Predicting Stress Levels

Moh'd Dawood Abutouq
Computer Engineering Department
University of Jordan
Amman, Jordan
mhm0197037@ju.edu.jo

***Abstract*—Predicting stress levels from sleeping data using multiple machine learning algorithms.**

Keywords—stress, sleep, machine learning, classification.

I. INTRODUCTION

Stress is defined as the physiological or psychological response to internal or external stressors. Stress involves changes affecting nearly every system of the body, influencing how people feel and behave. For example, it may be manifested by palpitations, sweating, dry mouth, shortness of breath, fidgeting, accelerated speech, augmentation of negative emotions (if already being experienced), and longer duration of stress fatigue [1].

As society evolved, we have become more prone to stress and less capable of handling it in the ideal way which leads to a multiple of serious problems that will manifest in the long run if the stress was not dealt with in a healthy manner.

Stress can be caused by a lot of factors in one's life, some of the main sources of stress include work, finances, relationships, parenting, and day-to-day inconveniences. Being continuously prone to stress leads to serious physical health conditions especially that people resort to unhealthy habits to cope with it, such as smoking and overeating.

Serious acute stress can trigger heart attacks, arrhythmias, and even sudden death. However, this happens mostly in individuals who already have heart disease [2].

Stress also takes an emotional toll. While some stress may produce feelings of mild anxiety or frustration, prolonged stress can also lead to burnout, anxiety disorders, and depression.

Chronic stress can have a serious impact on your health as well. If you experience chronic stress, your autonomic nervous system will be overactive, which is likely to damage your body.

As mentioned before stress can influence some conditions such as:

- Diabetes
- Hair loss
- Heart disease
- Hyperthyroidism
- Obesity
- Sexual dysfunction
- Tooth and gum disease

The second part of the stress response is known as the hypothalamus-pituitary-adrenal (HPA) axis. The HPA axis involves a cascade of hormones including cortisol, which is released in high amounts during times of stress. Cortisol directs energy away from processes that are not urgent, such as wound healing and immune system functioning, to help the body prepare to fight an immediate attacker.

When the perceived threat is gone, systems are designed to return to normal function via the relaxation response. But in cases of chronic stress, the relaxation response doesn't occur often enough, and being in a near-constant state of fight-or-flight can cause damage to the body.

II. STRESS AND SLEEP

Stress and sleep have a reciprocal relationship. High levels of stress can contribute to trouble sleeping, and poor-quality or insufficient sleep can lead to maladaptive changes to the stress response.

The level of the stress hormone cortisol has important implications for the sleep-wake cycle. While cortisol usually decreases at night in preparation for sleep, studies have found that people with insomnia have higher levels of cortisol in the evening, which are linked in turn to a greater number of night-time awakenings. However, more research is needed to know whether high cortisol levels cause insomnia or whether sleep problems increase cortisol levels.

Some researchers have defined one cause of short-term insomnia as a response to a stressful event, seeing the inability to sleep as a natural reaction to a potential threat. The fight-or-flight response enacts immediate physiological changes that may make it hard to sleep, including muscle tension and elevated heartrate.

Which is where this dataset aims to intervene, The better the quality of sleep, the lower the stress levels [2].

A way to monitor and control physiological stress is presented as a smart pillow which collects data of the user's sleep that can be used to predict stress levels for the next day.

Smart-Yoga Pillow (SaYoPillow) is envisioned as an edge device that may help in recognizing the importance of a good quality sleep i.e., "Smart-Sleeping", of stress variations during sleep and the following day while providing secure storage for the user's data.

The prediction analysis in SaYoPillow considers factors such as sleep latency in minutes which is the length of time the user has taken to drift in to sleep, the quality of sleep and the total number of hours the user has slept [3].

III. THE DATA

In SayoPillow.csv, you will see the relationship between the parameters-snoring range(sr) of the user, respiration rate (rr), body temperature (t), limb movement rate (lm), blood oxygen levels (bo), eye movement (rem), number of hours of sleep (sr.1), heart rate (hr) and Stress Levels (sl) (0-

low/normal, 1 –medium low, 2-medium, 3-medium high, 4 - high).
The data consists of 9 columns (8 features and 1 labels), and 630 rows, Figure.1 shows a sample of the data.

	sr	rr	t	lm	bo	rem	sr.1	hr	sl
0	93.80	25.680	91.840	16.600	89.840	99.60	1.840	74.20	3
1	91.64	25.104	91.552	15.880	89.552	98.88	1.552	72.76	3
2	60.00	20.000	96.000	10.000	95.000	85.00	7.000	60.00	1
3	85.76	23.536	90.768	13.920	88.768	96.92	0.768	68.84	3
4	48.12	17.248	97.872	6.496	96.248	72.48	8.248	53.12	0

Figure 1: Sample of the data.

Figure 2

Development environment using python in Visual Studio code and Jupiter notebook to organize the entire process.

IV. PRE-PROCESSING THE DATA

After checking the data using pandas' df.info it was shown that there were not any missing values; therefore, there was no need to modify the data by dropping any rows or columns or replacing any missing values using simple imputer.

```
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   sr      630 non-null    float64
 1   rr      630 non-null    float64
 2   t       630 non-null    float64
 3   lm      630 non-null    float64
 4   bo      630 non-null    float64
 5   rem     630 non-null    float64
 6   sr.1    630 non-null    float64
 7   hr      630 non-null    float64
 8   sl      630 non-null    int64
dtypes: float64(8), int64(1)
```

Figure 3: Information about the data.

As shown in Figure 2 the data consists exclusively of numerical values thus there was no need to handling text and converting it into desired numerical values.
It is known that machine learning algorithms do not perform very well when the input numerical attributes have very different scales, which is a flaw of this dataset therefore ScikitLearn's StandardScaler was used to standardize the input numerical attributes.

	sr	rr	t	lm	bo	rem	sr.1	hr	sl
0	1.146845	0.979066	-0.272195	1.140539	-0.271838	0.934005	-0.609407	0.979066	3
1	1.035260	0.833720	-0.353853	0.972949	-0.345696	0.873421	-0.703767	0.833720	3
2	-0.599252	-0.454206	0.907316	-0.395697	1.051448	-0.294506	1.081206	-0.454206	1
3	0.731501	0.438056	-0.576145	0.516734	-0.546753	0.708498	-0.960635	0.438056	3
4	-1.212970	-1.148636	1.438095	-1.211299	1.371498	-1.347997	1.490099	-1.148636	0

Figure 4: Sample of the data after the scaling.

Then the correlation between the features and the labels was found using pandas.corr. The results shown in Figure4, the results usually range from –1 to 1 where the further the value is from zero the higher correlation there is.

The results were a great indication of this data being suitable for multiple algorithms, where snoring rate (sr) had the best score of 0.975 followed by sleeping hours (sr.1) with a score of –0.973, on the other hand eye movement (rem) had the worst score of 0.95, but it is still considered as a high score thus there is no need to discard any of the features.

```
sl      1.000000
sr      0.975322
lm      0.971071
rr      0.963516
hr      0.963516
rem     0.951988
bo     -0.961092
t      -0.962354
sr.1   -0.973036
```

Figure 5: Correlation values.

As for correlation between the features pandas plotting scatter matrix was used as shown in Figure 6 which shown high correlation between some of the features such as snoring rate (sr) and heart rate (hr), possibly leading to great accuracy when running with various algorithms.

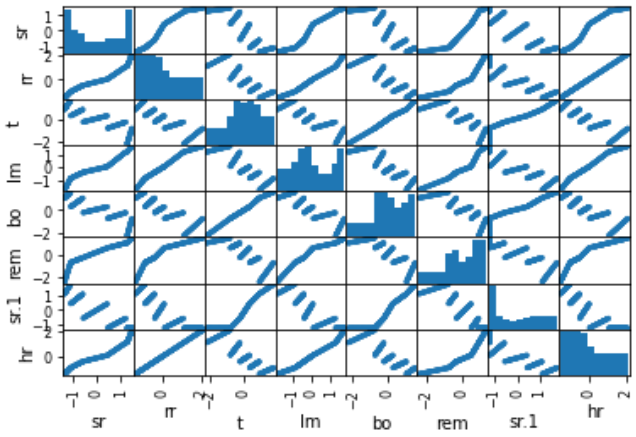


Figure 6: Correlation matrix between features.

An even closer look between correlating features and the expected stress level was done by plotting some of the correlating features as x and y as a scatter plot in addition to having the stress level as a color map ranging from blue (lowest stress level) to red (highest stress level).

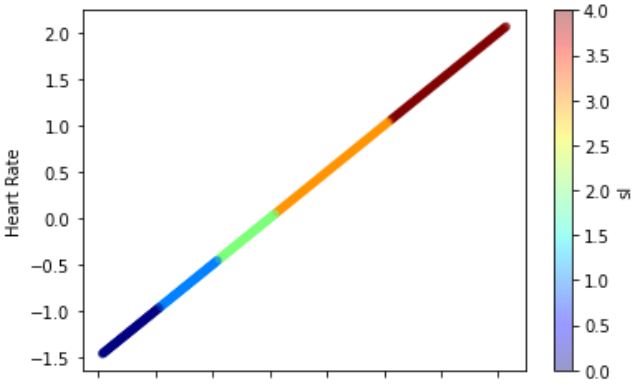


Figure 7: Plot of respiration rate, heart rate and stress level.

Figure7 shows a scatter plot of respiration rate as the x axis, heartrate as the y axis and stress level as the color map, this

plot indicates a high linear correlation with a slope of one approximately between the three values. Which shows that the higher the temperature the higher the heartrate is also the higher the stress level for the next day, all are directly proportional.

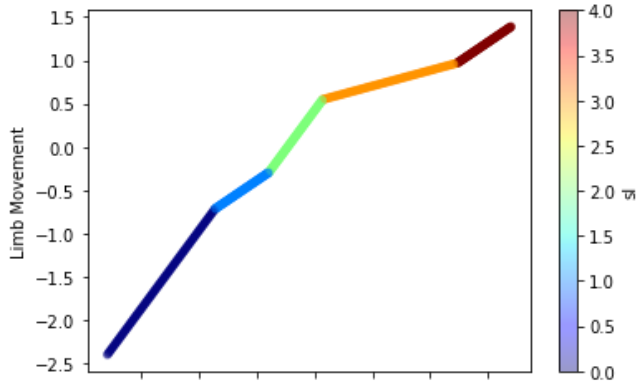


Figure 8: Plot of limb movement, eye movement and stress level.

Figure 8 shows a scatter plot of limb movement as the x axis, eye movement as the y axis and stress level as the color map, the graph is not as smooth as the previous one, but it indicates that the eye and limb movement are directly proportional with each other as well as the stress level.

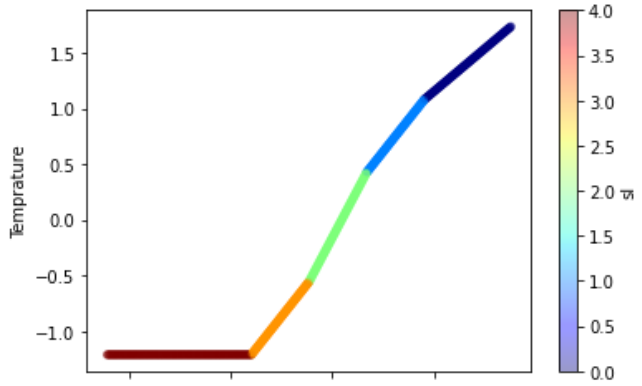


Figure 9: Plot of temperature, sleeping hours and stress level.

Figure 9 shows a scatter plot of temperature as the x axis, sleeping hours as the y axis and stress level as the color map, this plot can be split into two regions: first one is where both sleeping hours and temperature are low which yields into high stress, which contradicts the Figure before where it showed that the temperature is directly proportional with stress level, that is possibly because how much of a factor sleeping hours is so it over rules the temperature values. Sleeping hours is indirectly proportional with the stress level which is in line with the known research regarding this topic.

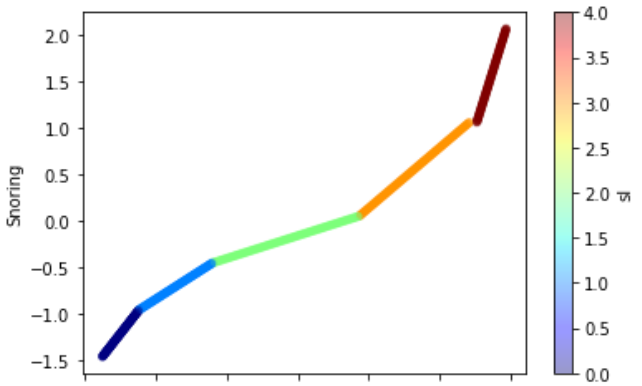


Figure 10: Plot of snoring rate, heartrate and stress level.

Figure 10 shows a scatter plot of the snoring rate as the x axis, heartrate as the y axis and stress level as the color map, this figure indicates a directly proportional relationship between the three values.

V. TRAINING THE MODEL

The data was split into test 80% and train 20%.

A) SGDClassifier was used as a binary classifier where stress of 1 to 4 was classifies as 1 and no stress was classified as 0.

The model was initially trained on defaults parameters, and it yielded a 100% accuracy on the training test, which was worrying at first as it seemed like it had memorized the entire dataset, but this was untrue because it also gave a 100% on the test set which it has never seen before.

Those results are not shocking especially how the correlation was very high in the pre-processing phase.

Even using cross fold validation on five folds it gave a score of 100% for each of the folds.

```
Accuracy on trainset (SGD) = 1.0
Accuracy on testset (SGD) = 1.0
[1. 1. 1. 1. 1.]
```

Figure 11: Accuracy results of SGDClassifier.

B) Multi-class classification using RandomForestClassifier from ScikitLearn, starting with default parameters the accuracy on the training set was 100% again but it was 97.62% on the test set, which is a great result, but higher accuracy can be achieved since the parameters were not tweaked to fit this problem. The accuracy on 5 folds ranged from 98% to 100% with an average of 99%.

```
Accuracy on training (Forest default) =  1.0
Accuracy on test (Forest default) =  0.9761904761904762
Accuracy on five folds (Forest default)
 =  [0.98019802 1.         1.         0.99009901 0.98      ]
CV average = 0.9900594059405939
```

Figure 12: Accuracy of RandomForestClassifier.

As for the confusing matrix as shown in the Figure13 there were three instances mislabeled all were mistaken as the previous or next level of stress which is expected.

```
[[23  0  0  0  0]
 [ 1 22  1  0  0]
 [ 0  0 28  0  0]
 [ 0  0  0 25  1]
 [ 0  0  0  0 25]]
```

Figure 13:Confusion matrix for RandomForest.

Finding the best hyperparameters for this problem was done using GridSearchCV, the best parameters are shown in the Figure 14.

```
{'criterion': 'gini',
 'max_depth': 4,
 'max_features': 'auto',
 'n_estimators': 200}
```

Figure 14: Best parameters for RandomForest.

Retraining the model gave the same results on the training and test sets 100% and 97.62% respectively, leading to no improvement compared with the default parameters, but it has shown improvement on the cross fold which ranged from 98% to 100% with an average of 99.2% which is a 0.2% improvement.

```
Accuracy on training(Forest best) =  1.0
Accuracy on test (Forest best)=  0.9761904761904762
Accuracy on five folds (Forest best) = [0.98019802 1.         1.         0.99009901 0.98      ]
CV average = 0.9920396039603961
```

Figure 15: Results of RandomForest with best parameters.

C) KNeighborsClassifier:

The best parameters were found using GridSearchCV as shown in Figure 16.

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits

{'n_neighbors': 1, 'weights': 'uniform'}
```

Figure 16:Best parameters for KNeighbors.

Training the model with the parameters found yielded great results with a 100% accuracy for both test and training sets, and an average of 99% for the five folds which can still be improved, Figure 17.

```
Accuracy on training (KN) =  1.0
Accuracy on test (KN) =  1.0
Accuracy on five folds (KN) = [0.98019802 1.         1.         0.99009901 0.9
CV average = 0.9900594059405939
```

Figure 17: Accuracy of KNeighbors.

This gave a perfect correlation matrix where the values are only on the main diagonal, Figure18.

```
[[23  0  0  0  0]
 [ 0 24  0  0  0]
 [ 0  0 28  0  0]
 [ 0  0  0 26  0]
 [ 0  0  0  0 25]]
```

Figure 18: KNeighbors correlation matrix.

D) DECISIONTREECLASSIFIER

The best parameters were found using GridSearchCV as shown in Figure 19.

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits

{'ccp_alpha': 0.001,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 'auto'}
```

Figure 19: DecisionTree best parameters.

The model with the best parameters yielded a 100% accuracy on the training set and 97.62% on the test set which are identical results to the RandomForestClassifier, both even had the same correlation matrix. But the cross-fold average was 98.8% which is worse than the RandomForestClassifier, Figure 20.

```
Accuracy on training (DecTree)=  1.0
Accuracy on test (DecTree) =  0.9761904761904762
Accuracy on five folds (DecTree) = [0.98019802 1.         1.         0.99009901 0.97      ]
CV average = 0.988059405940594
[[23  0  0  0  0]
 [ 1 22  1  0  0]
 [ 0  0 28  0  0]
 [ 0  0  0 25  1]
 [ 0  0  0  0 25]]
```

Figure 20: Results of DesicionTree.

E) LogisticRegression

Best parameters were once again found using GridSearchCV as shown in Figure 21.

```
{'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
```

Figure 21: LogisticRegression best parameters.

This model yielded perfect results with a 100% accuracy for the test, training and even the five folds, which was proven to be the best algorithm for this problem, Figure 22.

```
Accuracy on training (LogReg) =  1.0
Accuracy on test (LogReg) =  1.0
Accuracy on five folds (LogReg) = [1. 1. 1. 1. 1.]
CV average = 1.0
[[23  0  0  0  0]
 [ 0 24  0  0  0]
 [ 0  0 28  0  0]
 [ 0  0  0 26  0]
 [ 0  0  0  0 25]]
```

Figure 22: Results of LogisticRegression.

VI. CONCLUSION

The data was prepared for training after taking a good look at the features and correlations.

At first a binary classifier was used to have a better idea on what to expect from the results, then multiple algorithms were used.

All the algorithms yielded accurate results, where both RandomForest and DecisionTree gave an accuracy of

97.62% on the test set, but RandomForest was marginally better with the K-folds by a 0.2% difference. Also, both of KNeighbors and LogicticRegression yielded a 100% accuracy on the test set, but LogisticRegression excelled with the K-folds giving a 100% accuracy compared to a 99% for KNeighbors.

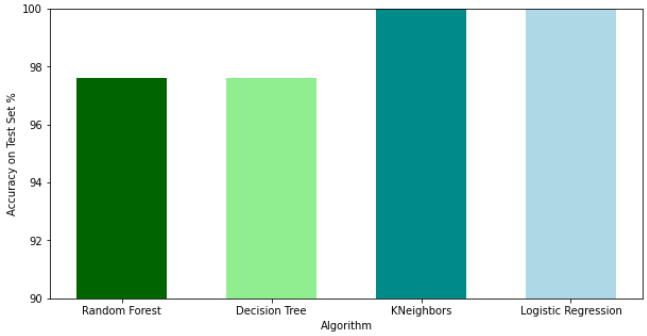


Figure 23: Bar plot of algorithms' accuracy on the test set.

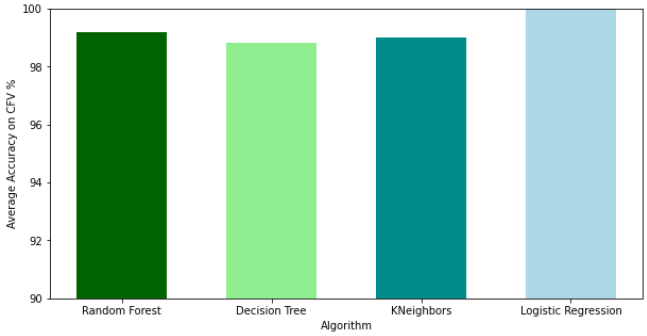


Figure 24: Bar plot of algorithms' accuracy on the K-Folds.

As for future work, the dataset could be more thorough and include multiple labels which could be helpful predicting other types of mental upsets such as panic attacks and depressive episodes.

It would also help adding more features which could improve the accuracy for more labels.

REFERENCES

[1] American psychological association (APA).

[2] Chi JS, Kloner RA. Stress and myocardial infarction. Heart. 2003;89(5):475–476. doi:10.1136/heart.89.5.475.

[3] L. Rachakonda, A. K. Bapatla, S. P. Mohanty and E. Kougianos, "SaYoPillow: Blockchain-Integrated Privacy-Assured IoMT Framework for Stress Management Considering Sleeping Habits," in IEEE Transactions on Consumer Electronics, vol. 67, no. 1, pp. 20-29, Feb. 2021, doi: 10.1109/TCE.2020.3043683.

[4] E.-J. Kim and J. E. Dimsdale, "The Effect of Psychosocial Stress on Sleep: A Review of Polysomnographic Evidence," Beh. sleep med., 2007.

In [46]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

data=pd.read_csv('/home/sdsrtnn/Studies/University/CurrentSem/AI/Project/Data/sleep.csv')
print(data.head())
print(data.info())
```

```
      sr      rr       t      lm      bo    rem   sr.1     hr  sl
0  93.80  25.680  91.840  16.600  89.840  99.60  1.840  74.20   3
1  91.64  25.104  91.552  15.880  89.552  98.88  1.552  72.76   3
2  60.00  20.000  96.000  10.000  95.000  85.00  7.000  60.00   1
3  85.76  23.536  90.768  13.920  88.768  96.92  0.768  68.84   3
4  48.12  17.248  97.872   6.496  96.248  72.48  8.248  53.12   0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 630 entries, 0 to 629
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   sr      630 non-null    float64
 1   rr      630 non-null    float64
 2   t       630 non-null    float64
 3   lm      630 non-null    float64
 4   bo      630 non-null    float64
 5   rem     630 non-null    float64
 6   sr.1    630 non-null    float64
 7   hr      630 non-null    float64
 8   sl      630 non-null    int64  
dtypes: float64(8), int64(1)
memory usage: 44.4 KB
None
```

In [47]:

```
# scaling the features dataa is same as data but feats are scaled
labels=data['sl']
data.drop('sl',axis=1,inplace=True)
scaler=StandardScaler()
scaled=scaler.fit_transform(data)
scaled=pd.DataFrame(scaled,index=data.index,columns=data.columns)

dataa=scaled.copy()
dataa['sl']=labels
dataa.head()
```

Out[47]:

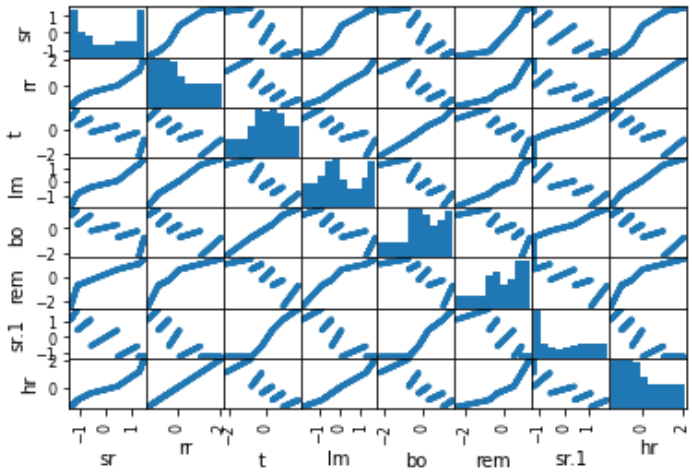
	sr	rr	t	lm	bo	rem	sr.1	hr	sl
0	1.146845	0.979066	-0.272195	1.140539	-0.271838	0.934005	-0.609407	0.979066	3
1	1.035260	0.833720	-0.353853	0.972949	-0.345696	0.873421	-0.703767	0.833720	3
2	-0.599252	-0.454206	0.907316	-0.395697	1.051448	-0.294506	1.081206	-0.454206	1
3	0.731501	0.438056	-0.576145	0.516734	-0.546753	0.708498	-0.960635	0.438056	3
4	-1.212970	-1.148636	1.438095	-1.211299	1.371498	-1.347997	1.490099	-1.148636	0

In [107]:

```
corr_matrix=dataa.corr()
print(corr_matrix["sl"].sort_values(ascending=False))
```

```
pd.plotting.scatter_matrix(scaled)
plt.show()
```

```
sl      1.000000
sr      0.975322
lm      0.971071
rr      0.963516
hr      0.963516
rem     0.951988
bo     -0.961092
t      -0.962354
sr.1   -0.973036
Name: sl, dtype: float64
```

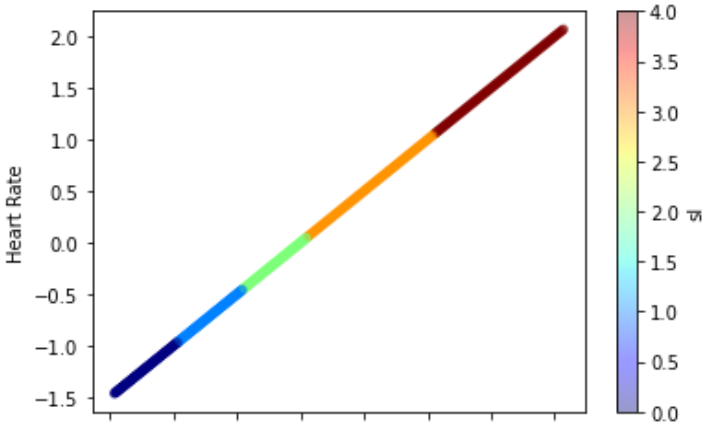


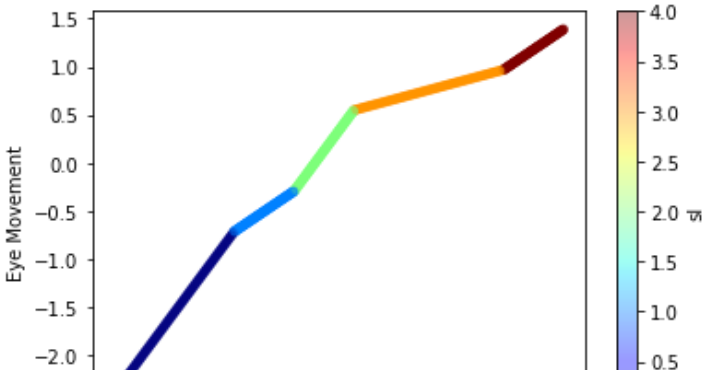
In [108]:

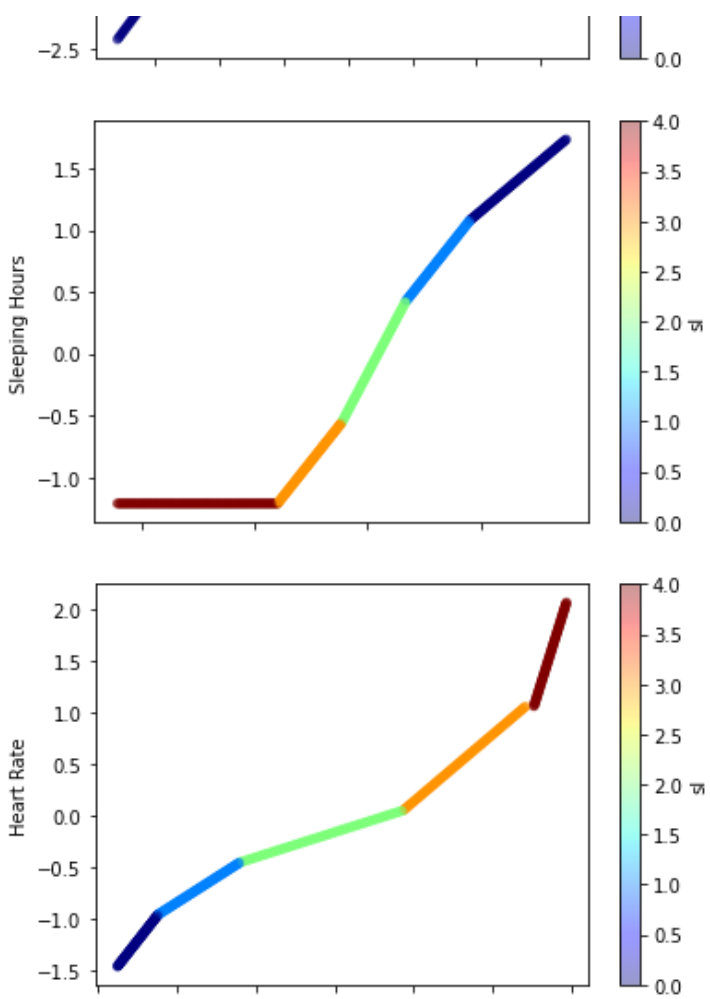
```
dataa.plot(kind="scatter",x='rr',y="hr",alpha=0.4,c="sl",cmap=plt.get_cmap("jet"),colorbar=True,ylabel="Heart Rate",xlabel="Resperation Rate",)
dataa.plot(kind="scatter",x='lm',y="rem",alpha=0.4,c="sl",cmap=plt.get_cmap("jet"),colorbar=True,xlabel="Limb Movement",ylabel="EyeMovement")
dataa.plot(kind="scatter",x='t',y="sr.1",alpha=0.4,c="sl",cmap=plt.get_cmap("jet"),colorbar=True,xlabel="Temprature",ylabel="Sleeping Hours")
dataa.plot(kind="scatter",x='sr',y="hr",alpha=0.4,c="sl",cmap=plt.get_cmap("jet"),colorbar=True,xlabel="Snoring",ylabel="Heart Rate")
```

Out[108]:

```
<AxesSubplot:xlabel='Snoring', ylabel='Heart Rate'>
```







In [111]:

```
from sklearn.linear_model import SGDClassifier


bin_cls=SGDClassifier(random_state=6)


binaryl=labels.replace([1,2,3,4],1)
X_trainb, X_testb, y_trainb, y_testb = train_test_split(
 scaled, binaryl, test_size=0.2, random_state=42)

bin_cls.fit(X_trainb,y_trainb)

train_pred=bin_cls.predict(X_trainb)
bin_acc_train=accuracy_score(y_trainb,train_pred)
print("Accuracy on trainset (SGD) =" ,bin_acc_train)
predictions_b=bin_cls.predict(X_testb)
bin_acc_test=accuracy_score(y_testb,predictions_b)
print("Accuracy on testset (SGD) =",bin_acc_test)
cv_sb=cross_val_score(bin_cls,X_trainb,y_trainb,scoring="accuracy",cv=5)
print(cv_sb)
```

```
Accuracy on trainset (SGD) = 1.0
Accuracy on testset (SGD) = 1.0
[1. 1. 1. 1. 1.]
```

In [51]:

```
X_train, X_test, y_train, y_test = train_test_split(
 scaled, labels, test_size=0.2, random_state=42)
```

In [123]:

```
from sklearn.ensemble import RandomForestClassifier
```

```
forest=RandomForestClassifier(random_state=6)
forest.fit(X_train,y_train)
forest_trainpred=forest.predict(X_train)
forest_acc_train=accuracy_score(y_train,forest_trainpred)
print("Accuracy on training (Forest default) = ",forest_acc_train)
forest_testpred=forest.predict(X_test)
forest_acc_test=accuracy_score(y_test,forest_testpred)
print("Accuracy on test (Forest default) = ",forest_acc_test)
cv_s=cross_val_score(forest,X_train,y_train,scoring="accuracy",cv=5)
print("Accuracy on five folds (Forest default) \n = ",cv_s)
print("CV average =",np.average(cv_s))
conf_forest=confusion_matrix(y_test,forest_testpred)
print(conf_forest)
```

```
Accuracy on training (Forest default) =  1.0
Accuracy on test (Forest default) =  0.9761904761904762
Accuracy on five folds (Forest default)
 =  [0.98019802 1.         1.         0.99009901 0.98      ]
CV average = 0.9900594059405939
[[23  0  0  0  0]
 [ 1 22  1  0  0]
 [ 0  0 28  0  0]
 [ 0  0  0 25  1]
 [ 0  0  0  0 25]]
```

In [42]:

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
CV_rfc = GridSearchCV(estimator=forest, param_grid=param_grid,cv=5)
CV_rfc.fit(X_train, y_train)
CV_rfc.best_params_
```

Out[42]:

```
{'criterion': 'gini',
 'max_depth': 4,
 'max_features': 'auto',
 'n_estimators': 200}
```

In [122]:

```
forestb=RandomForestClassifier(random_state=6,criterion= 'gini',
 max_depth= 4,
 max_features= 'auto',
 n_estimators= 200)
forestb.fit(X_train,y_train)
forestb_trainpred=forestb.predict(X_train)
forestb_acc_train=accuracy_score(y_train,forestb_trainpred)
print("Accuracy on training(Forest best) = ",forestb_acc_train)
forestb_testpred=forestb.predict(X_test)
forestb_acc_test=accuracy_score(y_test,forestb_testpred)
print("Accuracy on test (Forest best)= ",forestb_acc_test)
cv_sb=cross_val_score(forestb,X_train,y_train,scoring="accuracy",cv=5)
print("Accuracy on five folds (Forest best) =",cv_sb)
print("CV average =",np.average(cv_sb))
conf_forestb=confusion_matrix(y_test,forestb_testpred)
print(conf_forestb)
```

```
Accuracy on training(Forest best) =  1.0
Accuracy on test (Forest best)=  0.9761904761904762
Accuracy on five folds (Forest best) = [0.98019802 1.         1.         0.99009901 0.98
]
CV average = 0.9920396039603961
[[23  0  0  0  0]
 [ 1 22  1  0  0]
 [ 0  0 28  0  0]
 [ 0  0  0 25  1]
```

```
 [ 0  0  0  0 25]]
```

In [126]:

```
from sklearn.neighbors import KNeighborsClassifier
nei=KNeighborsClassifier()
k_range = list(range(1, 31))
weight_options = ["uniform", "distance"]
param_grid = dict(n_neighbors = k_range, weights = weight_options)

# defining parameter range
grid = GridSearchCV(nei, param_grid, cv=5, scoring='accuracy', return_train_score=False,
verbose=1)

# fitting the model for grid search
grid_search=grid.fit(X_train, y_train)
grid.best_params_
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

Out[126]:

```
{'n_neighbors': 1, 'weights': 'uniform'}
```

In [127]:

```
nei=KNeighborsClassifier(n_neighbors=1)
nei.fit(X_train,y_train)
nei_trainpred=nei.predict(X_train)
nei_acc_train=accuracy_score(y_train,nei_trainpred)
print("Accuracy on training (KN) = ",nei_acc_train)
nei_testpred=nei.predict(X_test)
nei_acc_test=accuracy_score(y_test,nei_testpred)
print("Accuracy on test (KN) = ",nei_acc_test)
cv_sk=cross_val_score(forest,X_train,y_train,scoring="accuracy",cv=5)
print("Accuracy on five folds (KN) =",cv_sk)
print("CV average =",np.average(cv_sk))
conf_nei=confusion_matrix(y_test,nei_testpred)
print(conf_nei)
```

```
Accuracy on training (KN) =  1.0
Accuracy on test (KN) =  1.0
Accuracy on five folds (KN) = [0.98019802 1.         1.         0.99009901 0.98      ]
CV average = 0.9900594059405939
[[23  0  0  0  0]
 [ 0 24  0  0  0]
 [ 0  0 28  0  0]
 [ 0  0  0 26  0]
 [ 0  0  0  0 25]]
```

In [45]:

```
from sklearn.tree import DecisionTreeClassifier
tree=DecisionTreeClassifier()
param_grid = {'max_features': ['auto', 'sqrt', 'log2'],
              'ccp_alpha': [0.1, .01, .001],
              'max_depth' : [5, 6, 7, 8, 9],
              'criterion' :['gini', 'entropy']
             }
tree_clas = DecisionTreeClassifier(random_state=1024)
grid_search = GridSearchCV(estimator=tree, param_grid=param_grid, cv=5, verbose=True)
grid_search.fit(X_train, y_train)
grid_search.best_params_
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
```

Out[45]:

```
{'ccp_alpha': 0.001,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 'auto'}
```

In [130]:

```
tree=DecisionTreeClassifier(ccp_alpha= 0.001, criterion= 'gini', max_depth= 8, max_featu
res= 'sqrt')
tree.fit(X_train,y_train)
tree_trainpred=tree.predict(X_train)
tree_acc_train=accuracy_score(y_train,tree_trainpred)
print("Accuracy on training (DecTree)= ",tree_acc_train)
tree_testpred=tree.predict(X_test)
tree_acc_test=accuracy_score(y_test,tree_testpred)
print("Accuracy on test (DecTree) = ",tree_acc_test)
cv_t=cross_val_score(tree,X_train,y_train,scoring="accuracy",cv=5)

print("Accuracy on five folds (DecTree) =",cv_t)
print("CV average =",np.average(cv_t))

conf_tree=confusion_matrix(y_test,tree_testpred)
print(conf_tree)
```

```
Accuracy on training (DecTree)=  1.0
Accuracy on test (DecTree) =  0.9761904761904762
Accuracy on five folds (DecTree) = [0.98019802 1.         1.         0.99009901 0.97
]
CV average = 0.988059405940594
[[23  0  0  0  0]
 [ 1 22  1  0  0]
 [ 0  0 28  0  0]
 [ 0  0  0 25  1]
 [ 0  0  0  0 25]]
```

In [55]:

```
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore')
lr = LogisticRegression(random_state=6)
parameters = {
    'penalty' : ['l1','l2'],
    'C'       : np.logspace(-3,3,7),
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
}
clfa = GridSearchCV(lr,
                    param_grid = parameters,
                    scoring='accuracy',
                    cv=5)
clfa.fit(X_train,y_train)
clfa.best_params_
```

Out[55]:

```
{'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
```

In [132]:

```
lr = LogisticRegression(C= 0.1, penalty= 'l2', solver= 'newton-cg')
lr.fit(X_train,y_train)
lr_trainpred=lr.predict(X_train)
lr_acc_train=accuracy_score(y_train,lr_trainpred)
print("Accuracy on training (LogReg) = ",lr_acc_train)
lr_testpred=lr.predict(X_test)
lr_acc_test=accuracy_score(y_test,lr_testpred)
print("Accuracy on test (LogReg) = ",lr_acc_test)
cv_lr=cross_val_score(lr,X_train,y_train,scoring="accuracy",cv=5)
print("Accuracy on five folds (LogReg) =",cv_lr)
print("CV average =",np.average(cv_lr))
conf_lr=confusion_matrix(y_test,lr_testpred)
print(conf_lr)
```

```
Accuracy on training (LogReg) =  1.0
Accuracy on test (LogReg) =  1.0
Accuracy on five folds (LogReg) = [1. 1. 1. 1. 1.]
CV average = 1.0
```

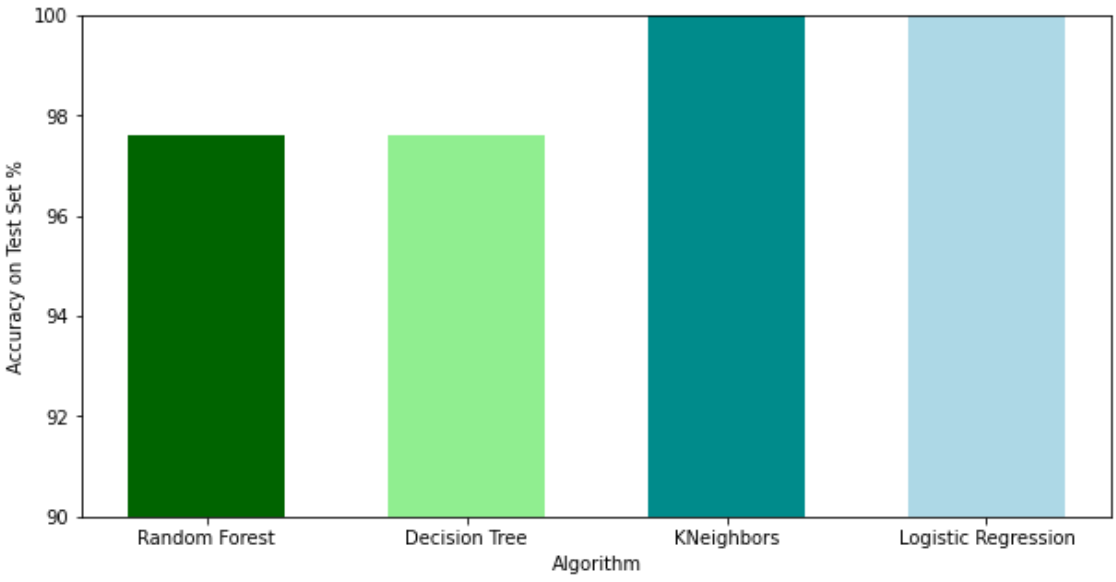
```
[[23  0  0  0  0]
 [ 0 24  0  0  0]
 [ 0  0 28  0  0]
 [ 0  0  0 26  0]
 [ 0  0  0  0 25]]
```

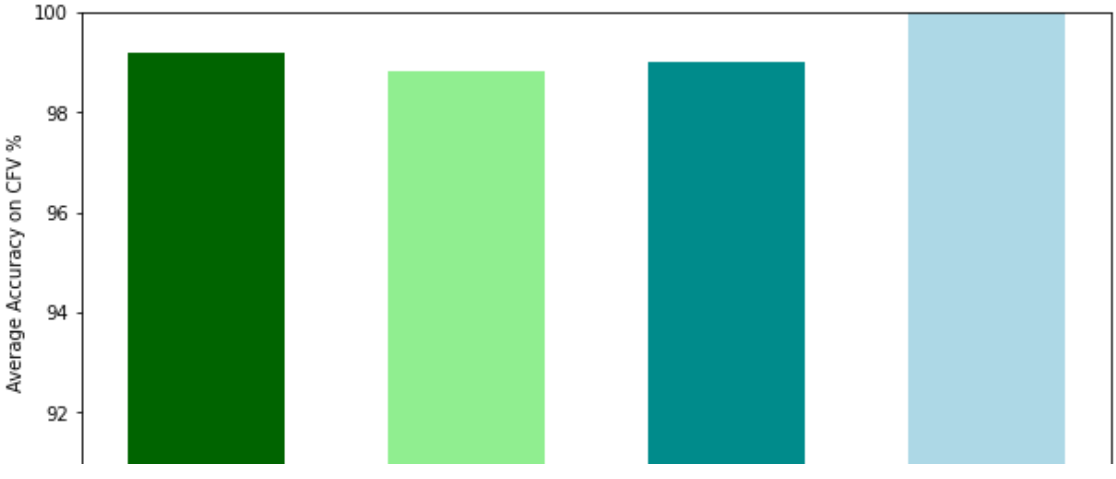
In [135]:

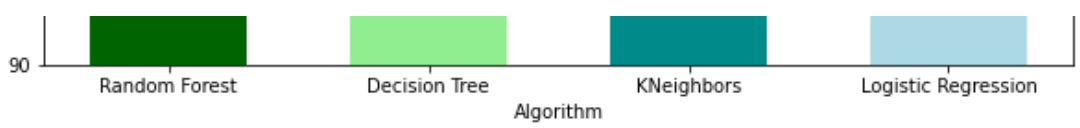
```
train_acc=np.array([bin_acc_train,forest_acc_train,tree_acc_train,nei_acc_train,lr_acc_train])

test_acc=np.array([forest_acc_test,tree_acc_test,nei_acc_test,lr_acc_test])
cv_acc=np.array([np.average(cv_sb),np.average(cv_t),np.average(cv_sk),np.average(cv_lr)])*100
test_acc=test_acc*100
print(train_acc)
print(test_acc)
fig = plt.figure(figsize = (10, 5))
plt.ylim(90,100)
plt.ylabel("Accuracy on Test Set %")
plt.xlabel("Algorithm")
plt.bar(height=test_acc,x=["Random Forest","Decision Tree","KNeighbors","Logistic Regression"], color =["darkgreen","lightgreen","darkcyan","lightblue"],
        width = 0.6)
plt.show()
fig = plt.figure(figsize = (10, 5))
plt.ylim(90,100)
plt.ylabel("Average Accuracy on CFV %")
plt.xlabel("Algorithm")
plt.bar(height=cv_acc,x=["Random Forest","Decision Tree","KNeighbors","Logistic Regression"], color =["darkgreen","lightgreen","darkcyan","lightblue"],
        width = 0.6)
plt.show()
```

```
[1. 1. 1. 1. 1.]
[ 97.61904762  97.61904762 100.         100.        ]
```






90
Random Forest
Decision Tree
KNeighbors
Logistic Regression
Algorithm