Data Diary

Julia

A student who is curious about how different factors, such as her nutrition, exercise, and sleep, would impact her overall well-being and performance. She is always looking for ways to optimize her life and make informed decisions about her health and happiness.

Data Description

A dataset from Kaggle has been taken which was originally posted by Korean Sports Promotion Foundation. It is comprised of 365 instances and 11 attributes.

```
df.keys()
```

```
Index(['date', 'age', 'gender', 'height', 'weight', 'steps', 'hear_rate',
       'calories', 'distance', 'steps_times_distance', 'activity'],
      dtype='object')
```

Exploratory Data Analysis

head() displays the top 5 observations of the dataset.

```
df.head()
```

	date	age	gender	height	weight	steps	hear_rate	calories	distance	steps_times_distance	activity
0	2019-01-01	20	1	168	65.4	10.771429	78.531302	0.344533	0.008327	0.089692	Lying
1	2019-01-02	20	1	168	65.4	11.475325	78.453390	3.287625	0.008896	0.102088	Lying
2	2019-01-03	20	1	168	65.4	12.179221	78.540825	9.484000	0.009466	0.115287	Lying
3	2019-01-04	20	1	168	65.4	12.883117	78.628260	10.154556	0.010035	0.129286	Lying
4	2019-01-05	20	1	168	65.4	13.587013	78.715695	10.825111	0.010605	0.144088	Lying

tail() displays the last 5 observations of the dataset.

```
df.tail()
```

	date	age	gender	height	weight	steps	hear_rate	calories	distance	steps_times_distance	activity
360	2019-12-27	19	0	172	72.5	12.519481	78.625042	0.470516	0.009457	0.118399	Running 7 METs
361	2019-12-28	23	1	181	95.2	10.771429	78.531302	0.344533	0.008327	0.089692	Lying
362	2019-12-29	23	1	181	95.2	12.574603	78.453390	3.287625	0.009872	0.124132	Lying
363	2019-12-30	23	1	181	95.2	14.377778	78.154238	6.465719	0.011416	0.164143	Lying
364	2019-12-31	23	1	181	95.2	16.180952	77.855085	9.643813	0.012961	0.209725	Lying

shape displays the number of observations(rows) and features(columns) in the dataset. There are 13393 observations and 12 variables in our dataset.

```
[21] df.shape

     (365, 11)
```

Preprocessing

info() helps to understand the data type and information about data, including the number of records in each column, data having null or not null, Data type, the memory usage of the dataset.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   date                  365 non-null    object
 1   age                   365 non-null    int64
 2   gender                365 non-null    int64
 3   height                365 non-null    int64
 4   weight                365 non-null    float64
 5   steps                 365 non-null    float64
 6   hear_rate             365 non-null    float64
 7   calories              365 non-null    float64
 8   distance              365 non-null    float64
 9   steps_times_distance  365 non-null    float64
 10  activity              365 non-null    object
dtypes: float64(6), int64(3), object(2)
memory usage: 31.5+ KB
```

Missing Values

It can be seen that there are no missing values

```
df.isnull().sum()
```

```
date                    0
age                     0
gender                  0
height                  0
weight                  0
steps                   0
hear_rate               0
calories                0
distance                0
steps_times_distance    0
activity                0
dtype: int64
```

Categorical Variables

There are two variables with categories

```
[3] df['activity'].unique()
```

```
array(['Lying', 'Sitting', 'Self Pace walk', 'Running 3 METs',
       'Running 5 METs', 'Running 7 METs'], dtype=object)
```

Let's covert them into numeric

Replacing categorical classes to numerical

```
[6] df['activity'].replace(['Lying', 'Sitting', 'Self Pace walk',
                            'Running 3 METs','Running 5 METs',
                            'Running 7 METs'],[1,2,3,4,5,6], inplace=True)
```

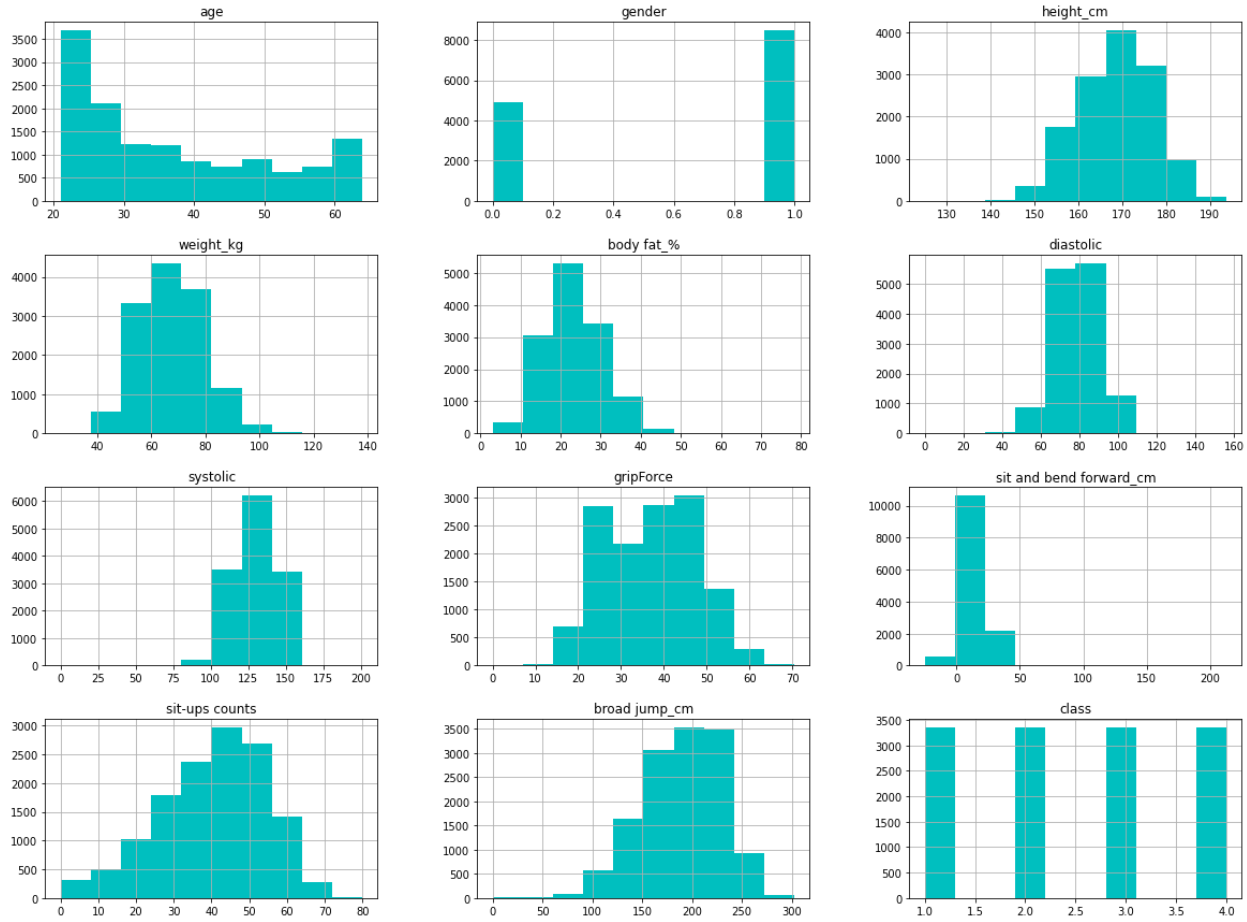
Feature Engineering

Feature Engineering for Date

```
[4]  df['date'] = pd.to_datetime(df['date'])
     df['year'] = df['date'].dt.year
     df['month'] = df['date'].dt.month
     df['day'] = df['date'].dt.day


     df['day_of_week'] = df['date'].dt.dayofweek
```

Data Visualization

```
[14] import matplotlib.pyplot as plt
     import seaborn as sns
     import numpy as np
     df[df.columns].hist(color="c",figsize=(20,15))
     plt.show()
```



Data Normalization

```
[14] from sklearn.preprocessing import MinMaxScaler

     scaler = MinMaxScaler()

     normalized_data = scaler.fit_transform(df)

     print(normalized_data)
```

```
[[0.13953488 1.         0.6875     ... 0.75       0.71617162 0.66666667]
 [0.09302326 1.         0.58139535 ... 0.6625     0.75577558 0.        ]
 [0.23255814 1.         0.79360465 ... 0.6125     0.59735974 0.66666667]
 ...
 [0.41860465 1.         0.75872093 ... 0.5625     0.75577558 0.        ]
 [1.         0.         0.30668605 ... 0.         0.24752475 1.        ]
 [0.30232558 1.         0.56686047 ... 0.6375     0.59405941 0.66666667]]
```

Models used:

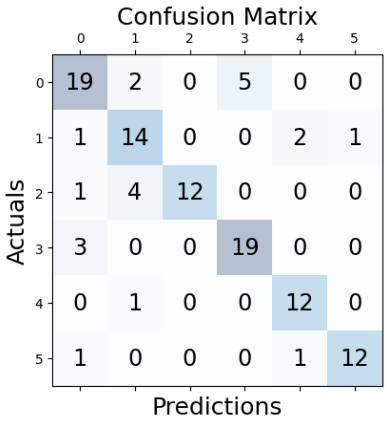
AdaBoost Classifier

```
[9] from sklearn.ensemble import AdaBoostClassifier
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
    from sklearn.tree import DecisionTreeClassifier
    import matplotlib.pyplot as plt
    base_estimator = DecisionTreeClassifier(max_depth=8)
    adaboost = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50, random_state=42)
    adaboost.fit(x_train, y_train)
    pred1 = adaboost.predict(x_test)
    accuracy = accuracy_score(y_test, pred1)
    print('Accuracy:', accuracy)

    conf_matrix = confusion_matrix(y_true=y_test, y_pred=pred1)
    fig, ax = plt.subplots(figsize=(4.5, 4.5))
    ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

    plt.xlabel('Predictions', fontsize=18)
    plt.ylabel('Actuals', fontsize=18)
    plt.title('Confusion Matrix', fontsize=18)
    plt.show()
```

```
Accuracy Obtained using Ada Boost: 0.8909090909090909
```

AdaBoost was used on a fitness dataset, and it achieved an accuracy of 0.890, which is a good accuracy score. This means that AdaBoost was able to effectively combine multiple weak learners to

produce a strong classifier that could accurately classify fitness-related data. This accuracy score indicates that the model is performing well and is likely to be useful in predicting outcomes or making recommendations based on the data.

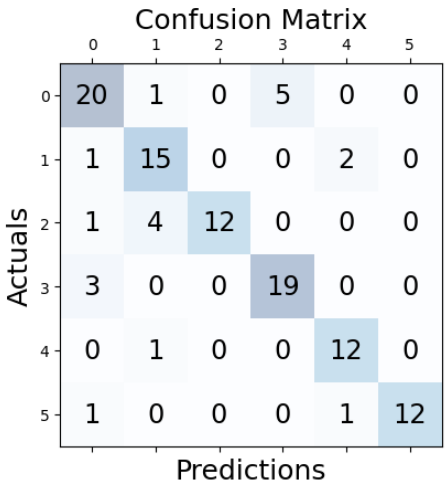


Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(x_train, y_train)
pred2 = rf_model.predict(x_test)
accuracy = rf_model.score(x_test, y_test)
print(f"Accuracy: {accuracy}")
conf_matrix = confusion_matrix(y_true=y_test, y_pred=pred2)
fig, ax = plt.subplots(figsize=(4.5, 4.5))
ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```

```
Accuracy: 0.8454545454545455
```

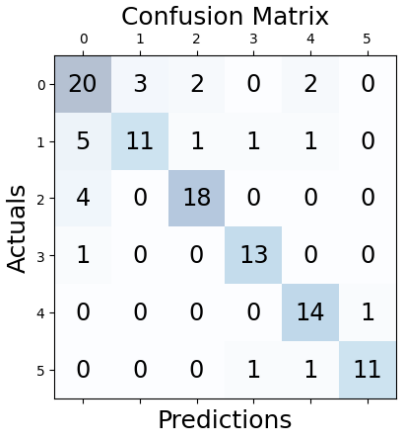
The accuracy of 0.845 on a fitness dataset means that the Random Forest algorithm correctly predicted the outcome of 84.5% of the instances in the dataset. This indicates that the algorithm is performing well on the dataset and can be considered a reliable model.



Gradient Boosting Classifier

```
[45] from sklearn.ensemble import GradientBoostingClassifier
     import pandas as pd
     gb_model = GradientBoostingClassifier(n_estimators=50, random_state=42)
     gb_model.fit(x_train, y_train)
     pred3 = gb_model.predict(x_test)
     accuracy = gb_model.score(x_test, y_test)
     print(f"Accuracy: {accuracy}")


     conf_matrix = confusion_matrix(y_true=y_test, y_pred=pred3)
     fig, ax = plt.subplots(figsize=(4.5, 4.5))
     ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
     for i in range(conf_matrix.shape[0]):
         for j in range(conf_matrix.shape[1]):
             ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

     plt.xlabel('Predictions', fontsize=18)
     plt.ylabel('Actuals', fontsize=18)
     plt.title('Confusion Matrix', fontsize=18)
     plt.show()

     Accuracy: 0.7909090909090909
```

The Gradient Boosting algorithm was trained on this dataset, and it achieved an accuracy of 0.790. This means that the model correctly predicted the outcome of 79% of the instances in the dataset. The accuracy score indicates how well the model performed in terms of correctly classifying the data.

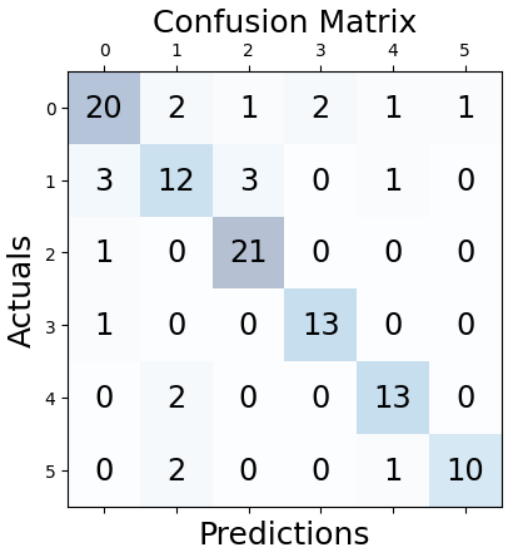


Bagging Classifier

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(random_state=42,max_depth=10)
bagging = BaggingClassifier(base_estimator=tree, n_estimators=50, random_state=42)
bagging.fit(x_train, y_train)
pred4 = bagging.predict(x_test)
accuracy = accuracy_score(y_test, pred4)
print(f"Accuracy: {accuracy}")
conf_matrix = confusion_matrix(y_true=y_test, y_pred=pred4)
fig, ax = plt.subplots(figsize=(4.5, 4.5))
ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning:
  warnings.warn(
Accuracy: 0.8090909090909091
```

A Bagging classifier is an ensemble learning technique that combines multiple base classifiers to improve the overall performance of the model. The Bagging classifier produced an accuracy of 0.809 on this dataset. This means that the classifier correctly classified approximately 81% of the instances in the test set. The accuracy of the classifier indicates the proportion of correctly classified instances out of the total number of instances in the test set.



KNN Classifier

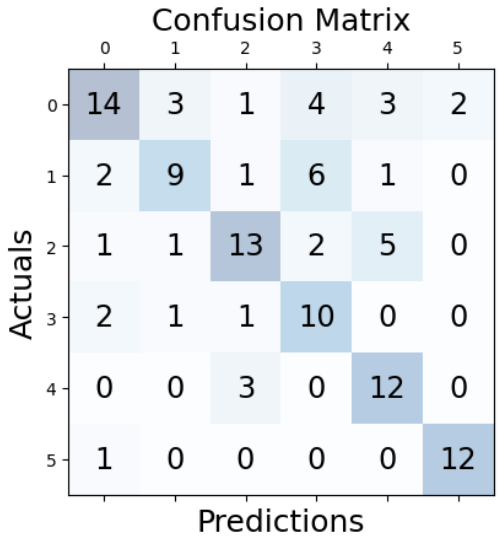
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_train, y_train)
pred5 = knn.predict(x_test)
accuracy = accuracy_score(y_test, pred5)
print(f"Accuracy: {accuracy}")
conf_matrix = confusion_matrix(y_true=y_test, y_pred=pred5)
fig, ax = plt.subplots(figsize=(4.5, 4.5))
ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```

```
Accuracy: 0.6363636363636364
```

K-Nearest Neighbors (KNN) is a classification algorithm that is commonly used in machine learning to classify new data points based on their proximity to existing data points in a training dataset. The KNN classifier produced an accuracy of 0.636, which means that it correctly classified 63.6% of the data points in the dataset. This accuracy score suggests that the classifier may not be performing very well on this dataset, and there may be room for improvement.

To improve the accuracy of the KNN classifier, one approach could be to try different values of K and see which one produces the best results. Another approach could be to preprocess the data to remove any outliers or noise that may be affecting the accuracy of the classifier. Additionally, using

feature selection techniques to identify the most important features for classification may also help to improve the accuracy of the classifier.




```
import pandas as pd
accuracy_values = [0.890, 0.854, 0.790, 0.809, 0.636]
data = {'Model': ['AdaBoost Classifier', 'Random Forest Classifier', 'Gradient Boosting Classifier', 'Bagging Classifier', 'KNeighbors Classifier'],
        'Accuracy': accuracy_values}
df = pd.DataFrame(data)

df
```

	Model	Accuracy
0	AdaBoost Classifier	0.890
1	Random Forest Classifier	0.854
2	Gradient Boosting Classifier	0.790
3	Bagging Classifier	0.809
4	KNeighbors Classifier	0.636

Conclusion:

Based on the above analysis, it can be concluded that Adaboost and Random Forest classifiers have performed the best with accuracy scores of 0.890 and 0.845 respectively. These models have been able to correctly classify the data with high accuracy rates, indicating their efficiency in predicting the fitness levels based on the input features.

On the other hand, Gradient Boost and Bagging Classifier models have also demonstrated decent performance with accuracy scores of 0.790 and 0.809 respectively. Although their accuracy rates are lower compared to Adaboost and Random Forest, they still show promise as effective classifiers.

However, the K-Nearest Neighbor (KNN) model has produced the lowest accuracy score of 0.636, indicating its inefficiency in classifying the fitness dataset. The KNN model might not be the best choice for this specific dataset, and it is recommended to explore other models that may perform better.

Overall, the Adaboost and Random Forest models appear to be the most effective in predicting the fitness levels in this dataset, while the KNN model needs further improvement.