8enqwv4b8

March 12, 2025

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#!pip install seaborn
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪StratifiedKFold
from sklearn.preprocessing import StandardScaler
```

[2]:
```
df = pd.read_csv("alzheimers_disease_data.csv")
```

[3]:
```
df.head()
```

[3]:
```
   PatientID  Age  Gender  Ethnicity  EducationLevel        BMI  Smoking  \
0       4751   73       0          0               2  22.927749        0
1       4752   89       0          0               0  26.827681        0
2       4753   73       0          3               1  17.795882        0
3       4754   74       1          0               1  33.800817        1
4       4755   89       0          0               0  20.716974        0

   AlcoholConsumption  PhysicalActivity  DietQuality  …  MemoryComplaints  \
0           13.297218          6.327112     1.347214  …                 0
1            4.542524          7.619885     0.518767  …                 0
2           19.555085          7.844988     1.826335  …                 0
3           12.209266          8.428001     7.435604  …                 0
4           18.454356          6.310461     0.795498  …                 0

   BehavioralProblems       ADL  Confusion  Disorientation  \
0                   0  1.725883          0               0
1                   0  2.592424          0               0
2                   0  7.119548          0               1
3                   1  6.481226          0               0
4                   0  0.014691          0               0

   PersonalityChanges  DifficultyCompletingTasks  Forgetfulness  Diagnosis  \
0                   0                          1              0          0
1                   0                          0              1          0
```

```
2                  0                          1              0          0
3                  0                          0              0          0
4                  1                          1              0          0

   DoctorInCharge
0       XXXConfid
1       XXXConfid
2       XXXConfid
3       XXXConfid
4       XXXConfid

[5 rows x 35 columns]
```

[4]:
```
df.dtypes
```

[4]:
```
PatientID                      int64
Age                            int64
Gender                         int64
Ethnicity                      int64
EducationLevel                 int64
BMI                          float64
Smoking                        int64
AlcoholConsumption           float64
PhysicalActivity             float64
DietQuality                  float64
SleepQuality                 float64
FamilyHistoryAlzheimers        int64
CardiovascularDisease          int64
Diabetes                       int64
Depression                     int64
HeadInjury                     int64
Hypertension                   int64
SystolicBP                     int64
DiastolicBP                    int64
CholesterolTotal             float64
CholesterolLDL               float64
CholesterolHDL               float64
CholesterolTriglycerides     float64
MMSE                         float64
FunctionalAssessment         float64
MemoryComplaints               int64
BehavioralProblems             int64
ADL                          float64
Confusion                      int64
Disorientation                 int64
PersonalityChanges             int64
DifficultyCompletingTasks      int64
```

```
Forgetfulness                               int64
Diagnosis                                   int64
DoctorInCharge                             object
dtype: object
```

[5]:
```python
# Vérification des valeurs manquantes
print("\nValeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Valeurs manquantes par colonne:
PatientID                    0
Age                          0
Gender                       0
Ethnicity                    0
EducationLevel               0
BMI                          0
Smoking                      0
AlcoholConsumption           0
PhysicalActivity             0
DietQuality                  0
SleepQuality                 0
FamilyHistoryAlzheimers      0
CardiovascularDisease        0
Diabetes                     0
Depression                   0
HeadInjury                   0
Hypertension                 0
SystolicBP                   0
DiastolicBP                  0
CholesterolTotal             0
CholesterolLDL               0
CholesterolHDL               0
CholesterolTriglycerides     0
MMSE                         0
FunctionalAssessment         0
MemoryComplaints             0
BehavioralProblems           0
ADL                          0
Confusion                    0
Disorientation               0
PersonalityChanges           0
DifficultyCompletingTasks    0
Forgetfulness                0
Diagnosis                    0
DoctorInCharge               0
dtype: int64
```

[6]:
```
# Drop colonnes inutiles
df = df.drop(['PatientID','DoctorInCharge'],axis=1)
```

[7]:
```
df
```

[7]:
```
      Age  Gender  Ethnicity  EducationLevel        BMI  Smoking  \
0      73       0          0               2  22.927749        0
1      89       0          0               0  26.827681        0
2      73       0          3               1  17.795882        0
3      74       1          0               1  33.800817        1
4      89       0          0               0  20.716974        0
…    …     …       …             …        …        …
2144   61       0          0               1  39.121757        0
2145   75       0          0               2  17.857903        0
2146   77       0          0               1  15.476479        0
2147   78       1          3               1  15.299911        0
2148   72       0          0               2  33.289738        0

      AlcoholConsumption  PhysicalActivity  DietQuality  SleepQuality  …  \
0              13.297218          6.327112     1.347214      9.025679  …
1               4.542524          7.619885     0.518767      7.151293  …
2              19.555085          7.844988     1.826335      9.673574  …
3              12.209266          8.428001     7.435604      8.392554  …
4              18.454356          6.310461     0.795498      5.597238  …
…                    …                 …          …             …  …
2144            1.561126          4.049964     6.555306      7.535540  …
2145           18.767261          1.360667     2.904662      8.555256  …
2146            4.594670          9.886002     8.120025      5.769464  …
2147            8.674505          6.354282     1.263427      8.322874  …
2148            7.890703          6.570993     7.941404      9.878711  …

      FunctionalAssessment  MemoryComplaints  BehavioralProblems       ADL  \
0                 6.518877                 0                   0  1.725883
1                 7.118696                 0                   0  2.592424
2                 5.895077                 0                   0  7.119548
3                 8.965106                 0                   1  6.481226
4                 6.045039                 0                   0  0.014691
…                      …                 …                   …       …
2144              0.238667                 0                   0  4.492838
2145              8.687480                 0                   1  9.204952
2146              1.972137                 0                   0  5.036334
2147              5.173891                 0                   0  3.785399
2148              6.307543                 0                   1  8.327563

      Confusion  Disorientation  PersonalityChanges  \
0             0               0                   0
1             0               0                   0
```

```
2               0                1                 0
3               0                0                 0
4               0                0                 1
...           ...              ...               ...
2144            1                0                 0
2145            0                0                 0
2146            0                0                 0
2147            0                0                 0
2148            0                1                 0

      DifficultyCompletingTasks  Forgetfulness  Diagnosis
0                             1              0          0
1                             0              1          0
2                             1              0          0
3                             0              0          0
4                             1              0          0
...                         ...            ...        ...
2144                          0              0          1
2145                          0              0          1
2146                          0              0          1
2147                          0              1          1
2148                          0              1          0

[2149 rows x 33 columns]
```

[8]:

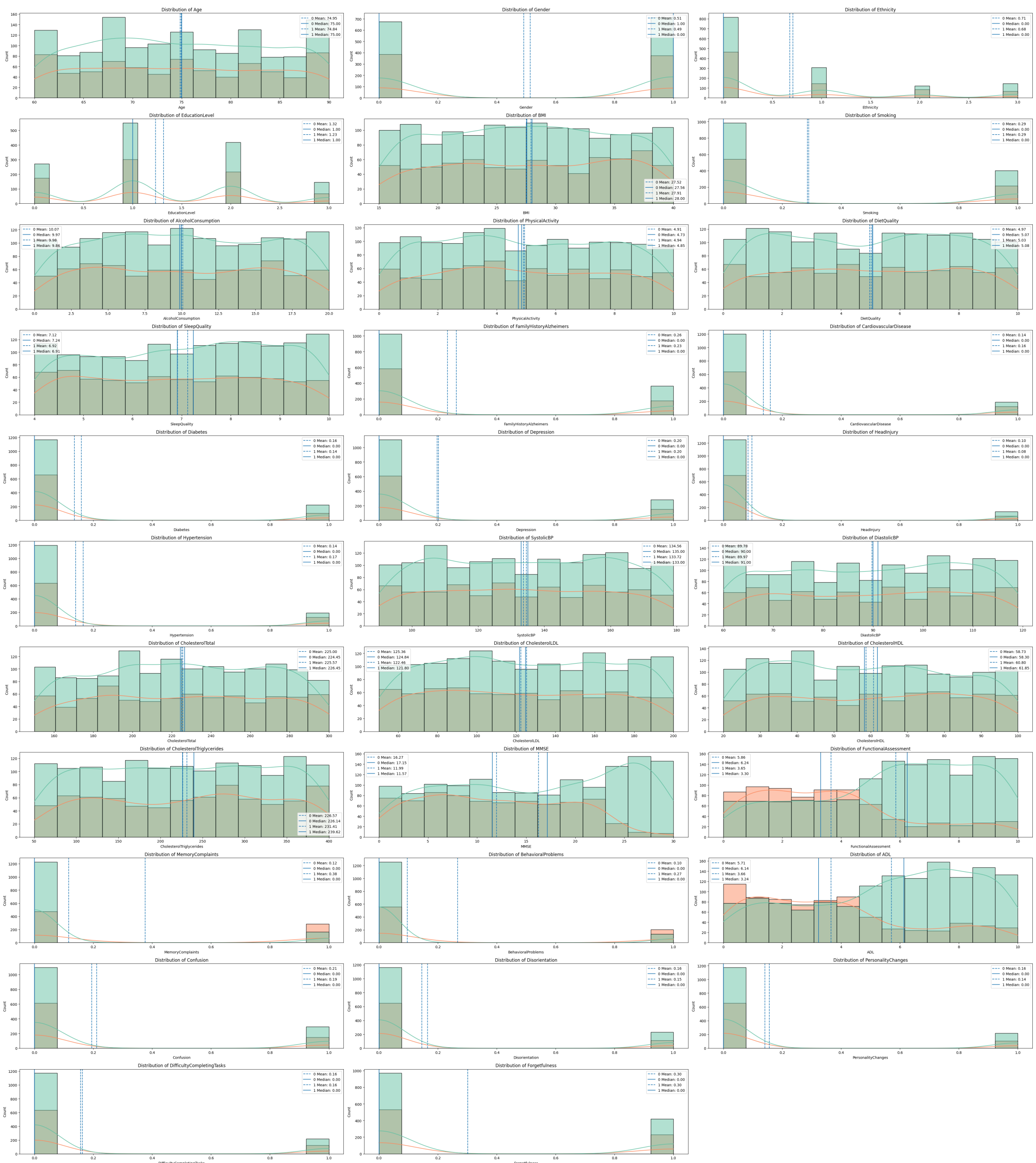
```python
# Test pour les outliers avec la méthode de l'écart interquartile (IQR)
def detect_outliers(df):
    outlier_indices = []
    for col in df.columns[:-1]:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        outlier_step = 1.5 * IQR
        outliers = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 +
 ↪outlier_step)].index
        outlier_indices.extend(outliers)
    outlier_indices = list(set(outlier_indices))
    return outlier_indices

outlier_indices = detect_outliers(df)
print(f"Nombre d'outliers détectés : {len(outlier_indices)}")
```

```
Nombre d'outliers détectés : 1895
```

[9]:

```python
# Visualisation de la normalité des données
a = len(df.columns[:-1])
b = 3
```

```python
rows = (a // b) + (1 if a % b != 0 else 0)

fig = plt.figure(figsize=(40, 45))

for idx, col in enumerate(df.columns[:-1], start=1):
    plt.subplot(rows, b, idx)

    # Tracé avec hue
    sns.histplot(data=df, x=col, kde=True, hue=df['Diagnosis'], alpha=0.5, palette="Set2")

    # Moyenne et médiane par catégorie de hue
    unique_hues = df['Diagnosis'].unique()  # Catégories uniques du hue
    for hue_value in unique_hues:
        subset = df[df['Diagnosis'] == hue_value]  # Filtre les données pour chaque catégorie
        mean_value = np.mean(subset[col])
        median_value = np.median(subset[col])

        # Ajout des lignes verticales pour chaque catégorie
        plt.axvline(x=mean_value, linestyle='--', label=f"{hue_value} Mean: {mean_value:.2f}")
        plt.axvline(x=median_value, linestyle='-', label=f"{hue_value} Median: {median_value:.2f}")

    plt.title(f'Distribution of {col}')
    plt.legend()

plt.tight_layout()
plt.show()
```



[10]:
```python
# Séparation des features et de la target
X = df.drop(columns=['Diagnosis'])
y = df['Diagnosis']
```

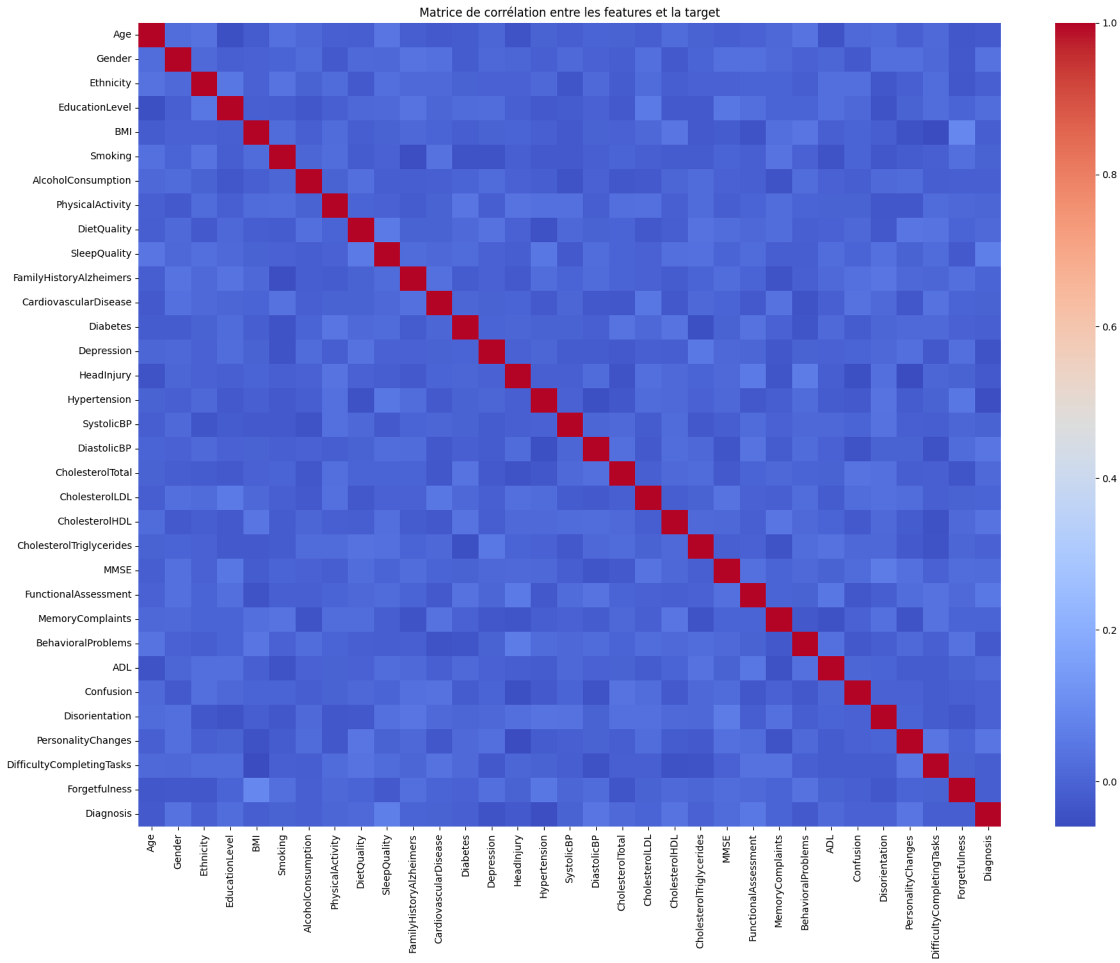
[11]:
```python
# Analyse de la distribution des classes
unique_classes, class_counts = np.unique(y, return_counts=True)
print("\n Distribution des classes ")
for cls, count in zip(unique_classes, class_counts):
    print(f"Classe: {cls} | Nombre: {count}")
```

```
 Distribution des classes
Classe: 0 | Nombre: 1389
Classe: 1 | Nombre: 760
```

[12]:
```python
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

[13]:
```python
# Standardisation des variables

# Création du Scaler
scaler = StandardScaler()

# Standardisation
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convertir en DataFrame
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

[14]:
```python
# Analyse de corrélation

# Matrice de corrélation
corr_matrix = X_train_scaled_df.join(y_train).corr()

# Visualisation
plt.figure(figsize=(20, 15))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm')
plt.title("Matrice de corrélation entre les features et la target")
plt.show()

# Features importantes
correlated_features = corr_matrix[abs(corr_matrix['Diagnosis']) > 0.1].index.
 ↪tolist()
correlated_features.remove('Diagnosis')
print("\nFeatures corrélées avec la target:")
print(correlated_features)
```

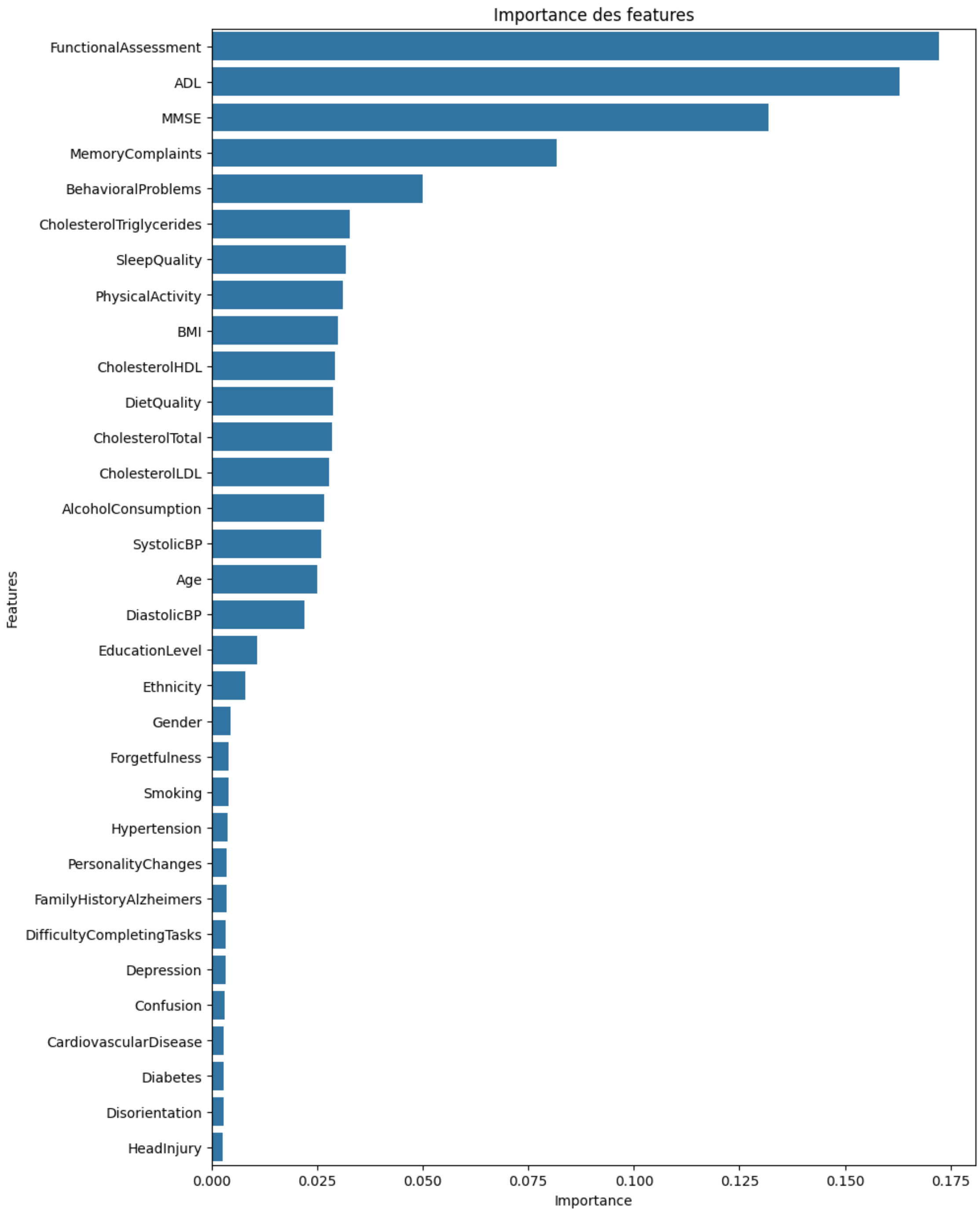



```
Features corrélées avec la target:
[]
```

[15]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest, f_classif
```

[16]:
```python
# Sélection de features importantes

# Feature Importance via RF
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train_scaled_df, y_train)

feature_importances = pd.Series(rf.feature_importances_, index=X_train.columns)
feature_importances = feature_importances.sort_values(ascending=False)

# Visualisation
```

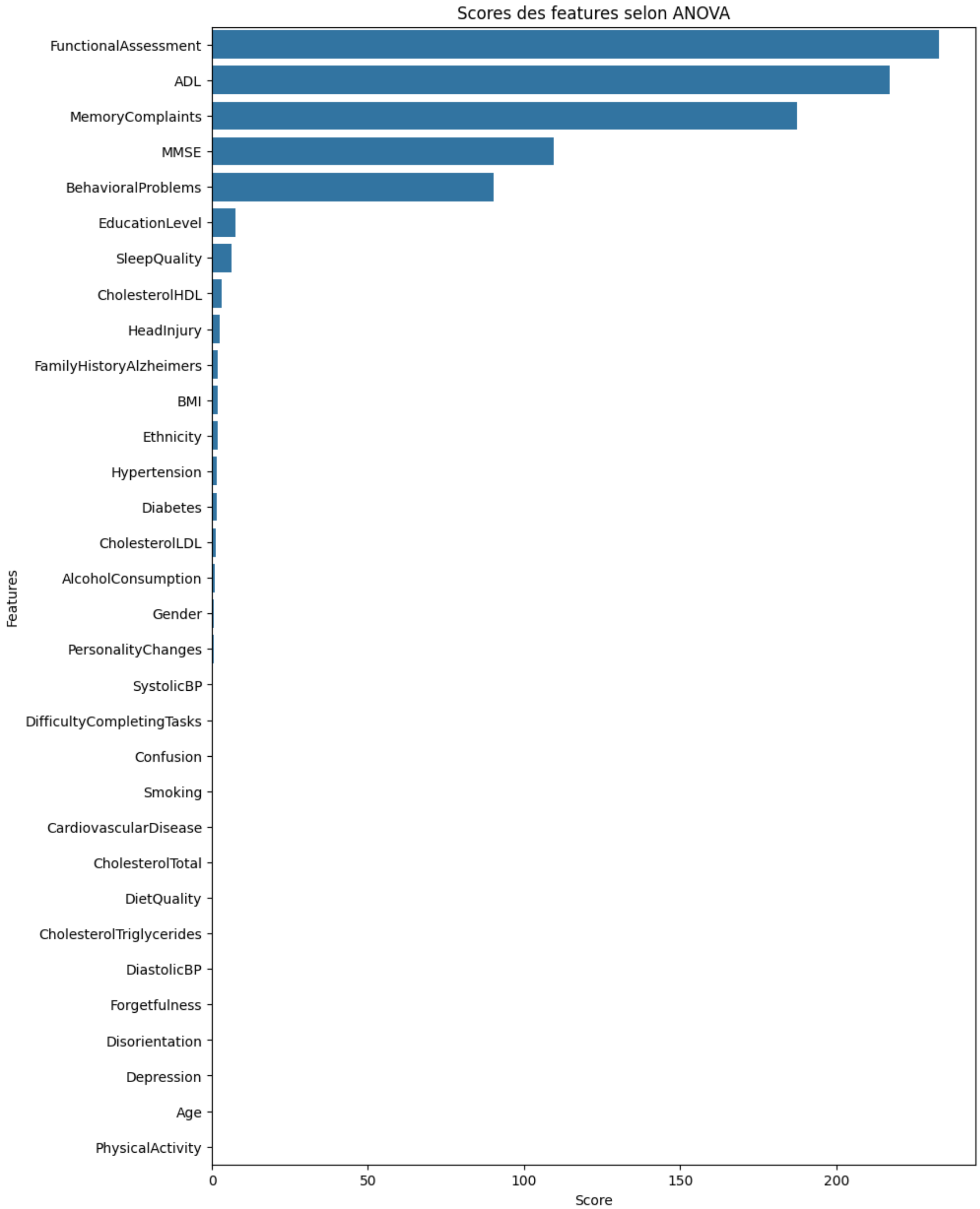
```python
plt.figure(figsize=(10, 15))
sns.barplot(x=feature_importances.values, y=feature_importances.index)
plt.title("Importance des features")
plt.xlabel("Importance")
plt.ylabel("Features")
plt.show()
```



```
[17]: # Sélection des features importantes
selected_features_rf = feature_importances[feature_importances > 0.02].index.
 ↪tolist()
print("\nFeatures sélectionnées selon l'importance dans le modèle:")
print(selected_features_rf)
```

```
Features sélectionnées selon l'importance dans le modèle:
['FunctionalAssessment', 'ADL', 'MMSE', 'MemoryComplaints',
'BehavioralProblems', 'CholesterolTriglycerides', 'SleepQuality',
'PhysicalActivity', 'BMI', 'CholesterolHDL', 'DietQuality', 'CholesterolTotal',
'CholesterolLDL', 'AlcoholConsumption', 'SystolicBP', 'Age', 'DiastolicBP']
```

```
[18]: # Utilisation d'ANOVA pour évaluer la pertinence des features
selector = SelectKBest(score_func=f_classif, k='all')
selector.fit(X_train_scaled_df, y_train)

# Scores des features
feature_scores = pd.Series(selector.scores_, index=X_train.columns)
feature_scores = feature_scores.sort_values(ascending=False)

# Visualisation
plt.figure(figsize=(10, 15))
sns.barplot(x=feature_scores.values, y=feature_scores.index)
plt.title("Scores des features selon ANOVA")
plt.xlabel("Score")
plt.ylabel("Features")
plt.show()
```



[19]:
```
# Sélection des features importantes
selected_features_anova = feature_scores[feature_scores > 10].index.tolist()
print("\nFeatures sélectionnées:")
print(selected_features_anova)
```

```
Features sélectionnées:
['FunctionalAssessment', 'ADL', 'MemoryComplaints', 'MMSE',
'BehavioralProblems']
```

[20]:
```python
# Ensemble des features sélectionnées
selected_features = list(set(correlated_features + selected_features_rf +
    selected_features_anova))
print("\nFeatures finalement sélectionnées:")
print(selected_features)
```

```
Features finalement sélectionnées:
['Age', 'SystolicBP', 'MemoryComplaints', 'CholesterolTriglycerides',
'CholesterolHDL', 'FunctionalAssessment', 'SleepQuality', 'CholesterolTotal',
'CholesterolLDL', 'DietQuality', 'AlcoholConsumption', 'BMI',
'BehavioralProblems', 'PhysicalActivity', 'ADL', 'DiastolicBP', 'MMSE']
```

[21]:
```python
# Application de la sélection de features sur les ensembles d'entraînement et
    de test
X_train_selected = X_train_scaled_df[selected_features]
X_test_selected = X_test_scaled_df[selected_features]
```

[22]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, classification_report, confusion_matrix
```

[23]:
```python
# Ignorer les warnings
import warnings
warnings.filterwarnings('ignore')
```

[24]:
```python
# Développement des modèles

# Modèles
models = {
    'Logistic Regression': LogisticRegression(max_iter=10000),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(probability=True)}

results = {}
```

[25]:
```python
# Fonction pour évaluer un modèle

def evaluate_model(model_name, model, X_train, y_train, X_test, y_test):
    # Entraînement du modèle
    model.fit(X_train, y_train)
```

```python
    # Prédiction sur l'ensemble de test
    y_pred = model.predict(X_test)

    # Calcul des métriques
    accuracy = accuracy_score(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)

    # Stockage des résultats
    results[model_name] = {
        'model': model,
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'classification_report': classification_report(y_test, y_pred),
        'confusion_matrix': confusion_matrix(y_test, y_pred)
    }

    # Afficher les résultats
    print(f"\nRésultats pour {model_name}:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Précision: {precision:.4f}")
    print(f"Rappel: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")
    print("\nClassification Report:")
    print(results[model_name]['classification_report'])
    print("\nConfusion Matrix:")
    print(results[model_name]['confusion_matrix'])
```

[26]:
```python
# Évaluation de chaque modèle
for model_name, model in models.items():
    print(f"\nEntraînement et évaluation du {model_name}")
    evaluate_model(model_name, model, X_train_selected, y_train,␣
 ↪X_test_selected, y_test)
```

```
Entraînement et évaluation du Logistic Regression

Résultats pour Logistic Regression:
Accuracy: 0.8395
Précision: 0.8239
Rappel: 0.7267
F1-score: 0.7723
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.91      0.88       269
           1       0.82      0.73      0.77       161

    accuracy                           0.84       430
   macro avg       0.84      0.82      0.82       430
weighted avg       0.84      0.84      0.84       430


Confusion Matrix:
[[244  25]
 [ 44 117]]

Entraînement et évaluation du Random Forest

Résultats pour Random Forest:
Accuracy: 0.9302
Précision: 0.9226
Rappel: 0.8882
F1-score: 0.9051

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.94       269
           1       0.92      0.89      0.91       161

    accuracy                           0.93       430
   macro avg       0.93      0.92      0.92       430
weighted avg       0.93      0.93      0.93       430


Confusion Matrix:
[[257  12]
 [ 18 143]]

Entraînement et évaluation du SVM

Résultats pour SVM:
Accuracy: 0.8512
Précision: 0.8299
Rappel: 0.7578
F1-score: 0.7922

Classification Report:
              precision    recall  f1-score   support
```

```
              0       0.86      0.91      0.88       269
              1       0.83      0.76      0.79       161

       accuracy                           0.85       430
      macro avg       0.85      0.83      0.84       430
   weighted avg       0.85      0.85      0.85       430


Confusion Matrix:
[[244  25]
 [ 39 122]]
```

[27]:
```
# Optimisation des hyperparamètres

# Paramètres à optimiser pour chaque modèle
param_grids = {
    'Logistic Regression': {
        'C': [0.001, 0.01, 0.1, 1, 10, 100],
        'penalty': ['l2']
    },
    'Random Forest': {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    },
    'SVM': {
        'C': [0.1, 1, 10, 100],
        'kernel': ['linear', 'rbf'],
        'gamma': ['scale', 'auto', 0.1, 1, 10]
    }
}
```

[28]:
```
# Fonction pour optimiser les hyperparamètres

def optimize_hyperparameters(model_name, model, param_grid, X_train, y_train):
    print(f"\nOptimisation des hyperparamètres pour {model_name}")

    # Validation croisée via StratifiedKFold
    cv = StratifiedKFold(n_splits=5, shuffle=True)

    # GridSearchCV
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid,␣
 ↪scoring='f1', cv=cv, n_jobs=-1, verbose=1)

    # Entraînement sur les données d'entraînement
```

```python
    grid_search.fit(X_train, y_train)

    # Meilleurs paramètres et meilleur score
    best_params = grid_search.best_params_
    best_score = grid_search.best_score_

    # Entraînement du modèle avec les meilleurs paramètres
    best_model = grid_search.best_estimator_

    # Stockage des résultats
    results[model_name]['best_params'] = best_params
    results[model_name]['best_score'] = best_score
    results[model_name]['best_model'] = best_model

    # Affichage des résultats
    print(f"Meilleurs paramètres: {best_params}")
    print(f"Meilleur score (F1): {best_score:.4f}")

    return best_model
```

[29]:

```python
# Optimisation de  chaque modèle
for model_name, model in models.items():
    best_model = optimize_hyperparameters(model_name, model,␣
 ↪param_grids[model_name], X_train_selected, y_train)
    # Réévaluation du modèle optimisé
    print(f"\nRéévaluation du {model_name} optimisé")
    evaluate_model(f"{model_name} (optimisé)", best_model, X_train_selected,␣
 ↪y_train, X_test_selected, y_test)
```

```
Optimisation des hyperparamètres pour Logistic Regression
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Meilleurs paramètres: {'C': 10, 'penalty': 'l2'}
Meilleur score (F1): 0.7724

Réévaluation du Logistic Regression optimisé

Résultats pour Logistic Regression (optimisé):
Accuracy: 0.8395
Précision: 0.8239
Rappel: 0.7267
F1-score: 0.7723

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.91      0.88       269
```

```
           1       0.82      0.73      0.77       161

    accuracy                           0.84       430
   macro avg       0.84      0.82      0.82       430
weighted avg       0.84      0.84      0.84       430


Confusion Matrix:
[[244  25]
 [ 44 117]]

Optimisation des hyperparamètres pour Random Forest
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Meilleurs paramètres: {'max_depth': 10, 'min_samples_leaf': 2,
'min_samples_split': 10, 'n_estimators': 100}
Meilleur score (F1): 0.9343

Réévaluation du Random Forest optimisé

Résultats pour Random Forest (optimisé):
Accuracy: 0.9395
Précision: 0.9355
Rappel: 0.9006
F1-score: 0.9177

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       269
           1       0.94      0.90      0.92       161

    accuracy                           0.94       430
   macro avg       0.94      0.93      0.93       430
weighted avg       0.94      0.94      0.94       430


Confusion Matrix:
[[259  10]
 [ 16 145]]

Optimisation des hyperparamètres pour SVM
Fitting 5 folds for each of 40 candidates, totalling 200 fits
Meilleurs paramètres: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
Meilleur score (F1): 0.7750

Réévaluation du SVM optimisé

Résultats pour SVM (optimisé):
```

```
Accuracy: 0.8512
Précision: 0.8299
Rappel: 0.7578
F1-score: 0.7922

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.91      0.88       269
           1       0.83      0.76      0.79       161

    accuracy                           0.85       430
   macro avg       0.85      0.83      0.84       430
weighted avg       0.85      0.85      0.85       430


Confusion Matrix:
[[244  25]
 [ 39 122]]
```

[30]:

```
# Comparaison des modèles optimisés

# DataFrame pour comparer les performances
comparison_df = pd.DataFrame({
    'Modèle': [],
    'Accuracy': [],
    'Précision': [],
    'Rappel': [],
    'F1-score': []
})

new_rows = []

for model_name, result in results.items():
    if 'optimisé' in model_name:
        new_rows.append({
            'Modèle': model_name,
            'Accuracy': result['accuracy'],
            'Précision': result['precision'],
            'Rappel': result['recall'],
            'F1-score': result['f1']
        })

if new_rows:
    comparison_df = pd.concat([comparison_df, pd.DataFrame(new_rows)],␣
 ↪ignore_index=True)
```

[31]:
```python
# Tableau de comparaison
print("\nComparaison des modèles optimisés:")
print(comparison_df)
```

```
Comparaison des modèles optimisés:
                         Modèle  Accuracy  Précision    Rappel  F1-score
0  Logistic Regression (optimisé)  0.839535   0.823944  0.726708  0.772277
1        Random Forest (optimisé)  0.939535   0.935484  0.900621  0.917722
2                  SVM (optimisé)  0.851163   0.829932  0.757764  0.792208
```