ML/DL Assignment - Sleep Age Prediction

Clayton Coulter

Predicting Age by Sleep and Demographic

- Wanted to use "NCH Sleep DataBank: A Large Collection of Real-world Pediatric Sleep Studies with Longitudinal Clinical Data" dataset.
- Have always had an interest in sleep studies
- Wanted to see if demographic, diagnosis, and sleep amount would be a good indicator for predicting the age.
- **Diagnosis dataset contains the following features:** STUDY_DX_ID, STUDY_ENC_ID, STUDY_PAT_ID, DX_START_DATETIME, DX_END_DATETIME, DX_SOURCE_TYPE, DX_ENC_TYPE, DX_CODE_TYPE, DX_CODE, DX_NAME, DX_ALT_CODE, CLASS_OF_PROBLEM, CHRONIC_YN, PROV_ID.
- **Demographic dataset contains the following features:** STUDY_PAT_ID, BIRTH_DATE, PCORI_GENDER_CD, PCORI_RACE_CD, PCORI_HISPANIC_CD, GENDER_DESCR, RACE_DESCR, ETHNICITY_DESCR, LANGUAGE_DESCR, PEDS_GEST_AGE_NUM_WEEKS, PEDS_GEST_AGE_NUM_DAYS
- **Sleep Study dataset contains the following features:** STUDY_PAT_ID, SLEEP_STUDY_ID, SLEEP_STUDY_START_DATETIME, SLEEP_STUDY_DURATION_DATETIME, AGE_AT_SLEEP_STUDY_DAYS

Importing Libraries

```
#import the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import StackingClassifier
from sklearn.datasets import make_classification
```

- Need to import all the necessary libraries and functions in order to conduct our analysis.
- **pandas (pd)**: pandas is a powerful data manipulation library in Python. It provides data structures and functions needed to manipulate numerical tables and time series data.
- **numpy (np):** numpy is a fundamental package for scientific computing in Python. It provides support for arrays, matrices, and mathematical functions to operate on these arrays efficiently.
- **matplotlib.pyplot (plt):** matplotlib is a plotting library for Python. pyplot is a module in matplotlib that provides a MATLAB-like interface for creating plots and visualizations.
- **seaborn (sns):** seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **sklearn.model_selection.train_test_split:** This function is used to split the dataset into training and testing sets for model evaluation and validation.
- **sklearn.metrics.r2_score:** r2_score is a function from scikit-learn (sklearn) used to compute the R-squared regression score.
- **sklearn.metrics.mean_squared_error:** mean_squared_error is a function from scikit-learn used to compute the mean squared error regression loss.
- **sklearn.preprocessing.MinMaxScaler:** MinMaxScaler is a function from scikit-learn used for feature scaling. It scales features to a given range, often between 0 and 1.
- **sklearn.neighbors.KNeighborsRegressor:** KNeighborsRegressor is a class from scikit-learn used for regression based on k-nearest neighbors algorithm.
- **sklearn.svm.SVR:** SVR stands for Support Vector Regressor. It's a class from scikit-learn used for Support Vector Machine regression.
- **sklearn.ensemble.**RandomForestRegressor: RandomForestRegressor is a class from scikit-learn used for regression based on random forest algorithm.
- **sklearn.tree.DecisionTreeRegressor:** DecisionTreeRegressor is a class from scikit-learn used for regression based on decision tree algorithm.
- **sklearn.ensemble.GradientBoostingRegressor:** GradientBoostingRegressor is a class from scikit-learn used for gradient boosting regression.
- **sklearn.model_selection.GridSearchCV:** GridSearchCV is a class from scikit-learn used for hyperparameter tuning via grid search.
- **sklearn.ensemble.StackingClassifier:** StackingClassifier is a class from scikit-learn used for stacking multiple classifiers for better performance.
- **sklearn.datasets.make_classification:** make_classification is a function from scikit-learn used to generate synthetic classification datasets.

Load and Read our Datasets

- Import our Sleep Study, demographic, and diagnosis data leveraging Google Colab Import files function.
- Let's read and save our datasets to a variable

```
#import Sleep Study data
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving SLEEP_STUDY.csv to SLEEP_STUDY.csv
```

```
[ ]
#import demographic data
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving DEMOGRAPHIC.csv to DEMOGRAPHIC.csv
```

```
[ ]
#import diagnosis data
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving DIAGNOSIS.csv to DIAGNOSIS.csv
```

```
[ ]
#patients information
df = pd.read_csv('SLEEP_STUDY.csv')

#read admission data
df_dem = pd.read_csv('DEMOGRAPHIC.csv')

#read diagnosis data
df_dia = pd.read_csv('DIAGNOSIS.csv')

```

Analyze our Features and Calculate Age

- Multiple Features could have been used to calculate age. However, I chose the ‘AGE_AT_SLEEP_STUDY_DAYS’ to calculate the age.
- Calculated the age by dividing the number of days /365
- New columns was created in our Sleep Study Dataframe named age

```
[ ]  list(df)
```

```
['STUDY_PAT_ID',
 'SLEEP_STUDY_ID',
 'SLEEP_STUDY_START_DATETIME',
 'SLEEP_STUDY_DURATION_DATETIME',
 'AGE_AT_SLEEP_STUDY_DAYS']
```

```
[ ]  #visualize all the demographic columns
     list(df_dem)
```

```
['STUDY_PAT_ID',
 'BIRTH_DATE',
 'PCORI_GENDER_CD',
 'PCORI_RACE_CD',
 'PCORI_HISPANIC_CD',
 'GENDER_DESCR',
 'RACE_DESCR',
 'ETHNICITY_DESCR',
 'LANGUAGE_DESCR',
 'PEDS_GEST_AGE_NUM_WEEKS',
 'PEDS_GEST_AGE_NUM_DAYS']
```

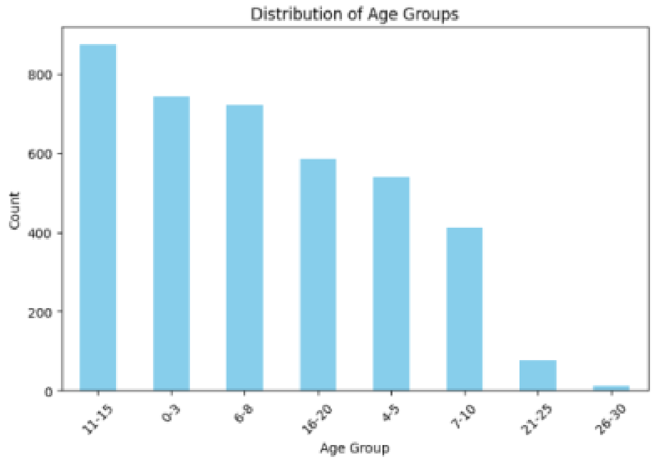
```
[ ]  # Calculate age by dividing the number of days by 365
     df['age'] = (df['AGE_AT_SLEEP_STUDY_DAYS'] / 365).astype(int)

     # Display the DataFrame with the calculated ages
     print(df)
```

```
      STUDY_PAT_ID  SLEEP_STUDY_ID SLEEP_STUDY_START_DATETIME  \
0                1            4789        2019-04-02 20:22:49
1                7           12595        2018-12-05 18:20:31
2               10           22339        2018-01-07 18:41:16
3               16           24241        2019-02-05 20:17:31
4               22           23233        2019-07-29 18:50:11
...            ...             ...                        ...
3979         20812           24841        2018-03-16 18:33:58
3980         20821           18340        2018-10-28 20:02:56
3981         20824            1357        2018-12-04 18:32:41
3982         20827            7042        2018-07-03 20:30:32
3983         20830           14089        2019-01-05 18:45:40

     SLEEP_STUDY_DURATION_DATETIME  AGE_AT_SLEEP_STUDY_DAYS  age
0                          9:35:32                     3338    9
1                         11:22:32                      775    2
2                         11:14:13                     4782   13
3                         10:03:36                     4408   12
4                         11:15:36                     1730    4
...                            ...                      ...  ...
3979                      11:45:15                     1912    5
3980                       9:38:16                     5723   15
3981                      11:34:21                     4522   12
3982                       9:32:59                     4883   13
3983                      12:05:09                     8771   24
```

Group our Age Calculation into ‘Age_Groups’

- Need to group our Ages into an Age group to get a better distribution for prediction
- Created a number of bins for the age groups
- Created labels for the bins from age ‘0’ all the way to age ‘30’
- Bucketted the ages into groups
- Counted the number of individuals in each age group for visualization purposes
- Created a bar plot to display the number of individuals in each age group

```python
# Define bins for age groups
bins = [0, 3, 5, 8, 10, 15, 20, 25, 30]

# Create labels for the bins
labels = ['0-3', '4-5', '6-8', '7-10', '11-15', '16-20', '21-25', '26-30']

# Bucket the ages into groups
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels, right=False)

# Count the number of individuals in each age group
age_group_counts = df['age_group'].value_counts()

# Create a bar plot to visualize the distribution of age groups
plt.figure(figsize=(8, 5))
age_group_counts.plot(kind='bar', color='skyblue')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.title('Distribution of Age Groups')
plt.xticks(rotation=45)
plt.show()
```



Extract the Hours from Sleep Duration

- Wanted to have 'SLEEP_STUDY_DURATION_DATETIME' to only show the hours that the individual slept. Did not want it to be displayed as datetime format
- Slice and get the first digit of the datetime format in the column 'SLEEP_STUDY_DURATION_DATETIME' Save the digits to the variable 'hours_of_sleep'.
- Print the data frame to see the new values

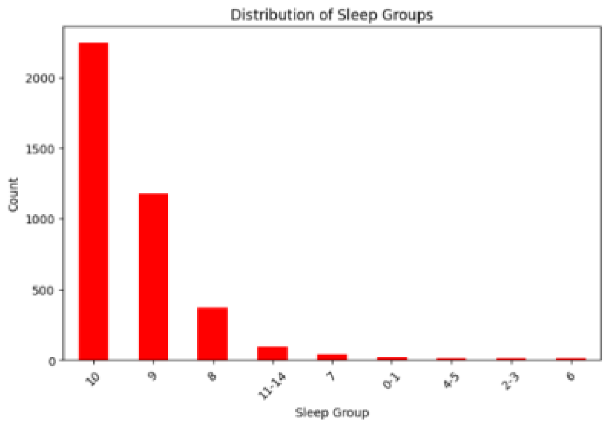
```python
# Extract hour part for the entire column
df['hours_of_sleep'] = df['SLEEP_STUDY_DURATION_DATETIME'].astype(str).apply(lambda x: int(x.split(':')[0]))

# Display the DataFrame with the extracted hour part
print(df)
```

```
      STUDY_PAT_ID  SLEEP_STUDY_ID SLEEP_STUDY_START_DATETIME  \
0                1            4789        2019-04-02 20:22:49
1                7           12595        2018-12-05 18:20:31
2               10           22339        2018-01-07 18:41:16
3               16           24241        2019-02-05 20:17:31
4               22           23233        2019-07-29 18:50:11
...            ...             ...                        ...
3979         20812           24841        2018-03-16 18:33:58
3980         20821           18340        2018-10-28 20:02:56
3981         20824            1357        2018-12-04 18:32:41
3982         20827            7042        2018-07-03 20:30:32
3983         20830           14089        2019-01-05 18:45:40

     SLEEP_STUDY_DURATION_DATETIME  AGE_AT_SLEEP_STUDY_DAYS  age age_group  \
0                          9:35:32                     3338    9      7-10
1                         11:22:32                      775    2       0-3
2                         11:14:13                     4782   13     11-15
3                         10:03:36                     4408   12     11-15
4                         11:15:36                     1730    4       4-5
...                            ...                      ...  ...       ...
3979                      11:45:15                     1912    5       6-8
3980                       9:38:16                     5723   15     16-20
3981                      11:34:21                     4522   12     11-15
3982                       9:32:59                     4883   13     11-15
3983                      12:05:09                     8771   24     21-25

      hours_of_sleep
0                  9
1                 11
2                 11
3                 10
4                 11
...              ...
3979              11
3980               9
3981              11
3982               9
3983              12

[3984 rows x 8 columns]
```

Group the Hours of Sleep for Better Distribution

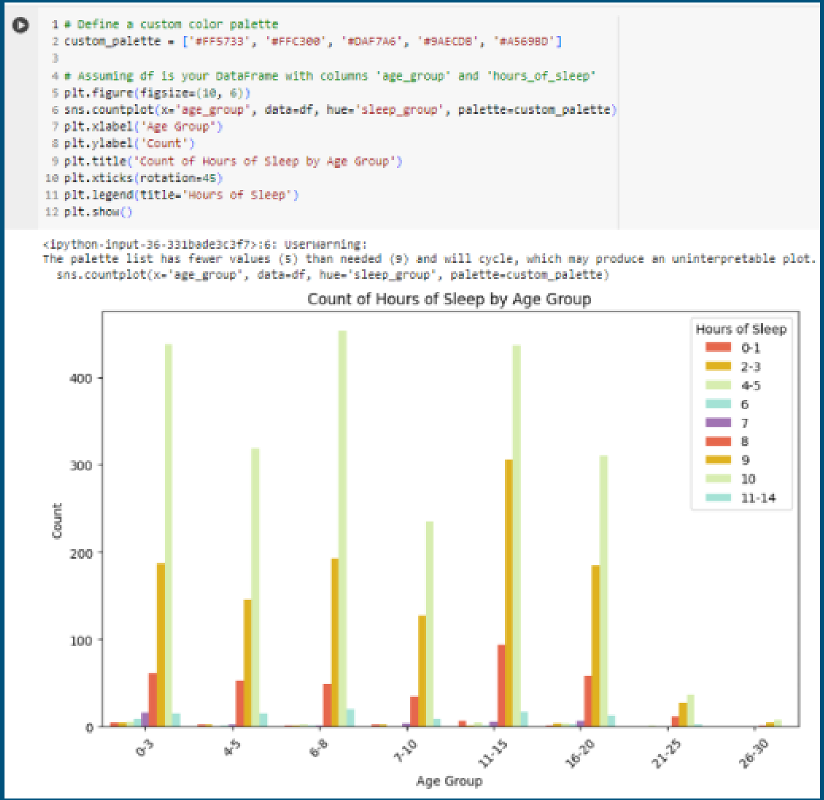
- Wanted to group the hours of sleep in order to have more even distribution of data. However, regardless of grouping, most individuals still slept 10 hours
- Define the bins for sleep duration
- Create labels for the bins
- Bucket the hours of sleep into groups
- Count the number of hours in each sleep group for visualization purposes
- Create a red bar plot to visualize the distribution of age groups

```
# Define bins for Sleep Duration
bins = [0, 2, 4, 6, 7, 8, 9, 10, 12, 14]

# Create labels for the bins
labels = ['0-1', '2-3', '4-5', '6', '7','8', '9','10', '11-14']

# Bucket the hours of sleep into groups
df['sleep_group'] = pd.cut(df['hours_of_sleep'], bins=bins, labels=labels, right=False)

# Count the number of hours in each sleep group
sleep_group_counts = df['sleep_group'].value_counts()

# Create a bar plot to visualize the distribution of age groups
plt.figure(figsize=(8, 5))
sleep_group_counts.plot(kind='bar', color='red')
plt.xlabel('Sleep Group')
plt.ylabel('Count')
plt.title('Distribution of Sleep Groups')
plt.xticks(rotation=45)
plt.show()
```



Visualize the Age Groups by Sleep Hours

- Needed to visualize the age of each individual and see how many hours of sleep they were receiving
- Create a custom palette
- Leverage seaborn library to create a countplot for visualization
- Notable Findings: Most common age for 10 hours of sleep was 6-8 year olds. Before this study I did not realize children of this age received this much sleep



Check Nulls

- Want to check through all our datasets and see if we have any missing data in any of our columns
- Leverage the isnull() function on each data frame and print true if there are any missing values

```
[ ]
#find if each of the datasets contian Null Values. We will fill those with the mean later
null_values_exist = df_dem.isnull().any().any()
print(null_values_exist, "In Demographic Data Set")

null_values_exist = df.isnull().any().any()
print(null_values_exist, "In Sleep Study Data Set")

null_values_exist = df_dia.isnull().any().any()
print(null_values_exist, "In Diagnosis Data Set")

```

```
True In Demographic Data Set
True In Sleep Study Data Set
True In Diagnosis Data Set
```

ICD9 vs. ICD10

- Want to check if there is an uneven distribution of ICD9 vs. ICD10 codes
- Leverage Value_Counts() function to count each ICD9 and ICD10 code
- ICD10 counted at 1017334 and ICD9 counted at 496519
- Want to set the amount of ICD9 codes equal to ICD10 codes to limit uneven distribution
- Get the number of occurrences for minority code
- Separate the dataset into ICD9 and ICD10 subsets
- Determine the minority and majority subsets
- Sample from the majority subset to match the count of the minority subset
- Concatenate the balanced majority subset with the minority subset
- Shuffle the dataset to ensure randomness and display
- Verify ICD9 = ICD10

```
#find the count difference in IDC Codes
df_dia['DX_CODE_TYPE'].value_counts()
```

```
ICD10    1017334
ICD9      496519
Name: DX_CODE_TYPE, dtype: int64
```

```
#lets have the number of ICD9 Codes equal the number of IDC10 Codes
icd_counts = df_dia['DX_CODE_TYPE'].value_counts()
minority_code = icd_counts.idxmin()

# Get the number of occurrences for the minority code
minority_count = icd_counts.min()

# Separate the dataset into ICD9 and ICD10 subsets
idc9_subset = df_dia[df_dia['DX_CODE_TYPE'] == 'ICD9']
idc10_subset = df_dia[df_dia['DX_CODE_TYPE'] == 'ICD10']

# Determine the minority and majority subsets
if minority_code == 'ICD9':
    minority_subset = idc9_subset
    majority_subset = idc10_subset
else:
    minority_subset = idc10_subset
    majority_subset = idc9_subset

# Sample from the majority subset to match the count of the minority subset
balanced_majority_subset = majority_subset.sample(n=minority_count, replace=True)

# Concatenate the balanced majority subset with the minority subset
df_dia = pd.concat([balanced_majority_subset, minority_subset])

# Shuffle the dataset to ensure randomness
df_dia = df_dia.sample(frac=1).reset_index(drop=True)

# Display the balanced dataset
print(df_dia.head())
```

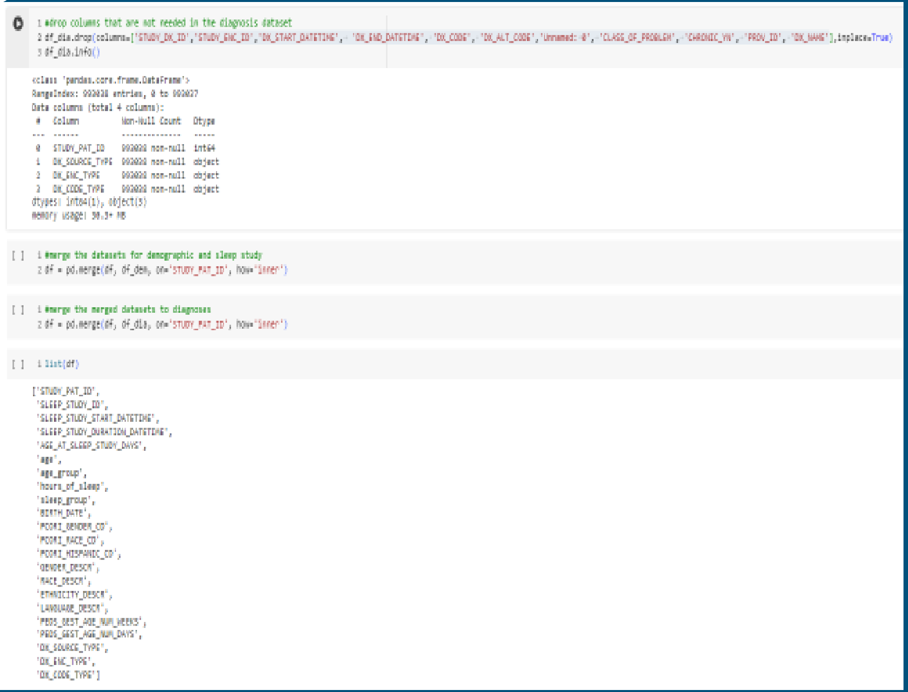
```
   Unnamed: 0  STUDY_DX_ID  STUDY_ENC_ID  STUDY_PAT_ID     DX_START_DATETIME  \
0     1452740      1580568      58591746         18133  2018-05-27 14:30:00
1     1161033      1264283      66317399         12127  2016-05-28 00:00:00
2     1343644      1461776      65886071          5347  2016-10-07 21:02:00
3      852164       932379      65717081           211  2016-02-26 00:00:00
4      360013       400486      66244184          6779  2017-09-18 09:56:00

       DX_END_DATETIME           DX_SOURCE_TYPE    DX_ENC_TYPE DX_CODE_TYPE  \
0  2018-06-27 22:59:00                 Final Dx       Final Dx        ICD10
1  2016-06-28 00:00:00  Secondary Encounter Dx   Encounter Dx        ICD10
2  2016-10-07 22:59:00                 Final Dx       Final Dx        ICD10
3  2018-02-28 00:00:00    Primary Encounter Dx   Encounter Dx        ICD10
4  2017-09-18 23:59:00                 Final Dx       Final Dx        ICD10

  DX_CODE                                    DX_NAME DX_ALT_CODE  \
```

```
#verify the ICD Codes equal each other
df_dia['DX_CODE_TYPE'].value_counts()
```

```
ICD10    496519
ICD9     496519
Name: DX_CODE_TYPE, dtype: int64
```

Drop and Merge Datasets

- Need to drop columns in Diagnosis dataset that are not valuable in prediction
  - Columns to drop: `'STUDY_DX_ID','STUDY_ENC_ID', 'DX_START_DATETIME',  'DX_END_DATETIME', 'DX_CODE', 'DX_ALT_CODE','Unnamed: 0', 'CLASS_OF_PROBLEM', 'CHRONIC_YN', 'PROV_ID', 'DX_NAME'`
- Merge all three datasets and save to the df dataframe variable



Filling in Missing Categorical and Numerical Data

- Need to eliminate any missing data
  - Categorical Data: Fill in missing data with the mode
  - Numerical Data: Fill in missing data with the mean
- Separate numeric and non-numeric columns
- Handle missing values in numeric columns with the mean with simple imputer function
- Handle missing values in non-numeric columns with mode with simple imputer function

```
#lets fill all missing values with the mean for numerical columns and them mode for categorical columns
from sklearn.impute import SimpleImputer

# Separate numeric and non-numeric columns
numeric_cols = df.select_dtypes(include='number').columns
non_numeric_cols = df.select_dtypes(exclude='number').columns

# Handle missing values in numeric columns
numeric_imputer = SimpleImputer(strategy='mean')
df[numeric_cols] = numeric_imputer.fit_transform(df[numeric_cols])

# Handle missing values in non-numeric columns
non_numeric_imputer = SimpleImputer(strategy='most_frequent')
df[non_numeric_cols] = non_numeric_imputer.fit_transform(df[non_numeric_cols])

df.info()

```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1127759 entries, 0 to 1127758
Data columns (total 22 columns):
 #   Column                         Non-Null Count    Dtype
---  ------                         --------------    -----
 0   STUDY_PAT_ID                   1127759 non-null  float64
 1   SLEEP_STUDY_ID                 1127759 non-null  float64
 2   SLEEP_STUDY_START_DATETIME     1127759 non-null  object
 3   SLEEP_STUDY_DURATION_DATETIME  1127759 non-null  object
 4   AGE_AT_SLEEP_STUDY_DAYS        1127759 non-null  float64
 5   age                            1127759 non-null  float64
 6   age_group                      1127759 non-null  object
 7   hours_of_sleep                 1127759 non-null  float64
 8   sleep_group                    1127759 non-null  object
 9   BIRTH_DATE                     1127759 non-null  object
 10  PCORI_GENDER_CD                1127759 non-null  object
 11  PCORI_RACE_CD                  1127759 non-null  object
 12  PCORI_HISPANIC_CD              1127759 non-null  object
 13  GENDER_DESCR                   1127759 non-null  object
 14  RACE_DESCR                     1127759 non-null  object
 15  ETHNICITY_DESCR                1127759 non-null  object
 16  LANGUAGE_DESCR                 1127759 non-null  object
 17  PEDS_GEST_AGE_NUM_WEEKS        1127759 non-null  float64
 18  PEDS_GEST_AGE_NUM_DAYS         1127759 non-null  float64
 19  DX_SOURCE_TYPE                 1127759 non-null  object
 20  DX_ENC_TYPE                    1127759 non-null  object
 21  DX_CODE_TYPE                   1127759 non-null  object
dtypes: float64(7), object(15)
memory usage: 197.9+ MB
```

Transform Predictor and Columns to Numeric

- Transform Age_group which is our predictor to a numeric value. Machine Learning algorithms need to have their predictor as a numeric value and not categorical
- Will be able to transform using the LabelEncoder() function
- Drop the unused or no longer needed columns from our newly merged dataset
- Create dummy columns for categorical variables and transform all the categorical variables into numerical values with new prefix
    - Machine Learning Values need all columns to be numeric for ease of computation

```python
from sklearn.preprocessing import LabelEncoder

# Instantiate LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the 'sleep_group' column
df['age_group_numeric'] = label_encoder.fit_transform(df['age_group'])
```

```python
# Drop unused or no longer needed columns
df.drop(columns=['STUDY_PAT_ID','SLEEP_STUDY_DURATION_DATETIME','SLEEP_STUDY_START_DATETIME','AGE_AT_SLEEP_STUDY_DAYS','BIRTH_DATE','SLEEP_STUDY_ID', 'hours_of_sleep','age', 'age_group', 'PEDS_GEST_AGE_NUM_DAYS','PEDS_GEST_AGE_NUM_WEEKS' ], inplace=True)



df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1127759 entries, 0 to 1127758
Data columns (total 12 columns):
 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   sleep_group         1127759 non-null  object
 1   PCORI_GENDER_CD     1127759 non-null  object
 2   PCORI_RACE_CD       1127759 non-null  object
 3   PCORI_HISPANIC_CD   1127759 non-null  object
 4   GENDER_DESCR        1127759 non-null  object
 5   RACE_DESCR          1127759 non-null  object
 6   ETHNICITY_DESCR     1127759 non-null  object
 7   LANGUAGE_DESCR      1127759 non-null  object
 8   DX_SOURCE_TYPE      1127759 non-null  object
 9   DX_ENC_TYPE         1127759 non-null  object
 10  DX_CODE_TYPE        1127759 non-null  object
 11  age_group_numeric   1127759 non-null  int64
dtypes: int64(1), object(11)
memory usage: 111.9+ MB
```

```python
# create dummy columns for categorical variables
prefix_cols = ['DXS','DXE','DXC','SLEEP','PCORI_G_', 'PCORI_R_', 'PCORI_H_', 'SEX', 'RACE', 'ETHNICITY', 'LANGUAGE']
dummy_cols = ['DX_SOURCE_TYPE','DX_ENC_TYPE','DX_CODE_TYPE','sleep_group', 'PCORI_GENDER_CD', 'PCORI_RACE_CD', 'PCORI_HISPANIC_CD', 'GENDER_DESCR', 'RACE_DESCR', 'ETHNICITY_DESCR', 'LANGUAGE_DESCR' ]
df = pd.get_dummies(df, prefix=prefix_cols, columns=dummy_cols)
df.info()

```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1127759 entries, 0 to 1127758
Data columns (total 88 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   age_group_numeric            1127759 non-null  int64
 1   DXS_Active                   1127759 non-null  uint8
 2   DXS_Admit Dx                 1127759 non-null  uint8
 3   DXS_External Injury Dx       1127759 non-null  uint8
 4   DXS_Final Dx                 1127759 non-null  uint8
 5   DXS_Primary Encounter Dx     1127759 non-null  uint8
```

Drop the Predictor, Train Data, Run the Models

- First, need to drop the predictor in order to leverage our features for prediction
    - Will drop the new numeric Age called ‘age_group_numeric’
- Split training data for 80% and 20% for testing
- Show the results
- Find our best model for prediction. Leverage GradientBoostingRegressor, KNeighborsRegressor, RandomForestRegressor
- Loop through each model and train/fit the data
- Make Predictions
- Calculate the R2 to show which model performed the best
- Notable Findings: Random Forest Regressor performed the best, however, only at 13%

```
[ ]
# Target Variable (Age_Group)
age = df['age_group_numeric'].values
# Prediction Features
features = df.drop(columns=['age_group_numeric'])
```

```
[ ]
# Split into train 80% and test 20%
X_train, X_test, y_train, y_test = train_test_split(features,
                                                    age,
                                                    test_size = .20,
                                                    random_state = 0)

# Show the results of the split
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Training set has 902207 samples.
Testing set has 225552 samples.
```

```
[ ]
# Regression models for comparison
models = [GradientBoostingRegressor(random_state = 0),
          KNeighborsRegressor(),
          RandomForestRegressor(random_state = 0)]

results = {}

for model in models:

    # Instantiate and fit Regressor Model
    reg_model = model
    reg_model.fit(X_train, y_train)

    # Make predictions with model
    y_test_preds = reg_model.predict(X_test)

    # Grab model name and store results associated with model
    name = str(model).split("(")[0]

    results[name] = r2_score(y_test, y_test_preds)
    print('{} done.'.format(name), results[name])
```

```
GradientBoostingRegressor done. 0.07068845377466448
KNeighborsRegressor done. -0.021265711738385873
RandomForestRegressor done. 0.13105279511723633
```

Display the Top 20 Features for Prediction

- Activate the RandomForestRegressor Model
- Fit the model
- Create a data frame that holds the important features
- Print and display
- Notable Findings:
    - DXC_IDC9 was the prominent predictor for age prediction

```
[ ]
# Instantiate the RandomForestRegressor model
reg_model = RandomForestRegressor()

# Fit the model to the training data
reg_model.fit(X_train, y_train)

# Create a DataFrame with feature importances
feature_imp = pd.DataFrame(reg_model.feature_importances_,
                           index=X_train.columns,
                           columns=['importance']).sort_values('importance', ascending=False)

# Display the first 20 rows of the DataFrame
print(feature_imp.head(20))
```

```
                 importance
DXC_ICD9           0.066043
DXC_ICD10          0.060523
LANGUAGE_English   0.052971
LANGUAGE_Somali    0.045406
SLEEP_10           0.044950
SLEEP_7            0.040949
SLEEP_9            0.038771
SLEEP_11-14        0.034664
SLEEP_4-5          0.033382
LANGUAGE_Spanish   0.029319
PCORI_R__05        0.027234
LANGUAGE_Arabic    0.025737
RACE_White         0.025630
SLEEP_0-1          0.021377
PCORI_R__02        0.020834
LANGUAGE_Fulani    0.019966
SEX_Male           0.019623
PCORI_G__M         0.019348
RACE_Asian         0.019138
SEX_Female         0.017998
```

Ensemble Learning: Stacking

- Stacking:ensemble learning technique that combines multiple base learners (or base models) to improve predictive performance. Instead of simply averaging the predictions of individual models as in traditional ensembles like bagging or boosting, stacking uses another model (called a meta-learner or blender) to learn how to best combine the predictions of the base models.
- In our Stacking, we will use Decision Tree Classifier, Random Forest Classifier, and Logistic Regression
- Additionally we will Finetune our parameters to improve our prediction
- Split training and test data
- Define models and classifiers with parameters
- Create the meta-classifier
- Creating stacking classifier
- Fit the tracking classifier
- Evaluate the stacking classifier
- Notable Findings:
  - Accuracy was around 36% when leveraging Stacking Machine Learning Technique
- Learning Outcome:
  - Difficult to predict the hours of sleep based on diagnosis data and demographic, however, leveraging the stacking method can help improve the accuracy. Children are receiving more sleep than I thought in my original hypothesis. I thought in this modern age kids would only be receiving 6-7 hours due to our current tech age.

```
[ ]
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier
from sklearn.metrics import accuracy_score


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, age, test_size=0.2, random_state=42)

# Define base classifiers
base_classifiers = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('gb', GradientBoostingClassifier(n_estimators=100, random_state=42)),
    ('dt', DecisionTreeClassifier(random_state=42))]

# Define meta-classifier
meta_classifier = LogisticRegression()

# Create stacking classifier
stacking_classifier = StackingClassifier(estimators=base_classifiers, final_estimator=meta_classifier)

# Train the stacking classifier
stacking_classifier.fit(X_train, y_train)

# Evaluate the stacking classifier
accuracy = stacking_classifier.score(X_test, y_test)
print("Accuracy:", accuracy)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Accuracy: 0.3623554657019224
```