Author: Andreas Schau - AS

Leitfaden für nachvollziehbare Schritte

1. Kurze Darstellung des Problembereichs / Aufriss des Themas

1.1 Inhaltlich

Kann mittels Ensemble-Learnerning eine Regression von Schlaflabor-Daten durchgeführt werden, welche mit über 80% Genauigkeit Vorhersagen zur Schlaf-Effizienz treffen kann?
Dies soll anhand des Sleep Efficiency Datasets durchgeführt werden, welches unter anderem Daten zum REM-Schlaf-Anteil, Schlaflänge und Alkohol-Konsum enthält.

1.2 Begründung desThemas

Darstellung der Relevanz des Themas?

In unserer technologisch entwickelten Welt haben viele Menschen einen sicheren Schlafplatz und sollten in der Lage sein gut zu schlafen und ausgeschlafen aufzustehen. Jedoch gibt es auch Faktoren die diese Fortschritte negieren.
Damit Menschen mit Schlafproblemen einen Ansatz haben, an welchen Ihrer Umgebungs-Bedingungen sie etwas verändern sollten, wurden Daten zur Bestimmung der Schlafqualität und Effizienz von über 500 Probanden aufgenommen, welche mithilfe maschinellen Lernens untersucht werden können, um aufzuzeigen, welche Variablen den größten Einfluss auf guten Schlaf haben.

Darstellung eines persönlichen Erkenntnisinteresses.

Ich habe, noch bis vor einem Jahr, sehr mit Schlafproblemen zu kämpfen gehabt. Mir gelang es glücklicherweise irgendwann meine Probleme auf meine Ernährung und unregelmäßige Schlafenszeiten zurück zu führen. Das Völlegefühl am Abend, sowie das Aufwachen in der Nacht um auf Toilette zu gehen haben die Erholsamkeit meines Schlafes stark beschränkt.

2. Nachvollziehbare Schritte

2.1 Der Stand der Forschung / Auswertung der vorhandenen Literatur / Tutorials ...

Untersuchungen im Schlaflabor gibt es schon seit Ende der 1960er Jahre. Seit dem wurde in dieser wissenschaftlichen Disziplin bereits einiges an Erkenntnissen für guten Schlaf gewonnen und warum es einigen Menschen schwer fällt diesen zu bekommen.
Bisher war es jedoch Wissenschaftlern vorbehalten die Schlafbedingungen mit der Schlaf-Effizienz in Relation zu setzen.
Mithilfe eines Regressionsmodells könnte man die Zusammenhänge, die man aus den Daten gewinnt jedem zur Verfügung stellen, sodass diese ihre eigene Situation angeben und ausprobieren können welche Änderungen ihnen zu besserem Schlaf verhelfen sollten.

Erklärung der Vorgehensweise im Code

```
# %%
# Imports
import json

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import shap
import xgboost
from bayes_opt import BayesianOptimization
from sklearn.discriminant_analysis import StandardScaler
from sklearn.ensemble import (AdaBoostRegressor, BaggingRegressor,
                              GradientBoostingRegressor, RandomForestRegressor,
                              StackingRegressor, VotingRegressor)
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import OrdinalEncoder
from sklearn.tree import DecisionTreeRegressor

# NOTE: referencing here so it does not get cleaned up by auto import cleaning tools
enable_iterative_imputer

```

Zunächst werden alle benötigten Bibliotheken importiert und wenn nötig mit einem Alias versehen. Im Detail sind es:

- json – Zum lesbaren ausgeben von Dictionaries (Datentyp)
- matplotlib – Wird für das Plotten von Histogrammen verwendet
- numpy – Wird für die Datenaufbereitung benötigt
- pandas – Zum laden der CSV-Daten und handhaben in einem DataFrame
- seaborn – Zur Darstellung einer Hearmap der Korrelation der Daten
- shap – Zur Veranschaulichung, der Merkmals-Wichtigkeiten
- xgboost – Zur Untersuchung der Performance als Regressor
- bayes_opt – Für die Hyperparameter-Optimierung des gewählten Regressions-Modells
- sklearn – Stellt alle benötigten Daten-Aufbereitungs und Validierungs Algorithmen, sowie Regressions-Modelle zur Verfügung

```
27  # %%
28  # Data preparation
29
30
31  def load_data():
32      return pd.read_csv('Sleep_Efficiency.csv')
33
34
35  def remove_outliers(data_frame):
36      # Calculate the IQR for each column in the dataframe
37      Q1 = data_frame.quantile(0.25, numeric_only=True)
38
39      Q3 = data_frame.quantile(0.75, numeric_only=True)
40
41      IQR = Q3 - Q1
42
43      # Print the shape of the dataframe before removing the outliers
44      print("The shape of the dataframe before removing the outliers is " +
45            str(data_frame.shape))
46
47      # Remove the outliers from the dataframe
48      data_frame = data_frame[~((data_frame < (Q1 - 1.5 * IQR)) |
49                                (data_frame > (Q3 + 1.5 * IQR))).any(axis=1)]
50
51      # Print the shape of the dataframe after removing the outliers
52      print("The shape of the dataframe after removing the outliers is " +
53            str(data_frame.shape))
54
55      return data_frame
56
```

Im ersten Block werden Funktionen zum laden und aufbereiten der Daten implementiert.

Die Funktion `load_data()` nutzt die pandas-Bibliothek um das “Sleep_Efficiency.csv”-Dataset in einen DataFrame zu laden.

Die Funktion `remove_outliers()` ist eine typische IQR-Outlier-Removal Implementation, die Werte unterhalb des 25% und oberhalb des 75% Quantils aus dem Datensatz entfernt. Sie wurde während der Untersuchungspahse angewendet und hat dazu geführt, dass die Vorhersagegenauigkeit stark verschlechtert wurde. Daher wurde sie für die weiteren Untersuchungen auskommentiert und nicht weiter verwendet.

```python
def get_null_entries(data_frame):
    return data_frame.isnull().sum().to_frame('null_count')


def split_input_output(data_frame):
    y = data_frame['Sleep efficiency']

    # drop id column as it has no learnable information in this context
    # drop sleep efficiency as it is the target value
    X = data_frame.drop(columns=['ID',
                                 'Sleep efficiency'],
                        axis=1)

    return (X, y)


def ordinal_encode(X, ordinal_encoder):
    return pd.DataFrame(ordinal_encoder.fit_transform(X),
                        columns=X.columns)


def impute_null_entries(data_frame):
    imputer_mean = IterativeImputer(missing_values=np.nan,
                                    initial_strategy='mean',
                                    random_state=42)
    return pd.DataFrame(imputer_mean.fit_transform(data_frame),
                        columns=data_frame.columns)

```

Die Funktion `get_null_entries()` nimmt den `data_frame` der den Schlaf-Effizienz Datensatz enthält und zählt alle `nan`-Werte in den einzelnen Spalten des Datensatzes zusammen und erstellt einen neuen DataFrame namens “null_count” den die Funktion auch zurück gibt, um später ausgeben zu können, wie viele Null-Werte/Missing-Values es im Datensatz gibt und wie sich diese auf die verschiedenen Spalten verteilt sind. (Siehe Ausführung weiter unten.)

Die Funktion `split_input_output()` bekommt den Datensatz und teilt ihn in die Ziel/Output-Variable auf indem sie die “Sleep efficiency” Spalte der Variable `y` zuweist und in die Input-Variable indem sie der `X` Variable den `data_frame` zuweist, aus dem die Spalten “ID” und “Sleep efficiency” entfernt wurden.
Diese beiden Spalten wurden entfernt, weil die ID keinen Mehrwert für die Genauigkeit des Modells beinhaltet und die Spalte “Sleep efficiency” die Zielvariable ist und diese nicht in den Eingangs-Variablen enhalten sein darf. Sie gibt diese beiden Variablen dann als Tupel (X, y) zurück.

Die Funktion `ordinal_encode()` bekommt die Eingangswerte und einen `OrdinalEncoder`, um den Datensatz, der noch Zeichenketten enthällt, in einen rein numerischen Datensatz umzuwandeln und gibt diesen zurück. Die Spaltennamen werden dabei mit übergeben, damit diese in späteren Diagrammen und anderen Ausgaben noch zur Verfügung stehen und nicht nur numerische Indicies sind.

Zuletzt die Funktion `impute_null_entries()`, welche den Datensatz entgegen nimmt und mit einem iterativen Imputer die Null-Werte via der initialen Imputations-Strategie “Mean” (Durchschnitt), mit realistischen Werten auffüllt. Dadurch wird vermieden, dass Null-Werte die Genauigkeit der Regression verringern.
Die Imputierten Werte werden dann in einem neuen DataFrame verpackt, bei dem die Spaltennamen mit angegeben werden. Dieser neue DataFrame wird von der Funktion zurück gegeben.

```
87  def scale_inputs(X, scaler):
88      # Compute the mean and standard deviation of the training set then transform it
89      return pd.DataFrame(scaler.fit_transform(X),
90                          columns=X.columns)
91
92
93  def clean_data(data_frame, ordinal_encoder, scaler):
94      print(f'>> Duplicate entries count:\n\n{data_frame.duplicated().sum()}\n')
95
96      (X, y) = split_input_output(data_frame)
97
98      print(
99          f'>> Null entries of input columns:\n\n{get_null_entries(X)}\n')
100
101     X = ordinal_encode(X, ordinal_encoder)
102     X = impute_null_entries(X)
103     print(
104         f'>> Null entries of input columns after label encoding and imputation:\n\n{get_null_entries(X)}\n')
105
106     X = scale_inputs(X, scaler)
107
108     return (X, y)
```

Es folgen weitere Funktionen, darunter `scale_inputs()`, welche die Eingangsdaten und einen Skalierer entgegen nimmt und den Eingangsdatensatz standard-skaliert zurück gibt.
Die Funktion `clean_data` bekommt den Gesamt-Datensatz, einen Ordinal-Encoder und einen Skalierer und orchestriert das bereinigen der Daten, indem sie die zuvor beschriebenen Funktionen mit den entsprechenden Werten aufruft.
Zuerst gibt sie die doppelten Einträge im Datensatz aus, dann teilt sie den Datensatz in Eingabe und Zielwerte auf und legt diese auf die Variablen `X` und `y`. Dann gibt sie die Summe aller Null-Einträge pro Spalte aus, wendet den Ordinal-Encoder und den Imputer auf die Eingangsdaten an und gibt wiederum aus, wie viele Null-Daten es dannach noch im Datensatz gibt.
Zuletzt werden die Eingangsdaten dann noch skaliert und das Tupel `(X, y)` zurück gegeben.

```
111  def histograms(data_frame):
112      print(">> Histograms of dataset's columns:")
113      data_frame.hist(figsize=(10, 10))
114      plt.show()
115
116
117  def heatmap(data_frame):
118      print(">> Heatmap of dataset's columns correlation:")
119      plt.figure(figsize=(10, 10))
120      sns.heatmap(data=data_frame.corr(),
121                  vmin=-1,
122                  vmax=1,
123                  annot=True,
124                  cmap='coolwarm')
125
```

Die nächsten beiden Funktionen `histograms()` und `heatmap()` nehmen den Datensatz entgegen und erstellen die jeweiligen Diagramme, also ein Histogramm für die Verteilung der Werte jeder Spalte und eine Korrelations-Heatmap, auf der man die Korrelation der Spalten untereinander visuell erkennen kann.

```
127  df = load_data()
128
129  # NOTE: Commented out because it made the predictions significantly worse
130  # df = remove_outliers(df)
131
132  ordinal_encoder = OrdinalEncoder()
133  scaler_linear = StandardScaler()
134  (X, y) = clean_data(df, ordinal_encoder, scaler_linear)
135
136  histograms(df)
137
138  X_and_y = X
139  X_and_y['Sleep efficiency'] = y
140  heatmap(pd.DataFrame.from_dict(X_and_y))
141
142  (X_train, X_test,
143   y_train, y_test) = train_test_split(X,
144                                       y,
145                                       train_size=0.2,
146                                       shuffle=True,
147                                       random_state=4)
148
```

In diesem Abschnitt werden via den vorher erstellten Funktionen nun mit `load_data()` die Daten in den DataFrame `df` geladen, dann wird ein OrdinalEncoder und eine StandardScaler erstellt und der Funktion `clean_data()` zusammen mit dem DataFrame als Parameter mitgegeben.
Diese gibt dann die aufbereiteten Eingangs- und Ziel-Daten als Tupel `(X, y)` zurück.
Darauf hin wird der Datensatz als Histogram und Heatmap visualisiert. Für die Heatmap wird dafür ein

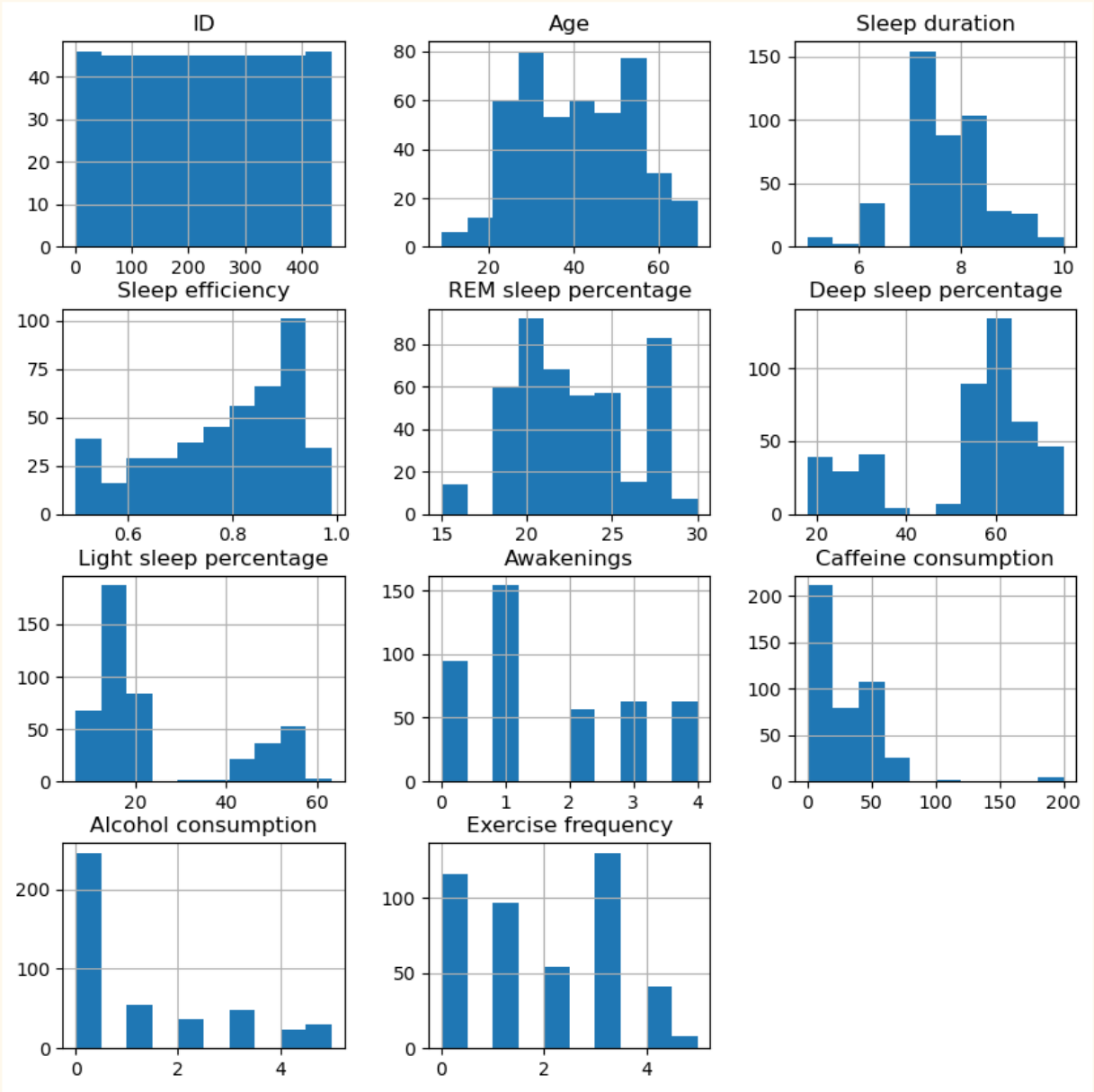
DataFrame aus den aufbereiteten Eingangs- und den abgespalteten Ziel-Daten erstellt und der `heatmap()` Funktion übergeben.
Zuetzt werden die Eingangs- und Ziel-Daten noch via der `train_test_split()` Funktion von Scikit-Learn in einen Trainings- und Test-Datensatz aufgeteilt.

Dieser Abschnitt erzeugt dadurch diese Ausgaben:

```
✓ # Data preparation ···
>> Duplicate entries count:

0

>> Null entries of input columns:

                        null_count
Age                              0
Gender                           0
Bedtime                          0
Wakeup time                      0
Sleep duration                   0
REM sleep percentage             0
Deep sleep percentage            0
Light sleep percentage           0
Awakenings                      20
Caffeine consumption            25
Alcohol consumption             14
Smoking status                   0
Exercise frequency               6

>> Null entries of input columns after label encoding and imputation:

                        null_count
Age                              0
Gender                           0
Bedtime                          0
Wakeup time                      0
Sleep duration                   0
REM sleep percentage             0
Deep sleep percentage            0
Light sleep percentage           0
Awakenings                       0
Caffeine consumption             0
Alcohol consumption              0
Smoking status                   0
Exercise frequency               0
```

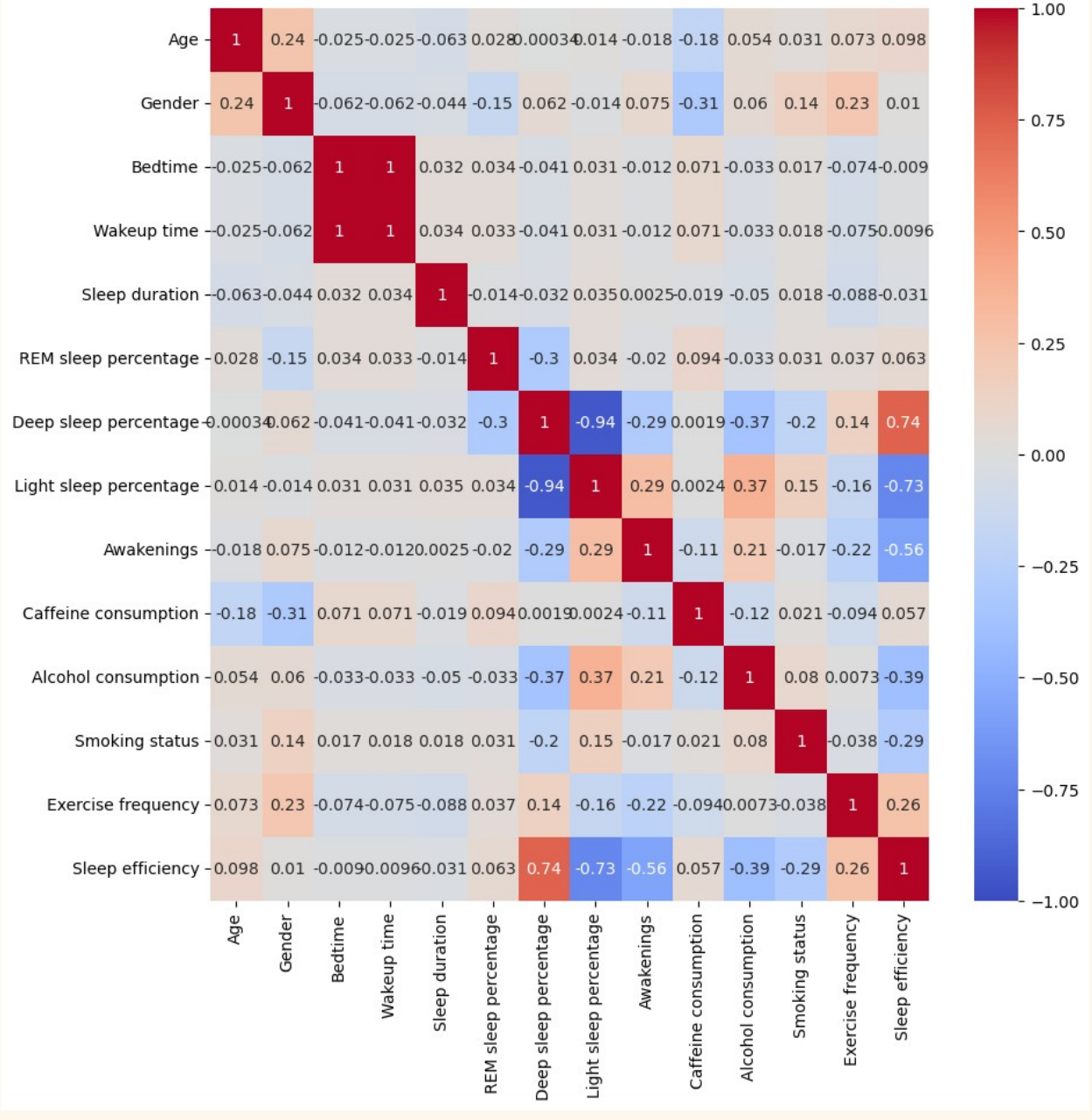
Ausgaben der `clean_data()`-Funktion.

```
>> Histograms of input columns:
```



Ausgabe der `histograms()`-Funktion.

```
>> Heatmap of input columns correlation:
```



Ausgabe der `heatmap()`-Funktion. Durch diese Darstellung kann man gut sehen, welche Parameter/Spalten einen Zusammenhang untereinander aufweisen. In der Zeile “Sleep efficiency” kann man auch gut ablesen, welche Faktoren für eine hohe Schlaf-Effizienz einen positiven (rot) sowie negativen (blau) Effekt haben. Schon hier kann man also erkennen, dass ein hoher Tiefschlaf-Anteil und eine häufige sportliche Betätigung sich günstig auf die Schlafeffizienz auswirken und dass ein hocher Leichtschlaf-Anteil, häufiges Aufwachen hoher Alkohol- und Zigaretten-Konsum sich negativ darauf auswirken.

```
150  # %%
151  # K-fold cross validation of various regression ensembles with full dataset
152
153  regressors = [
154      ('AdaBoostRegressorRFR', AdaBoostRegressor(estimator=RandomForestRegressor())),
155      ('GradientBoostingRegressor', GradientBoostingRegressor()),
156      ('RandomForestRegressor', RandomForestRegressor()),
157      ('XGBRegressor', xgboost.XGBRegressor()),
158      ('AdaBoostRegressorDTR', AdaBoostRegressor(estimator=DecisionTreeRegressor())),
159      ('Ridge', RidgeCV()),
160      ('KNeighborsRegressor', KNeighborsRegressor()),
161      ('DecisionTreeRegressor', DecisionTreeRegressor()),
162      ('StackingRegressor', StackingRegressor(
163          estimators=[('AdaBoostRegressorRFR', AdaBoostRegressor(estimator=RandomForestRegressor())),
164                      ('XGBRegressor', xgboost.XGBRegressor())],
165          final_estimator=VotingRegressor(
166              estimators=[
167                  ('rf', RandomForestRegressor()),
168                  ('gbrt', GradientBoostingRegressor())]))),
169      ('VotingRegressor', VotingRegressor(
170          estimators=[
171              ('abr-rfr', AdaBoostRegressor(estimator=RandomForestRegressor())),
172              ('gbr', GradientBoostingRegressor()),
173              ('rfr', RandomForestRegressor()),
174              ('xgb', xgboost.XGBRegressor()),
175              ('abr-dtr', AdaBoostRegressor(estimator=DecisionTreeRegressor()))])),
176      ('BaggingRegressor', BaggingRegressor(
177          estimator=VotingRegressor(
178              estimators=[
179                  ('abr-rfr', AdaBoostRegressor(estimator=RandomForestRegressor())),
180                  ('gbr', GradientBoostingRegressor()),
181                  ('rfr', RandomForestRegressor()),
182                  ('xgb', xgboost.XGBRegressor()),
183                  ('abr-dtr', AdaBoostRegressor(estimator=DecisionTreeRegressor()))])))),
184  ]
```

In diesem Abschnitt wird ein Array mit vielen verschiedenen Name-Regressor-Tupeln erstellt. Der Großteil ist dabei nur ein Regressor, es gibt aber auch die 3 verschachtelten größeren Regressor-Modelle “StackingRegressor”, der mehrere kleinere Regressoren via Stacking und Voting kombiniert, “VotingRegressor”, der die besten kleineren Regressoren via Voting kombiniert und “BaggingRegressor”, der Bagging ebenfalls mit Voting der besten kleineren Regressoren in der finalen Stufe miteinander kombiniert.

```
186  for name, regressor in regressors:
187      score = cross_val_score(regressor,
188                              X,
189                              y,
190                              cv=5,
191                              scoring='r2',
192                              n_jobs=-1)
193      print(f'The mean R-squared score of {name} is: {score.mean()}')
194      print(
195          f'>>  The train/test score of {name} is: {regressor.fit(X_train, y_train).score(X_test, y_test)}\n')
196
197
```

In dem darauf folgenden Abschnitt werden diese Regressoren nun alle via Cross-Validation auf dem gesamten Datensatz und separat auf dem Trainings- und Test-Datensatzt angewendet, um einen Einblick

auf ihre generelle Performance, sowie ihre Verallgemeinerungs-Fähigkeit zu ermöglichen.

```
✓ # K-fold cross validation of various regression ensembles with full dataset ···

The mean R-squared score of AdaBoostRegressorRFR is: 0.859359810831344
>>  The train/test score of AdaBoostRegressorRFR is: 0.8353778300066832

The mean R-squared score of GradientBoostingRegressor is: 0.859886952785774
>>  The train/test score of GradientBoostingRegressor is: 0.8075396200064899

The mean R-squared score of RandomForestRegressor is: 0.8552902291815168
>>  The train/test score of RandomForestRegressor is: 0.83520107077613

The mean R-squared score of XGBRegressor is: 0.84223532615663
>>  The train/test score of XGBRegressor is: 0.6849645357627117

The mean R-squared score of AdaBoostRegressorDTR is: 0.8426245694478199
>>  The train/test score of AdaBoostRegressorDTR is: 0.806983245730867

The mean R-squared score of Ridge is: 0.8021702868719608
>>  The train/test score of Ridge is: 0.7853664926593082

The mean R-squared score of KNeighborsRegressor is: 0.7772780547563373
>>  The train/test score of KNeighborsRegressor is: 0.713906961217637

The mean R-squared score of DecisionTreeRegressor is: 0.7533645827595444
>>  The train/test score of DecisionTreeRegressor is: 0.6808178473793026

The mean R-squared score of StackingRegressor is: 0.832352004185792
>>  The train/test score of StackingRegressor is: 0.7493786779397669

The mean R-squared score of VotingRegressor is: 0.8598068493747212
>>  The train/test score of VotingRegressor is: 0.8242786518485563

The mean R-squared score of BaggingRegressor is: 0.863496758061378
>>  The train/test score of BaggingRegressor is: 0.8358826557555437
```

Das ist die Ausgabe der Bewertung der verschiedenen Regressor-Modelle. Besonders die Modelle “AdaBoostRegressorR(andom)F(orest)R(egressor)”, “GradientBosstingRegressor” und der “BaggingRegressor” stechen durch ihre hohe Genauigkeit hervor.

Als Schlusslichter fallen der “DecisionTreeRegressor”, sowie der “XGB(oost)Regressor” ins Auge. Diese performen auf dem Gesamt-Datensatz gut, sind jedoch beim Test-Datensatz auffällig schlechter. Das könnte bedeuten, dass diese die Zusammenhänge in den Daten weniger verallgemeinert gelernt haben und an overfitting, sozusagen `auswendiglernen`, leiden.

Man beachte jedoch, dass all diese Regressoren hier nur mit ihren Standard-Werten untersucht wurden und es möglich wäre, dass optimierte Parameter, zu besseren Ergebnissen des jeweiligen Regressores führen

könnten.

```
198  # %%
199  # Run Bayesian Optimization
200
201  def optimizer(max_depth, max_features, learning_rate, n_estimators, subsample):
202      params_gbm = dict()
203      params_gbm['max_depth'] = round(max_depth)
204      params_gbm['max_features'] = max_features
205      params_gbm['learning_rate'] = learning_rate
206      params_gbm['n_estimators'] = round(n_estimators)
207      params_gbm['subsample'] = subsample
208
209      score = cross_val_score(
210          GradientBoostingRegressor(random_state=123, **params_gbm),
211          X,
212          y,
213          cv=5,
214          scoring='r2',
215          n_jobs=-1).mean()
216
217      return score
218
```

Nun da ein gutes Modell gefunden wurde, kann man noch versuchen, es via Hyper-Parameter-Optimierung um ein paar Prozent besser zu machen.
Dafür wurde die Funktion `optimizer()` implementiert, welche einige Parameter für den ausgewählten “GradientBosstingRegressor” entgegennimmt und in ein Dictionary überträgt, welches sie dem Regressor als Parameter übergibt. Der doppelte Stern sorgt dabei dafür, dass dieses Dictionary als Key-Word-Arguments ausgepackt an die Funktion übergeben wird.
Dann wird der Regressor via der `cross_val_score` Funktion am gesammten Datensatz ausprobiert und seine Bewertung via R-Square Metrik von der Funktion zurückgegeben.

```
220  params_dict = {
221      'max_depth': (1, 20),
222      'max_features': (0.5, 1),
223      'learning_rate': (0.001, 1),
224      'n_estimators': (20, 550),
225      'subsample': (0.7, 1)
226  }
227
228  optimization = BayesianOptimization(optimizer,
229                                      params_dict,
230                                      random_state=111)
231  optimization.maximize(init_points=50,
232                        n_iter=25)
233
234  params_optimized = optimization.max['params']
235  params_optimized['max_depth'] = round(params_optimized['max_depth'])
236  params_optimized['n_estimators'] = round(params_optimized['n_estimators'])
237  params_optimized['score'] = optimization.max['target']
238
239  print(
240      f'>> Best parameters for GradientBoostingRegressor:\n\n{json.dumps(params_optimized, indent=2)}')
241
```

Dann folgt die Definition eins Parameter-Dictionaries, in dem die Wertebereiche der einzelnen Parameter für die Optimierung des Regressors definiert werden.
Sowie wird ein `BayesianOptimization()` Optimierer mit der `optimizer`-Funktion und dem Wertebereich Parameter-Dictionary erstellt und in der nächsten Zeile via dem `maximize()` Funktionsaufruf des Optimierer-Objektes gestartet.
Je größer man die Werte für diese Maximierung auswählt, deste mehr Variationen können ausprobiert werden und desto höher ist die Chance für eine ganz besonders passende Hyperparameter-Optimierung.
Dannach wird die beste Kombination, die bei der Optimierung gefunden wurde ausgegeben.
Damit dies möglichst lesbar ist, wurde hier auf das “json” Modul und dessen Formatierung zurück gegriffen.

```
✓ # Run Bayesian Optimization ···
|   iter    |  target   | learni... | max_depth | max_fe... | n_esti... | subsample |
-------------------------------------------------------------------------------------
| 1         | 0.7828    | 0.6126    | 4.212     | 0.718     | 427.7     | 0.7886    |
| 2         | 0.8248    | 0.15      | 1.427     | 0.7101    | 146.5     | 0.8013    |
| 3         | 0.6635    | 0.9907    | 5.517     | 0.5406    | 374.9     | 0.8864    |
| 4         | 0.8419    | 0.275     | 9.858     | 0.5592    | 59.2      | 0.9702    |
| 5         | 0.7335    | 0.7942    | 16.97     | 0.9076    | 545.2     | 0.8732    |
| 6         | 0.6287    | 0.814     | 9.005     | 0.5137    | 260.7     | 0.7316    |
| 7         | 0.7629    | 0.8174    | 14.26     | 0.7826    | 165.3     | 0.9995    |
```

[...]

```
| 70        | 0.3801    | 0.001     | 8.619     | 1.0       | 293.7     | 0.7       |
| 71        | 0.8327    | 0.1649    | 6.682     | 0.8206    | 535.7     | 0.9973    |
| 72        | 0.8035    | 0.5048    | 4.226     | 0.5537    | 116.1     | 0.7419    |
| 73        | 0.796     | 0.4084    | 18.62     | 0.8508    | 478.3     | 0.854     |
| 74        | 0.7572    | 0.7975    | 10.37     | 0.9822    | 290.1     | 0.9811    |
| 75        | 0.7751    | 0.5447    | 19.31     | 0.6449    | 434.4     | 0.7576    |
=================================================================================
>> Best parameters for GradientBoostingRegressor:

{
  "learning_rate": 0.05064189475150359,
  "max_depth": 4,
  "max_features": 0.5232935914021011,
  "n_estimators": 116,
  "subsample": 0.9292360729264322,
  "score": 0.864996070750481
}
```

Die durch die Hyperparameter-Optimierung erzeugte Ausgabe.
Hier konnte durch die Optimierung der “GradientBosstingRegressor” von 85.99% Genauigkeit noch auf 86.5% verbessert werden. Immerhin eine Verbesserung um ein halbes Prozent, wodurch es damit sogar den oben besten “BaggingRegressor” um ein Müh übertrifft.
Der Grund, warum hier der GradientBosstingRegressor zur Hyperparameter-Optimierung verwendet wurde und nicht der BagginRegressor ist, dass beide schon ähnlich gut performt haben, dafür der GradientBosstingRegressor aber viel weniger Rechenleistung benötigt hat. Dieser war von den einfacheren der Beste und Resourcenschonendste.

```
242  # %%
243  # Initialize gradient boosting regressor model with best found parameters
244
245  model_best_gbr = GradientBoostingRegressor(random_state=123,
246                                             learning_rate=0.016224867884746044,
247                                             n_estimators=264,
248                                             subsample=0.989757891175945,
249                                             max_depth=4,
250                                             max_features=0.9135540990898587)
251
252  score_best = model_best_gbr.fit(X_train, y_train).score(X_test, y_test)
253  print(
254      f'>> Best gradient boosting regressor r2 (R squared) score on train-test-split dataset:\n{score_best}\n')
255
```

In diesem Abschnitt wird nun ein GradientBosstingRegressor mit den in einem vorherigen Lauf gefundenen besten Parametern erstellt und berechnet, wie gut dieser auf dem in Trainings- und Test-Daten aufgeteilten Datensatz performt. Dadurch kann man erkennen, ob dieser Regressor die Trainingsdaten gut verallgemeinert gelernt hat, indem er auf den Testdaten immernoch eine hohe Genaigkeit erziehlen kann.

```
✓ # Initialize gradient boosting regressor model with best found parameters ···

>> Best gradient boosting regressor r2 (R squared) score on train-test-split dataset:
0.819173742656195
```

Mit 82% Genaigkeit ist das der Fall.

```
# %%
# Predict single subjects sleep-efficiency

predict_input_high_caffeine = pd.DataFrame.from_dict({
    'Age': [34],
    'Gender': ['Male'],
    'Bedtime': ['2021-02-06 23:00:00'],
    'Wakeup time': ['2021-02-07 08:00:00'],
    'Sleep duration': [9.0],
    'REM sleep percentage': [24],
    'Deep sleep percentage': [67],
    'Light sleep percentage': [52],
    'Awakenings': [1.0],
    'Caffeine consumption': [200.0],
    'Alcohol consumption': [0.0],
    'Smoking status': ['No'],
    'Exercise frequency': [4.0]
})

predict_input_low_caffeine = pd.DataFrame.from_dict({
    'Age': [34],
    'Gender': ['Male'],
    'Bedtime': ['2021-02-06 23:00:00'],
    'Wakeup time': ['2021-02-07 08:00:00'],
    'Sleep duration': [9.0],
    'REM sleep percentage': [24],
    'Deep sleep percentage': [67],
    'Light sleep percentage': [52],
    'Awakenings': [1.0],
    'Caffeine consumption': [0.0],
    'Alcohol consumption': [0.0],
    'Smoking status': ['No'],
    'Exercise frequency': [4.0]
})

predict_input_low_caffeine_no_awaking = pd.DataFrame.from_dict({
    'Age': [34],
    'Gender': ['Male'],
    'Bedtime': ['2021-02-06 23:00:00'],
    'Wakeup time': ['2021-02-07 08:00:00'],
    'Sleep duration': [9.0],
    'REM sleep percentage': [24],
    'Deep sleep percentage': [67],
    'Light sleep percentage': [52],
    'Awakenings': [0.0],
    'Caffeine consumption': [0.0],
    'Alcohol consumption': [0.0],
    'Smoking status': ['No'],
    'Exercise frequency': [4.0]
})


def predict(to_predict):
    print(f'>> Test data for single prediction:\n{to_predict}\n')
    to_predict_transformed = ordinal_encoder.transform(
        to_predict)
    prediction = model_best_gbr.predict(to_predict_transformed)
    print(f'>> Predicted sleep efficiency:\n{prediction}\n')


predict(predict_input_high_caffeine)
predict(predict_input_low_caffeine)
predict(predict_input_low_caffeine_no_awaking)

```

Es folgen ein paar Predictions bei denen in etwa das Schlafverhalten des Autors angegeben wurde mit ein paar Veränderungen um zu sehen, wie diese sich auf den vorhergesagten Schlaf-Effizienz Wert auswirken.

```
>> Test data for single prediction:
   Age Gender              Bedtime          Wakeup time  Sleep duration  \
0   34   Male  2021-02-06 23:00:00  2021-02-07 08:00:00             9.0

   REM sleep percentage  Deep sleep percentage  Light sleep percentage  \
0                    24                     67                      52

   Awakenings  Caffeine consumption  Alcohol consumption Smoking status  \
0         1.0                 200.0                  0.0             No

   Exercise frequency
0                 4.0

>> Predicted sleep efficiency:
[0.68403902]

>> Test data for single prediction:
   Age Gender              Bedtime          Wakeup time  Sleep duration  \
0   34   Male  2021-02-06 23:00:00  2021-02-07 08:00:00             9.0

   REM sleep percentage  Deep sleep percentage  Light sleep percentage  \
0                    24                     67                      52

   Awakenings  Caffeine consumption  Alcohol consumption Smoking status  \
0         1.0                   0.0                  0.0             No

   Exercise frequency
0                 4.0

>> Predicted sleep efficiency:
[0.68359057]

>> Test data for single prediction:
   Age Gender              Bedtime          Wakeup time  Sleep duration  \
0   34   Male  2021-02-06 23:00:00  2021-02-07 08:00:00             9.0

   REM sleep percentage  Deep sleep percentage  Light sleep percentage  \
0                    24                     67                      52

   Awakenings  Caffeine consumption  Alcohol consumption Smoking status  \
0         0.0                   0.0                  0.0             No

   Exercise frequency
0                 4.0

>> Predicted sleep efficiency:
[0.71025232]
```

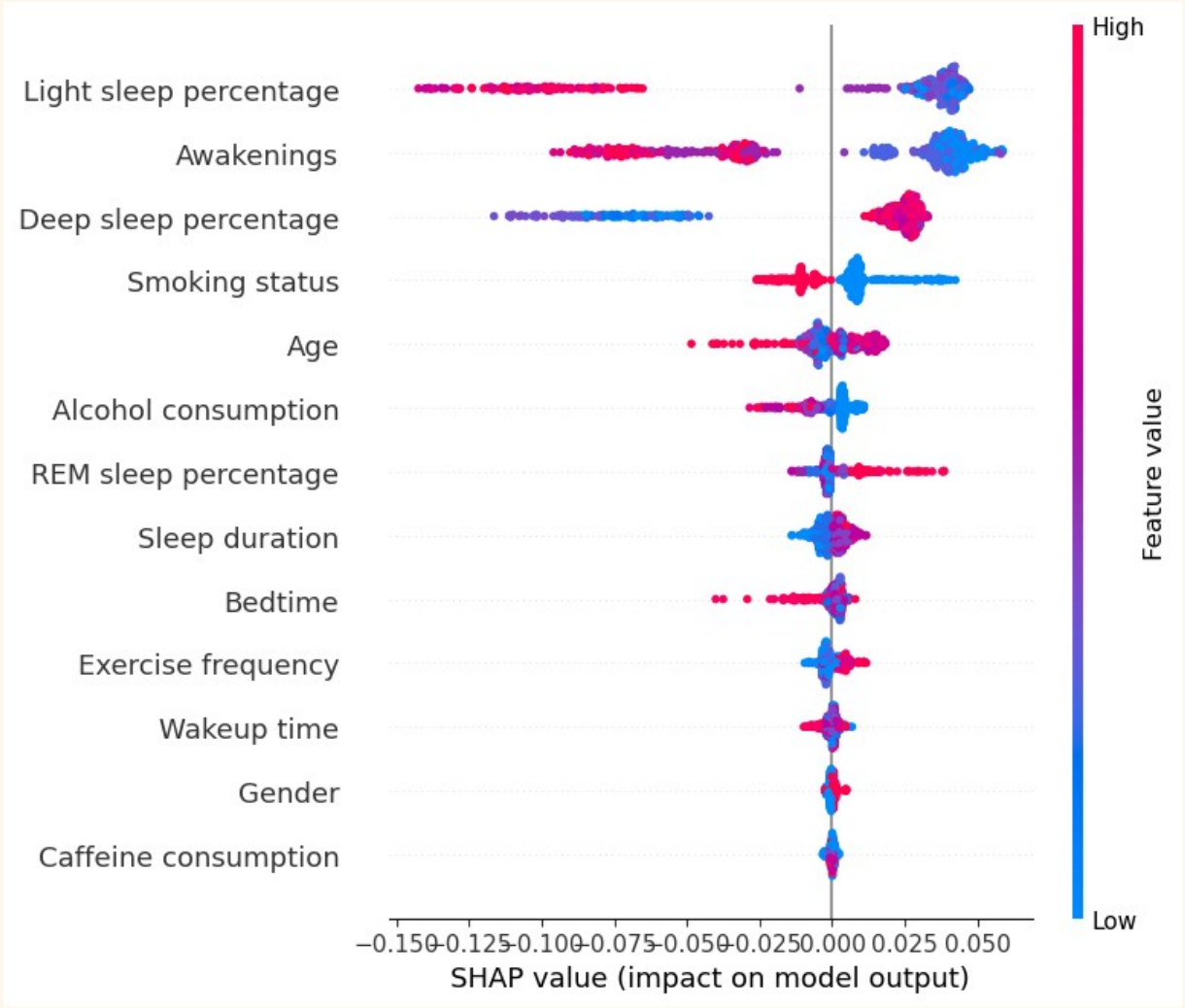
Man sieht, dass durch weniger Kaffe-Konsum der Wert etwas schlechter wird und durch das Vermeiden von zwischenzeitlichem Aufwachen der Wert wiederum etwas größer wird.

```
320  # %%
321  # Plot most important features with shapley
322
323  shap.initjs()
324  explainer = shap.TreeExplainer(model_best_gbr)
325  shap_values = explainer.shap_values(X)
326
327  i = 4
328  shap.force_plot(explainer.expected_value,
329                  shap_values[i], features=X.iloc[i], feature_names=X.columns)
330
     Zelle Ausführen | Oben laufen | Zelle debuggen | Gehe zu [7]
331  # %%
332  shap.summary_plot(shap_values, features=X, feature_names=X.columns)
333
     Zelle Ausführen | Oben laufen | Zelle debuggen | Gehe zu [8]
334  # %%
335  shap.summary_plot(shap_values, features=X,
336                    feature_names=X.columns, plot_type='bar')
337
```

Um besser zu verstehen, welche Veränderungen die Schlaf-Effizienz wie stark verbessern oder verschlechtern, wurden mit der shap-Bibliothek einige Diagramme erstellt, aus denen man dies, mit geschultem Auge, gut ablesen kann.

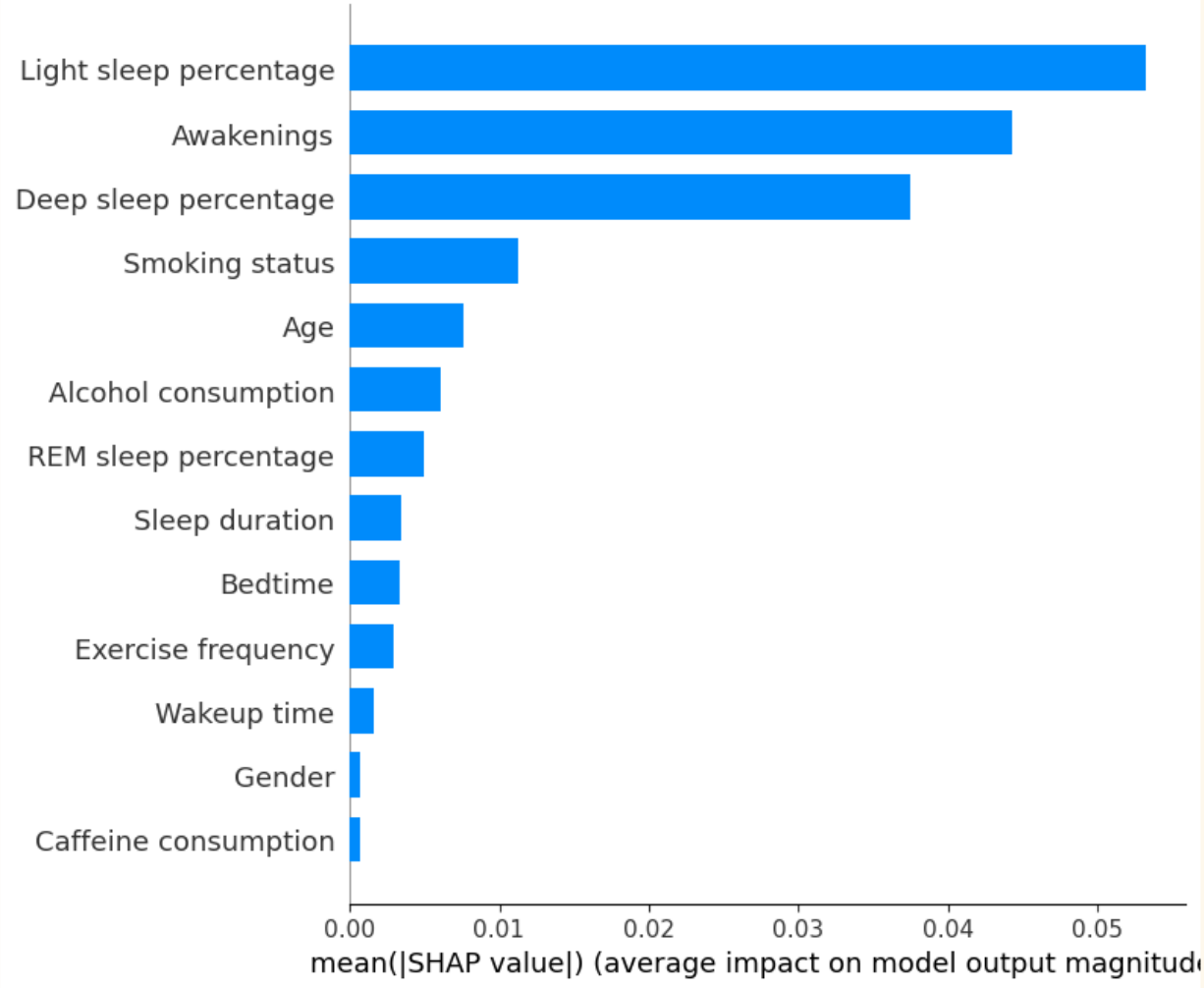
Das Kräfte-Diagramm.

```
✓ shap.summary_plot(shap_values, features=X, feature_names=X.columns) ···
```



Das Zusammenfassungs-Diagram.

```
shap.summary_plot(shap_values, features=X, ...
```



Das Zusammenfassungs-Diagramm in der Balken-Variante.
Hier erkennt man ganz deutlich, dass die Features “Light sleep percentage”, “Awakenings” und “Deep sleep percentage” den größten Einfluss auf die finale Vorhersage haben.
Diese kann man ohne spezielle Messgeräte jedoch nicht gut messen. Dem normalen Nutzer bleiben dadurch aber immernoch die mäßig großen Einflüsse von “Smoking status”, “Alcohol consumption”, “Sleep duration”, “Bedtime” und “Exercise-Frequency”.
(Auf sein Alter oder REM-Schlaf-Anteil kann man bewusst keinen Einfluss nehmen.)

2.6 Ergebnisse

Mit dem hier gefundenen, optimierten und trainierten GradientBosstingRegressor kann man nun mit einer Vorhersage-Genauigkeit von 82% die eigenen Schlafgewohnheiten angeben und eine Vorhersage darüber generieren lassen, wie gut man unter diesen Bedingungen schlafen können müsste.

Durch ein wenig Versuch und Fehler beim Verstellen der größten Einflussfaktoren, kann man so seine Parameter anpassen und probieren die besten Schlafbedingungen zu finden, die man unter seinen Umständen erwirken kann.
Diese kann man dann im eigenen Privatleben ausprobieren und nachvollziehen, ob einem diese Veränderungen zu mehr Lebensqualität während der Wachphasen verhelfen.

2.7 Ausblick

Der hier gefundene Regressor ist noch relativ einfach und resourcenschonend. Währe noch mehr Genauigkeit vonnöten könnte man ausprobieren einen der größeren/komplexeren Regressoren, durch Hyperparameter Optimierung, noch besser zu machen, als den hier optimierten Gradient-Bossting-Regressor.
Da es bei der eigenen Schlafoptimierung jedoch hauptsächlich um relative Veränderungen geht, also was ist besser und was ist schlechter an Stelle von, was ist das absolut Beste, dürfte auch eine 80% Genauigkeit dafür vollständig ausreichen.