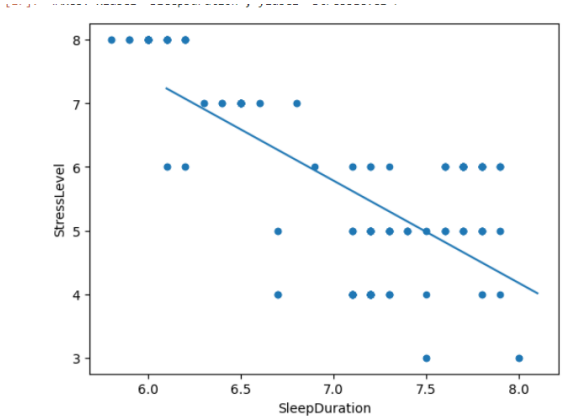
Regression

I analyzed the SleepHealthLifestyle data set from Kaggle.

Using Linear Regression

```
[17]: #below, we are trying to now make a visualization of our model
      #since there are only 2 variables involved, we can easily make a visualization

      import numpy as np
      X_new = pd.DataFrame()

      #use np.linspace to make a plot of values from sleep duration of 6.1 to 8.1 spread out
      X_new["SleepDuration"] = np.linspace(6.1, 8.1,num = 100)

      y_new_ = pd.Series(
          model.predict(X_new),         #predicts y: StressLevel   y values in Series.plot.line()
          index = X_new["SleepDuration"] #uses x: SleepDuration    x values in Series.plot.line()
      )

      #now plot the data, then the model
      df_train.plot.scatter(x="SleepDuration",y="StressLevel")
      y_new_.plot.line()
```



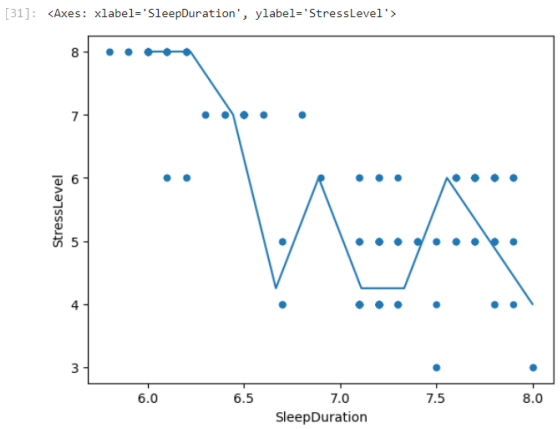
I decided to start off my data analysis simply by just examining the relationship between the duration of sleep and the stress levels of an individual.

As we can see above, I built a linear regression model that predicts a person's Stress Level based on their Duration of Sleep (Number of Hours of Sleep Received), then I made a scatter plot with a fit line.

As we can see, the fitted line is sloping downwards in the model. The stress levels tend to be higher when someone has had less hours of sleep, while the stress levels tend to be lower when someone has had more hours of sleep. From this visual, we can conclude that there is a relationship between Sleep Duration and Stress Levels.

Using K Nearest Neighbors Regression Function

```
[29]: X_train1 = df_train[["SleepDuration"]]
      y_train1 = df_train["StressLevel"]

[30]: #Below, we will be making a K Nearest Neighbors Regression Function
      model = KNeighborsRegressor(n_neighbors = 4)
      model.fit(X=X_train1, y=y_train1)
      #trying to fit and build a model predicting with the 4 neighbors surrounding

[30]: KNeighborsRegressor
      KNeighborsRegressor(n_neighbors=4)

[27]: import numpy as np

[31]: #Plotting this:
      X_new = pd.DataFrame()
      X_new["SleepDuration"] = np.linspace(6,8,num = 10)

      #Making prediction on stress level based on these values
      y_new_ = pd.Series(
          model.predict(X_new),
          index = X_new["SleepDuration"]
      )

      #plot the predictions
      df_train.plot.scatter(x="SleepDuration",y="StressLevel")
      y_new_.plot.line()

[31]: <Axes: xlabel='SleepDuration', ylabel='StressLevel'>
```



Above, I have made a k nearest neighbors regression function with 4 neighbors to predict Stress Levels based on Sleep Duration in the span between 6 – 8 hours.

The visual demonstrates the relationship between Sleep Duration and Stress Levels. However, this visual is different from the linear regression model as this visual displays the resulting Stress Levels of individuals in a piecewise manner because we are trying to predict the Stress Levels using the idea of nearest neighbors.

For example, the Stress Levels for those whose sleep duration is between 6.0 – 6.5 hours is 8 as those data points in the similar sleep duration period have the same 4 nearest neighbors when predicting stress levels. Another example is that those with a sleep duration in the range between 7.0 – 7.5 hours all have a predicted Stress Level of

approximately 4.5, as demonstrated by the fitted line. Data points near the 7.0 – 7.5 hour Sleep Duration range have the same nearest neighbors, which is why the Stress Levels are predicted to be approximately 4.5 for all those data points, making the stress level prediction constant for this range of sleep duration periods.

Comparing the Results of Stress Levels from SleepDuration: Linear Regression Vs. K Nearest Neighbors:

I think that the K Nearest Neighbors model to predict Stress Levels is a more accurate depiction, compared to the Linear Regression Model.

The linear Regression Model does a great job at giving us the general idea that the more hours a person spend sleeping, the lower their stress levels will be. The line shows a clear negative correlation between Sleep Duration and Stress Levels.

However, the fitted line in the Linear Regression model fails to accurately show that data points in a similar SleepDuration range have similar StressLevels. While the individual data points in the linear model all lie next to each other, the fitted line does not group them together appropriately for analysis. The K Nearest Neighbors model is more effective for this as it groups the data in a piecewise manner based on similarity of Stress Levels and Sleep Duration using K Nearest Neighbors. From the K Nearest Neighbors model, we can tell that those who sleep a certain amount of hours will have a very similar Stress Level, which is how this model is more accurate.

RMSE & MAE SleepDuration & StressLevel

```
       /.  , /.  , /.  ])

[9]: #use Scikit to Calculate the Mean Squared Error:
     #y_train: actual data
     #y_train_: predcited data
     from sklearn.metrics import mean_squared_error
     mse = mean_squared_error(y_train, y_train_)
     mse

[9]: 0.41044776119402987

[10]: rmse = np.sqrt(mse)
      rmse

[10]: 0.6406619710846195

[ ]:
```

I fit the data by creating a pipeline that included KNeighborsRegressor with n_neighbors = 4. I then used the Skicit package to calculate the MSE and the RMSE.

As a result, I obtained a RMSE of 0.64.

The RMSE of 0.64 tells us that the K Neighbors model prediction of Stress Levels based on Duration of Sleep is off by an average of 0.64. This implies that the using K Nearest Neighbors model to fit the data on Stress Levels may not be the most accurate.

```
[21]: MAE = absYDiff.mean()
      MAE

[21]: 0.43283582089552236
```

As shown by the Mean Absolute Error above, the average variance between Predicted Stress Levels and Actual Stress Levels is 0.43. Our MAE of 0.43 is not terrible, however our K Neighbors model definitely does not predict Stress Levels perfectly.

Section 5.5 K Fold Cross Validation

Now, I decided that I would like to use 2 variables to predict a Stress Level.

I am using SleepDuration and SleepQuality variables from the data set to make predictions on StressLevel.

Basically, I pulled out the first 200 rows from the data set to make the train data set. I then randomized the data, and 50% of the random data went into train, and 50% of the random data went into val. I am using the train data set to predict val. I made a pipeline using K Nearest Neighbors value of 4, and fit the pipeline using the train data. I then used the *predict* function on the pipeline to predict the y values on the validation data sets.

```
[18]: from sklearn.metrics import mean_squared_error
      #now, lets calculate the RMSE on the Validation Data Set
      #y_val : actual data
      #y_val_ : predicted data
      rmse = np.sqrt(mean_squared_error(y_val, y_val_))
      rmse

[18]: 0.6154266812545586
```

I then computed the Root Mean Square error, and got a value of 0.61. This means that the average difference between the actual StressLevel and the predicted StressLevel is around 0.61. 0.61 is a relatively high RMSE, so this model does not necessarily predict the StressLevel precisely from SleepDuration and SleepDuration.

```
#writing a function for this
def get_val_error(train, val):

    # extract features and label from training set.
    X_train = train[["SleepDuration", "SleepQuality"]]
    y_train = train["StressLevel"]

    # eefine pipeline and fit to training set
    pipeline = make_pipeline(
          StandardScaler(),
          KNeighborsRegressor(n_neighbors=4)
    )
    pipeline.fit(X=X_train, y=y_train)

    # extract features and label from validation set
    X_val = val[["SleepDuration", "SleepQuality"]]
    y_val = val["StressLevel"]

    # get model's predictions on validation set
    y_val_ = pipeline.predict(X_val)

    # calculate RMSE on validation set
    rmse = np.sqrt(mean_squared_error(y_val, y_val_))

    return rmse
```

```
[20]: #uses first half (train) of training dataset to predict 2nd half (val)
      get_val_error(train,val)

[20]: 0.6154266812545586
```

```
[21]: #uses second half(val) of training dataset to predict 1st Half(train)
      get_val_error(val,train)

[21]: 0.525
```

Basically, the idea of cross validation is that you use the first half of the training dataset to predict the 2nd half of the data, or you can use the 2nd half of the training dataset to predict the first half of the training dataset.

In the photograph above, I use the train data (1st half) to predict the val data (2nd half) and I use the val data(2nd half) to predict the train data (1st half). I obtain the RMSE for both. As we can see, their RMSE is not the same, as we have an RMSE of 0.61 when using train to predict val, and an RMSE of 0.525 when using val to predict train. This means that using val to predict train for StressLevel is more accurate than using train to predict val for the StressLevel Variable.

Cross Validation Section 5.5

I learned that using half of the data set in cross validation to predict Stress Levels would not give us accurate results, so I decided to use the K Fold Cross Validation Method instead to try and predict StressLevel.

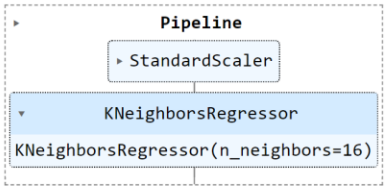
I made 4 folds, where 3 subsamples are used as training data, and 1 subsample represents val, which is the data that will be predicted. I computed the RMSE after using K Fold Cross Validation to predict StressLevel, which was 0.67. To my surprise, the RMSE was way higher compared to using cross validation.

However, changed CV = 4 to CV = 100, and I noticed that the RMSE had significantly lowered to 0.40 when using 100 folds instead of just 4 folds. This RMSE was significantly lower than the RMSE found for cross validation.

What I can conclude about my findings from K Fold Cross Validation and RMSE is that I would need at least 100 Folds for my model to accurately predict StressLevels based on SleepQuality and SleepDuration.

Model Selection and Hyperparameter Tuning

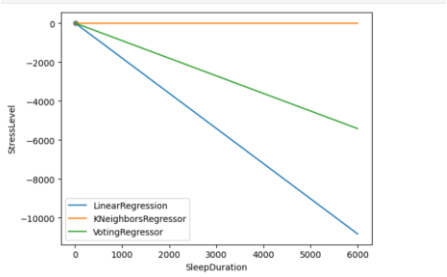
```
[51]: from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(pipeline,
                           param_grid={
                               "kneighborsregressor__n_neighbors":range(1,20)
                           },
                           scoring = "neg_mean_squared_error",
                           cv=100)
grid_search.fit(X_train,y_train)
grid_search.best_estimator_
```



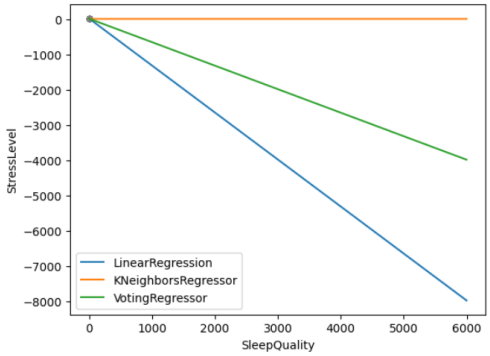
In the screen shot above, I used Hyperparameter Tuning to determine the number of nearest neighbors that will minimize the Mean Square Error in order to accurately predict StressLevel from SleepQuality and SleepDuration.

I basically fit a K Fold Cross Validation model with 100 folds, as specified by cv = 100. As a result, the number of neighbors I would need to use to get the most accurate prediction of StressLevel is 16 neighbors, as the MSE is minimized at 16 neighbors.

Lastly, I made a loadings plots with 16 neighbors.



The first loading plot above represents the relationship between SleepDuration and StressLevel.



[ ]:

The second loading plot represents the relationship between sleep quality and stress level.

From the loadings plots, we can see that as Sleep Quality and Sleep Duration increases, Stress Levels tend to decrease. These are shown from Linear Regression and Voting Regression methods. However, I notice that the K Neighbors Regressor Line is completely straight, which does not seem to be an accurate depiction between the variables.

Classification

6.1 K Nearest Neighbors for Classification

```
Name: count, dtype: int64
```

```
[18]: #K Nearest Neighbors using Skicit Learn
#import all needed packages
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline

pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsClassifier(n_neighbors=5)
)

#fit the pipeline
pipeline.fit(X_train, y_train)

#define the test data
pipeline.predict([[5.9,6]])
```

```
/opt/tljh/user/lib/python3.9/site-packages/sklearn/base.p
  warnings.warn(
```

```
[18]: array(['Normal'], dtype=object)
```

```
[19]: pipeline.predict_proba([[5.9,6]])
```

```
/opt/tljh/user/lib/python3.9/site-packages/sklearn/base.p
  warnings.warn(
```

```
[19]: array([[0.8, 0. , 0. , 0.2]])
```

For classification, I decided that I wanted to predict the BMI category that a person would fall under based on how long they sleep (SleepDuration) and the quality of their sleep (SleepQuality).

I fit a model with the X variables being SleepDuration and SleepQuality, and Y variable being “BMI Category”. I decided to do this prediction using 5 nearby neighbors.

First, I predicted the probabilities for each BMI category using sleep Duration of 5.9 Hours and Sleep Quality of 6. As we can see from the results, 80% of these people were overweight, and 20% of the these people were normal in weight. According to value counts, the majority of people with these sleep statistics were overweight.

```
]: BMICategory
   Overweight    3
   Normal        2
   Name: count, dtype: int64
```

```
[21]: pipeline.predict_proba([[5.9,6]])
```

```
/opt/tljh/user/lib/python3.9/site-packages/sklearn/b
mes
  warnings.warn(
```

```
[21]: array([[0.8, 0. , 0. , 0.2]])
```

When I used 10 nearest neighbors to predict BMICategory from SleepDuration and SleepQuality, 90% of the people were overweight and only 10% of these people were normal in weight, as shown below:

```
[11]: pipeline.predict_proba([[5.9,6]])

/opt/tljh/user/lib/python3.9/site-packages/sklearn/base.py
mes
  warnings.warn(

[11]: array([[0.9, 0. , 0. , 0.1]])
```

When utilizing 10 neighbors for prediction, I also found out from Value Counts that 7 of the people were overweight and 3 of the people were of normal weight.

```
[9]: df_sleepHealthLifestyle.loc[inds_nearest]["BMICategory"].value_counts()

[9]: BMICategory
     Overweight    7
     Normal        3
     Name: count, dtype: int64
```

Using 100 Nearest Neighbors:

```
]: BMICategory
   Overweight    66
   Normal        33
   Obese          1
   Name: count, dtype: int64

[37]: #K Nearest Neighbors using Skicit Learn
      #import all needed packages
      from sklearn.preprocessing import StandardScaler
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.pipeline import make_pipeline

      pipeline = make_pipeline(
          StandardScaler(),
          KNeighborsClassifier(n_neighbors=100)
      )

      #fit the pipeline
      pipeline.fit(X_train, y_train)

      #define the test data
      pipeline.predict([[5.9,6]])

      /opt/tljh/user/lib/python3.9/site-packages/sklearn/base.py:439: UserWarning: X
        warnings.warn(

[37]: array(['Overweight'], dtype=object)

[38]: pipeline.predict_proba([[5.9,6]])

      /opt/tljh/user/lib/python3.9/site-packages/sklearn/base.py:439: UserWarning: X
        warnings.warn(

[38]: array([[0.33, 0.  , 0.01, 0.66]])
```

Sleep Duration of 5.9 hours and Sleep Quality of 6.

I fit a model to predict the BMI categories that people fall under when they have a sleep duration of 5.9 hours and a Sleep Quality rating of 6. I found that 66% of these people fall under the Obese BMI category.

```
[36]: df_sleepHealthLifestyle.loc[inds_nearest]["BMICategory"].value_counts()

[36]: BMICategory
      Normal           58
      Overweight       33
      Normal Weight     7
      Obese             2
```

```
[39]: #K Nearest Neighbors using Skicit Learn
      #import all needed packages
      from sklearn.preprocessing import StandardScaler
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.pipeline import make_pipeline

      pipeline = make_pipeline(
          StandardScaler(),
          KNeighborsClassifier(n_neighbors=100)
      )

      #fit the pipeline
      pipeline.fit(X_train, y_train)

      #define the test data
      pipeline.predict([[8.1,9]])

      /opt/tljh/user/lib/python3.9/site-packages/sklearn/base.py:439: UserWarning: X does not ha
        warnings.warn(
[39]: array(['Normal'], dtype=object)

[40]: pipeline.predict_proba([[8.1,9]])

      /opt/tljh/user/lib/python3.9/site-packages/sklearn/base.py:439: UserWarning: X does not ha
        warnings.warn(
[40]: array([[0.58, 0.07, 0.02, 0.33]])
```

Sleep duration of 8.1hours with SleepDuration and SleepQuality of 9

I fit a model to predict the BMI categories that people fall under when they have a sleep duration of 8.1 hours and a sleep quality of 9 hours. I used 100 nearest neighbors, as I was able to get more data for each of the categories. I found that 58% of the people were normal in weight, and 2% of these people were obese. This shows that those who get more sleep are generally normal in weight.

From these comparisons using 100 nearest neighbors, I notice that those who sleep for more hours and have better quality sleep tend to fall under the normal weight BMI category. However, those who sleep for less hours and have poor quality of sleep tend to fall under the Obese BMI category.

Earlier, I had done my predictions using 5 nearest neighbors and 10 nearest neighbors. However, I think that my analysis using 100 nearest neighbors is a more accurate depiction of how sleep quality, sleep duration, and the BMI category that a person would fall under is related.

6.2 and 6.3 Precision, Recall, and F1 Score

I decided that I wanted to Calculate the precision and recall for the three BMICategories: Normal, Overweight, and Obese.

```
[6]: #Calculating Precision and Recall using Skicit Learn Package
     from sklearn.metrics import precision_score, recall_score


     (precision_score(y_train == "Normal", y_train_ == "Normal"),
      recall_score(y_train == "Normal", y_train_ == "Normal"))

[6]: (0.7991803278688525, 1.0)

[7]: (precision_score(y_train == "Obese", y_train_ == "Obese"),
      recall_score(y_train == "Obese", y_train_ == "Obese"))

[7]: (1.0, 0.4)

[8]: (precision_score(y_train == "Overweight", y_train_ == "Overweight"),
      recall_score(y_train == "Overweight", y_train_ == "Overweight"))

[8]: (0.9008264462809917, 0.7364864864864865)
```

The BMI Category of Normal has a precision score of 0.799 and a recall score of 1.0. The precision score of 0.799 means that approximately 80% of the observations that were predicted to fall under the Normal BMI category fell under the Normal BMI category correctly. The recall score of 1.0 means that 100% of the observations that actually fell under the Normal BMI category were predicted to be in the Normal BMI Category.

The BMI Category of Obese has a precision score of 1.0 and a recall score of 0.4. The precision score of 1.0 means that approximately 100% of the observations that were predicted to fall under the Obese BMI category fell under the Obese BMI category correctly. The recall score of 0.4 means that 40% of the observations that actually fell under the Obese BMI category were predicted to be in the Obese BMI Category.

The BMI Category of Overweight has a precision score of 0.9 and a recall score of 0.736. The precision score of 0.9 means that approximately 90% of the observations that were predicted to fall under the Overweight BMI category fell under the Overweight BMI category correctly. The recall score of 0.736 means that 73.6% of the observations that actually fell under the Overweight BMI category were predicted to be in the Overweight BMI Category.

Then, I decided to compute the F1 scores for each of the BMI categories to get a general idea of their accuracy.

```
[18]: #using sklearn.metrics to calculate the f1 Score
      from sklearn.metrics import f1_score
      f1_score(y_train == "Normal", y_train_ == "Normal")

[18]: 0.8883826879271071
```

```
[19]: from sklearn.metrics import f1_score
      f1_score(y_train == "Obese", y_train_ == "Obese")

[19]: 0.5714285714285715
```

```
[20]: from sklearn.metrics import f1_score
      f1_score(y_train == "Overweight", y_train_ == "Overweight")

[20]: 0.8104089219330854
```

As we can see above, I have collected the f1 scores for each of the BMI categories. The Normal BMI category has an f1 score of 0.88, and the Overweight BMI category has an f1 score of 0.81. These scores imply that the predictions for the Normal and Overweight BMI categories are relatively accurate, however, either the precision or recall score for one of these should be slightly higher to ensure more accuracy.

Unfortunately, the f1 score for the Obese category is around 57%, which is pretty low in accuracy compared to Normal and Overweight BMI. This means that either the precision or recall score is significantly low. In order to increase the f1 score, the precision or recall score must be increased.

I used hyperparameter tuning below to obtain a K value that will lead to the best f1 Score.

```
[7]: #idea of hyperparamter tuning: could we have better accuracy with a different value of k

     from sklearn.model_selection import GridSearchCV

     grid_search = GridSearchCV(
         pipeline,
         param_grid={"kneighborsclassifier__n_neighbors":range(1,50)},
         scoring = "f1_macro",
         cv = 10
     )

     grid_search.fit(X_train,y_train)
     grid_search.best_params_

[7]: {'kneighborsclassifier__n_neighbors': 3}
```

As a result, using 3 neighbors will give the highest f1 Score. This means that using 3 neighbors will most accurately predict the BMI category that a person falls under based on the variables SleepQuality, SleepDuration, and StressLevel.