scaled = scaler.transform(X_test)
```

3 Utilisant la validation croisée puis entrainement du modèle

3.1 1 Modèle de régression logistique

[11]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
scores = cross_val_score(model, X_resampled_scaled, y_resampled, cv=5)
print(f'Mean accurancy: {scores.mean()}')
print(f'Accurancy: {scores}')
```

```
model.fit(X_resampled_scaled, y_resampled)
```

```
Mean accurancy: 0.7474534161490685
Accurancy: [0.74534161 0.82608696 0.69565217 0.77018634 0.7       ]
```

[11]:
```
LogisticRegression()
```

[12]:
```
y_pred = model.predict(X_test_scaled)
```

3.1.1 Affichage des différents scores

[21]:
```
from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪confusion_matrix, classification_report

def print_metric(y_test, y_pred):
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)

    print(f"Precision score: {precision}")
    print(f"Recall score: {recall}")
    print(f"f1 score: {f1}")
    print(f"Confusion matrix: \n{cm}")

    print("Rapport de classification :\n", classification_report(y_test,␣
 ↪y_pred))

print_metric(y_test, y_pred)
```

```
Precision score: 0.6481481481481481
Recall score: 0.625
f1 score: 0.6363636363636364
Confusion matrix:
[[79 19]
 [21 35]]
Rapport de classification :
               precision    recall  f1-score   support

           0       0.79      0.81      0.80        98
           1       0.65      0.62      0.64        56

    accuracy                           0.74       154
   macro avg       0.72      0.72      0.72       154
weighted avg       0.74      0.74      0.74       154
```

3.2 2 Algorithme des K-Nearest Neighbors (KNN)

[18]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

knn = KNeighborsClassifier()
param_grid = {'n_neighbors': [2, 3, 4, 5, 7, 9]}
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_resampled_scaled, y_resampled)
print(f"Best k: {grid_search.best_params_}")
```

```
Best k: {'n_neighbors': 5}
```

[19]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_resampled_scaled, y_resampled)
y_knn_pred = knn.predict(X_test_scaled)
```

3.2.1 Affichage des différents scores

[22]:
```
print_metric(y_test, y_knn_pred)
```

```
Precision score: 0.6229508196721312
Recall score: 0.6785714285714286
f1 score: 0.6495726495726496
Confusion matrix:
[[75 23]
 [18 38]]
Rapport de classification :
              precision    recall  f1-score   support

           0       0.81      0.77      0.79        98
           1       0.62      0.68      0.65        56

    accuracy                           0.73       154
   macro avg       0.71      0.72      0.72       154
weighted avg       0.74      0.73      0.74       154
```

L'algorithme des k plus proches voisins a plus de précision mais un moins bon rappel, il y a moins de faux positifs mais plus de faux négatifs.

[43]:
```
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    'n_estimators': [50, 100, 150, 200, 300],
    'max_depth': [2, 40, 50, 70, 100, 150],
    'max_features': [None, 'sqrt', 'log2']
}
```

```
grid_search = GridSearchCV(RandomForestClassifier(random_state=24), param_grid,␣
↪cv=5, scoring='recall')
grid_search.fit(X_resampled_scaled, y_resampled)
print(f'Best parameters: {grid_search.best_params_}')
```

```
Best parameters: {'max_depth': 40, 'max_features': 'sqrt', 'n_estimators': 300}
```

[39]:
```
rf = RandomForestClassifier(n_estimators=300, random_state=24,␣
↪max_features='sqrt', max_depth=40)
rf.fit(X_resampled_scaled, y_resampled)
y_rf_pred = rf.predict(X_test_scaled)
```

[40]:
```
print_metric(y_test, y_rf_pred)
```

```
Precision score: 0.5849056603773585
Recall score: 0.5535714285714286
f1 score: 0.5688073394495413
Confusion matrix:
[[76 22]
 [25 31]]
Rapport de classification :
              precision    recall  f1-score   support

           0       0.75      0.78      0.76        98
           1       0.58      0.55      0.57        56

    accuracy                           0.69       154
   macro avg       0.67      0.66      0.67       154
weighted avg       0.69      0.69      0.69       154
```

Le modèle des arbres a aussi plus de précision mais un moins bon rappel, il y a moins de faux positifs mais plus de faux négatifs.

[ ]:
```

```