CAPSTONE PROJECT

Body Fat Prediction

Date: 08-11-2021

Contributor: Luka Anicin

Author: Ruslan S.

<u>Introduction</u>: around two-thirds of adults in the US are facing health conditions related to obesity, based on the credited resources in 2020. Obesity is a fight that is extremely nasty and finding the solution to control is very crucial. But what is the best way to control it, we should ask ourselves? The most convenient one today is the knowledge of knowing the percentage of fat in your body. So, once you know it, you can balance it. Unfortunately, there are not many suitable or simple ways to do it. That is why this project is targeting specifically those types of people who have no resources or no time. It's about giving them hope, a proper opportunity, and an option to fight this battle with appropriate instruments (in our case it will be software application). This application should be able to predict body fat with an accuracy of 80-95% with a simple step of measuring your body components. Importantly, it will help to reduce the financial loss (based on the regular medical procedures) on average and time (spent on transportation, documentation/application parts, waiting results, and other required things).

<u>Timing</u>: the full functional application is expected to be in production by the end of 2022.

<u>Objective</u>: possible ways for obese people to identify their percentage of body fat, with approximately result accuracy of 70-88%, by avoiding the expensive and complicated procedures, through a few simple steps of measuring their body components and using a well-developed software application by the end of June 2022.

<u>Success</u>: the software application is fully functional by the end of 2021. It predicts the percentage of body fat with an accuracy of 70-88%. It requires only the measurement of body components for its successful work (result).

Solution space: the focus will be spent on optimizing the accuracy of the working model (aka application). It will include the reduction of the possibility of occurring the resulting errors based on the provided data within a specified scope of features (body measurement components). Additionally, the center of work will be directed to the process of finding the most accurate (less computational expensive) equations of finding the percentage of body fat and other feature (variable) ratios.

Complications: high possibility to work with limited data. Also, some information could be misrepresented or could be missing. Thereby, it could bring issues with the loss of valuable (essential) insights. That is why valid and reliable data could help to collaborate the working model (application) accordingly. Moreover, there will be no insights on this project due to the limited personnel.

Target audience: the position of stakeholders will be taken by regular people. They will decide the fate of the success of the developed application and its beneficial influence on the existing issue.

Data source: is provided by the well-known website "www.kaggle.com". The direct link to the dataset is "https://www.kaggle.com/fedesoriano/body-fat-prediction-dataset".

DATA WRANGLING

STEPS:

A. collection

B. organization

C. definition

D. cleaning

Collection: data was taken from 'Kaggle.com'. It is the world's largest data science community that was developed as the crowd-sourced platform.

The link to the dataset: " https://www.kaggle.com/fedesoriano/body-fat-prediction-dataset ".

Organization: during the process of reconstruction, we were able to take a close look at our dataset in case to identify the type of features that should align with our objectives.

So, we managed the column renaming with more applicable and less complex identifications.

```python
# Changing column's names
col_names = ['density', 'bodyfat', 'age', 'weight', 'height', 'neck', 'chest', 'abdomen',
             'hip', 'thigh', 'knee', 'ankle', 'biceps', 'forearm', 'wrist']
df.columns = col_names
df.head()
```

	density	bodyfat	age	weight	height	neck	chest	abdomen	hip	thigh	knee	ankle	biceps	forearm	wrist
0	1.0708	12.3	23.0	154.25	67.75	36.2	93.1	85.2	94.5	59	37.3	21.9	32	27.4	17.1

Also, we corrected the data types for some features (like: abdomen, hip, thigh, knee, ankle, biceps, forearm, forearm, wrist).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 255 entries, 0 to 254
Data columns (total 15 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   density  254 non-null    float64
 1   bodyfat  254 non-null    float64
 2   age      254 non-null    float64
 3   weight   254 non-null    float64
 4   height   254 non-null    float64
 5   neck     254 non-null    float64
 6   chest    254 non-null    float64
 7   abdomen  254 non-null    object 
 8   hip      254 non-null    object 
 9   thigh    254 non-null    object 
 10  knee     254 non-null    object 
 11  ankle    254 non-null    object 
 12  biceps   254 non-null    object 
 13  forearm  254 non-null    object 
 14  wrist    254 non-null    object 
dtypes: float64(7), object(8)
memory usage: 30.0+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 255 entries, 0 to 254
Data columns (total 15 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   density  254 non-null    float64
 1   bodyfat  254 non-null    float64
 2   age      254 non-null    float64
 3   weight   254 non-null    float64
 4   height   254 non-null    float64
 5   neck     254 non-null    float64
 6   chest    254 non-null    float64
 7   abdomen  253 non-null    float64
 8   hip      253 non-null    float64
 9   thigh    253 non-null    float64
 10  knee     253 non-null    float64
 11  ankle    253 non-null    float64
 12  biceps   253 non-null    float64
 13  forearm  253 non-null    float64
 14  wrist    253 non-null    float64
dtypes: float64(15)
memory usage: 30.0 KB
```

Additionally, we were able to remove the existence of 'Nan' values in some rows (because it was only a few rows with ‘NaNs’, there was no reason to waste time to fill them).

Finally, we were able to deal with 'extreme values'.

Data definition: because some countries have different units of measurements, we were required to convert the columns: weight, height. The initial unit values were pounds and feet. So, we converted them to kilograms and centimeters respectively.

```
# Convert column "weight" from lbs to kgs
df['weight'] = np.around((df['weight'] / 2.205), decimals=1)
df['weight']

0      70.0
1      78.6
2      69.8
3      83.8
4      83.6
       ...
247    60.9
248    91.2
249    84.7
250    86.5
251    94.1
Name: weight, Length: 252, dtype: float64
```

```
# Convert column "height" from inch to cm
df['height'] = np.around((df['height'] * 2.54), decimals=1)
df['height']

0      172.1
1      183.5
2      168.3
3      183.5
4      181.0
       ...
247    170.2
248    177.2
249    167.6
250    179.1
251    177.8
Name: height, Length: 252, dtype: float64
```

Data cleaning: because the dataset was relatively clean in the beginning (not corrupted) 'cleaning' process was mostly skipped.

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

density     1
bodyfat     1
age         1
weight      1
height      1
neck        1
chest       1
abdomen     2
hip         2
thigh       2
knee        2
ankle       2
biceps      2
forearm     2
wrist       2
dtype: int64
```

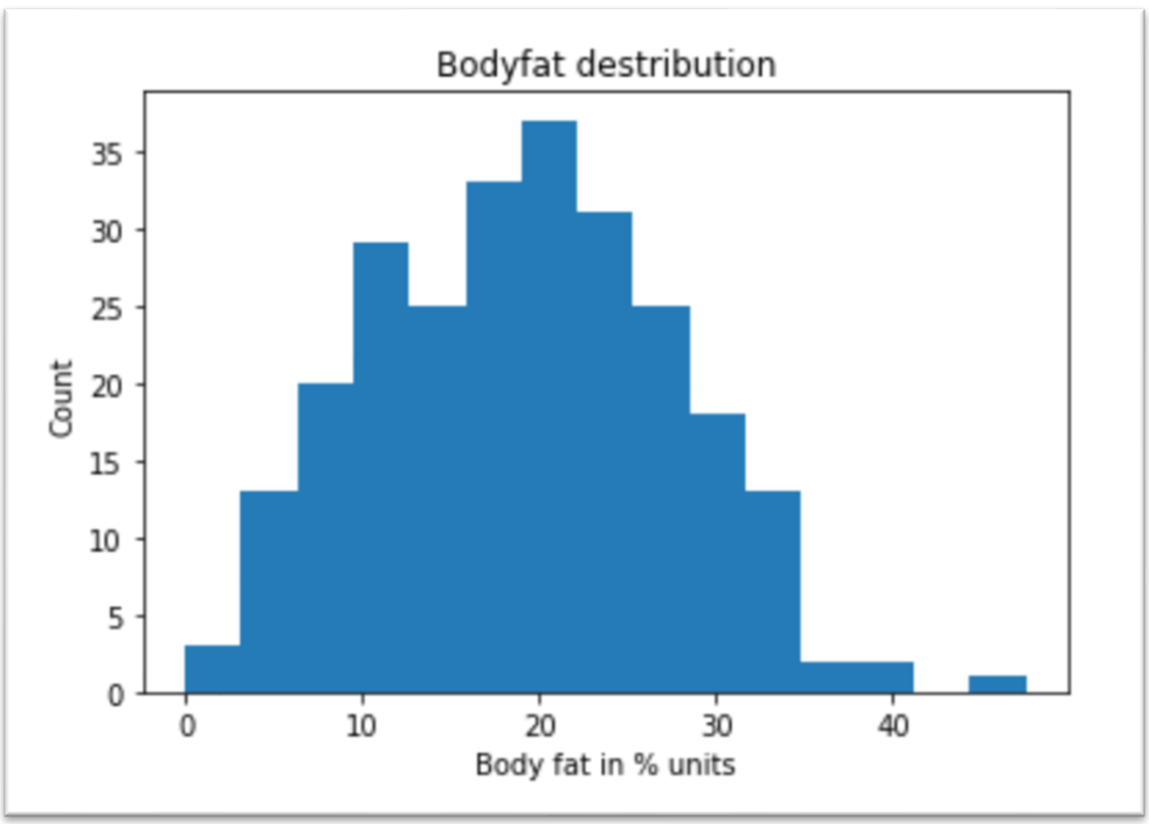
```
df.dropna(axis=0, how='any', inplace=True)
df.isnull().sum()

density    0
bodyfat    0
age        0
weight     0
height     0
neck       0
chest      0
abdomen    0
hip        0
thigh      0
knee       0
ankle      0
biceps     0
forearm    0
wrist      0
dtype: int64
```

EXPLORATORY DATA ANALYSIS (EDA)

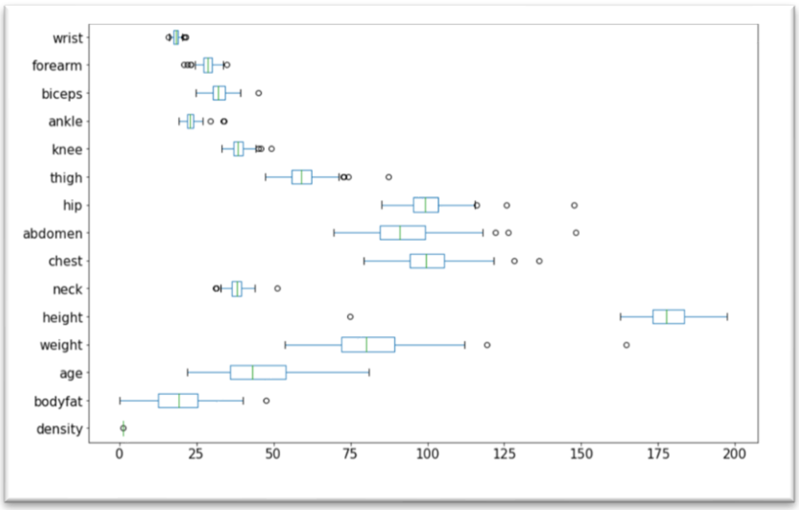
STEPS:

A. evaluate feature relationships using graphical visualization
B. identify learning algorithm

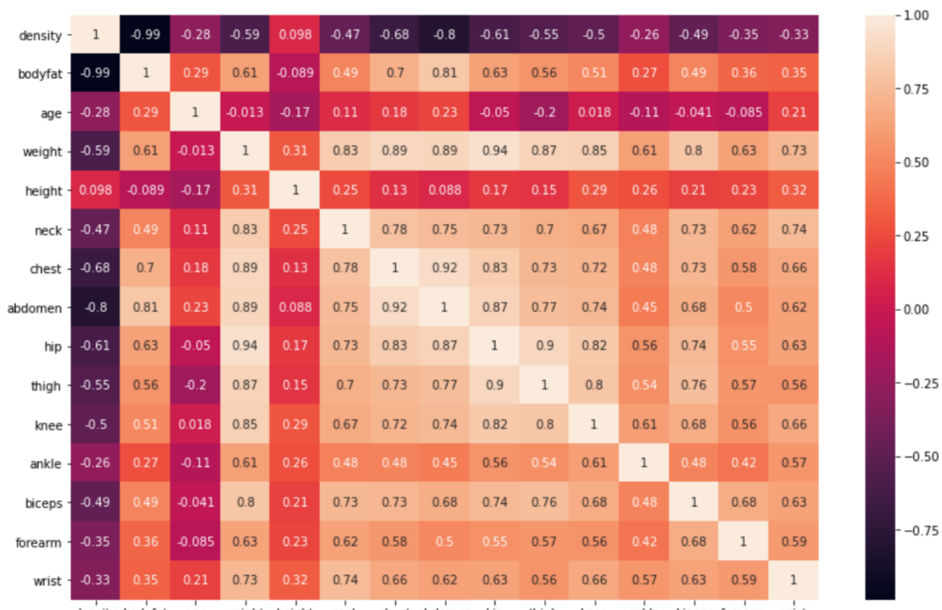
Graphical analysis: during this process, we were able to examine our data using graphical libraries like: 'matplotlib' and 'seaborn'. We began with an analysis of which variable will consider as our 'target' feature. So, we stuck with the 'bodyfat' column as our 'dependent' variable (values we will predict in future work). As we can see, the 'target' variable is almost normally distributed (that is a good sign for our model).



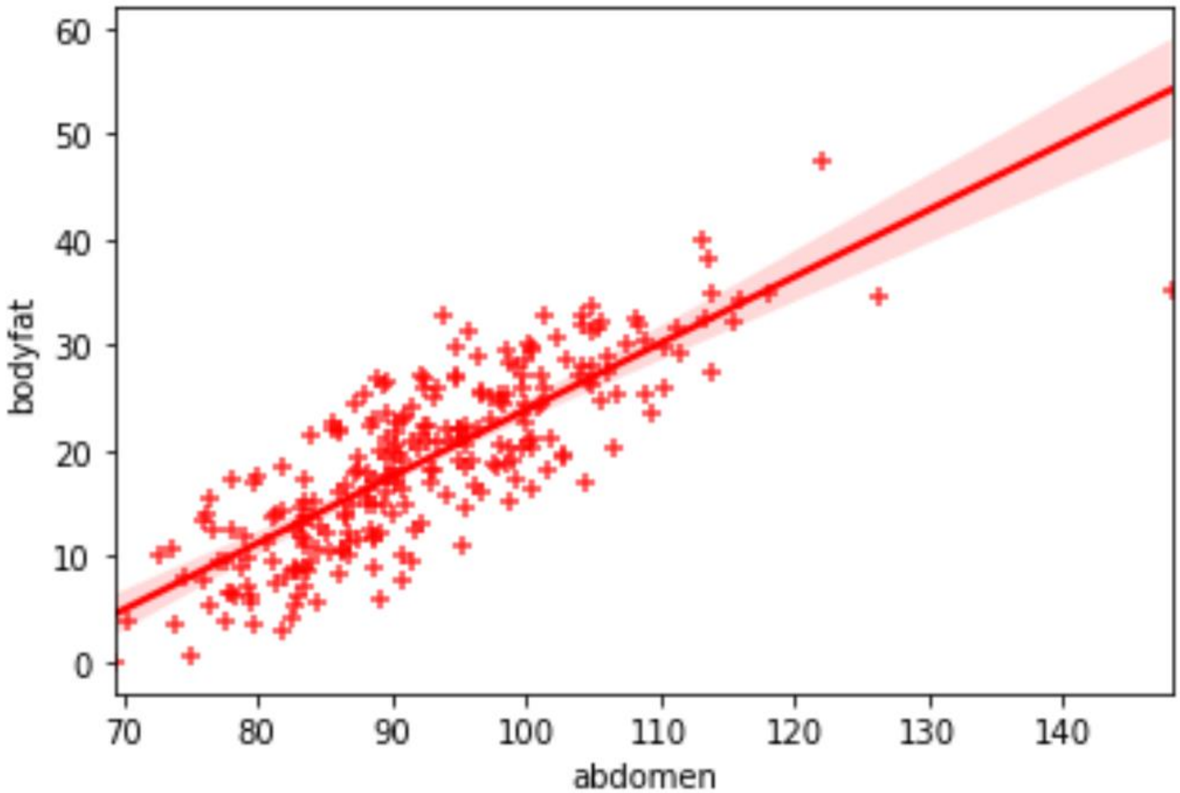
Unfortunately, there are many outliers in our dataset, that could badly affect the prediction. For now, we only keep our eye on them so later on, we could use it as leverage (room for improvement).



Additionally, we looked at the correlation among our independent features toward the 'target' one. For this process, we used the 'Pearson's correlation coefficient' method that gave us good insights about some 'strong' and 'week' correlations.



Learning algorithm: after we examined our data using special graphical libraries, we also were able to identify the relationship (behavior) of our data toward the target variable. It looked like, to solve our task we should be using the 'Linear Regression' approach.



PRE-PROCESSING AND TRAINING DATA

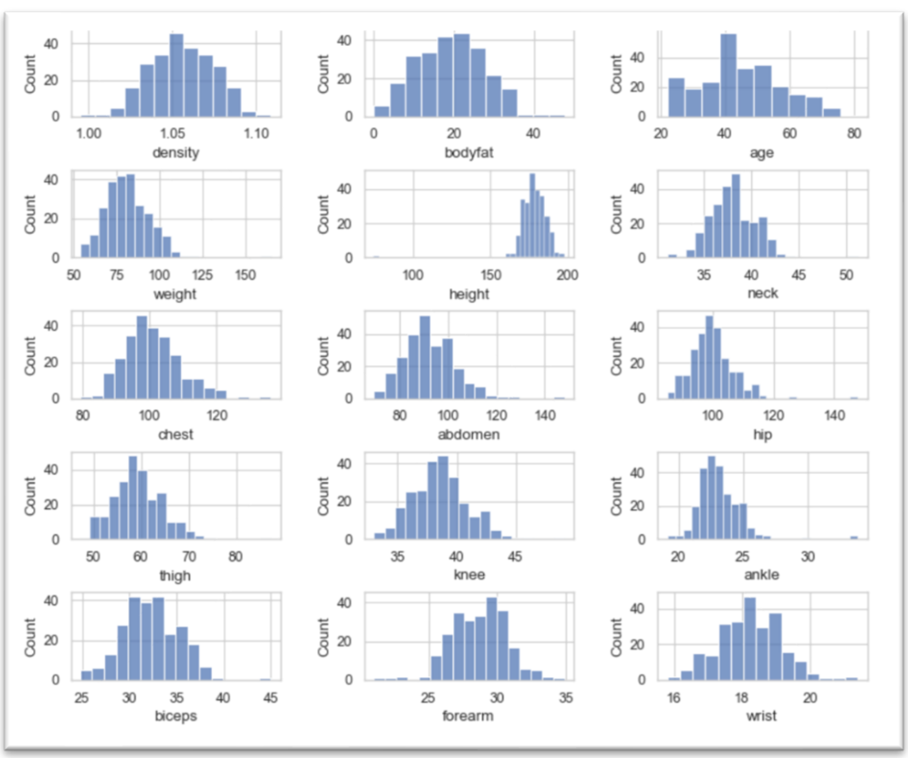
STEPS:

A. dealing with outliers

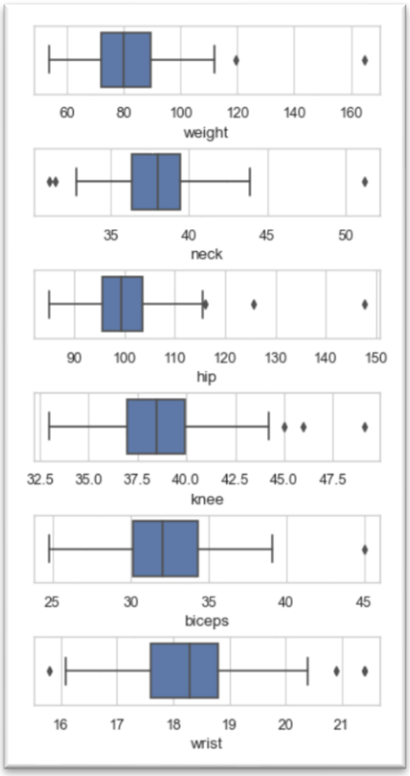
B. standardization process

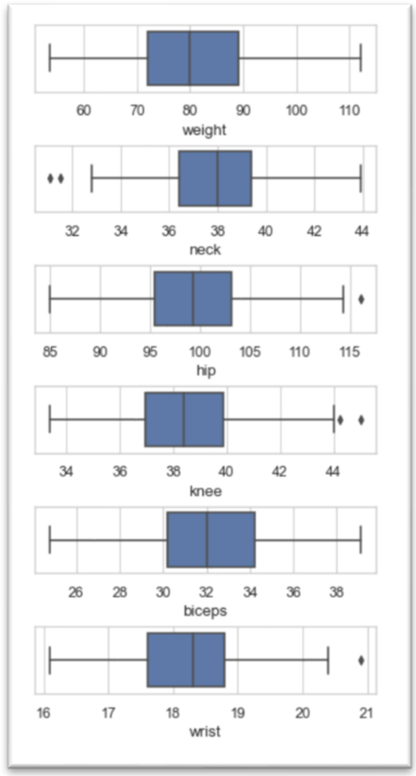
C. splitting into training and testing subsets

Outliers: during EDA we were able to spot some edge cases (another word for outliers). So, we decided to keep them for a while, until we will decide which type of logistic function we will be using for our prediction. So, now we decided to stick with the 'Linear Regression algorithm. As we know, this model is expecting the data to be 'normally distributed'. So, there were options between 'Normalization' or 'Standardization'. To make the right choice we looked at the distribution of our data.



As we could see most of the data was normally distributed. There was a significant issue with outliers that we decided to deal with before 'pre-processing'.

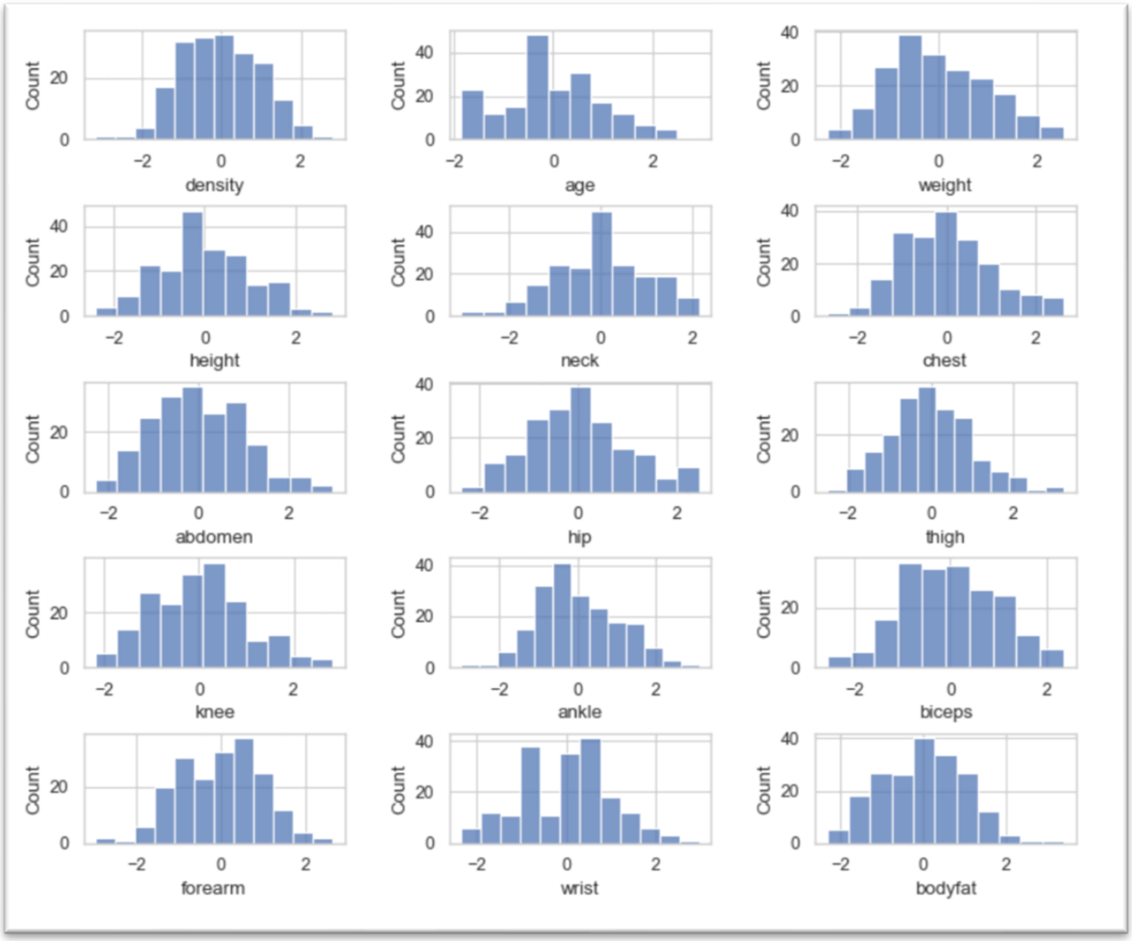




Splitting: once we have done with outliers, we went to the next step of splitting the dataset into training and testing parts. As a reminder, we did save two versions of our data, with and without outliers, because all outliers have meaning. But sometimes they do not apply to your task. That was the task for us to figure out.

```
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_1, y_1, test_size=0.2, random_state=42)
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_2, y_2, test_size=0.2, random_state=42)
print(X_train_1.shape, X_test_1.shape, y_train_1.shape, y_test_1.shape)
print(X_train_2.shape, X_test_2.shape, y_train_2.shape, y_test_2.shape)

(201, 14) (51, 14) (201,) (51,)
(194, 14) (49, 14) (194,) (49,)
```

Standardization: after the data was split, we applied the 'Standardization' process to normalize it. All features had the mean of '0' and standard deviation in the range of -1 to +1. Important to mention that 'Standardization' was 'fitted' on the training set and then used the 'transformed' method of the 'test' set in case to prevent 'data leakage'.



MODELING

STEPS:

A. train model

B. compare the results

C. final choice

Training model: at the final stage when the data was ready, we began applying different types of the 'Linear Regression' model. To understand our 'progress' (prediction quality) we had to pick the model that was the simplest and did not require much preparation. Our first choice was on 'DammyRegressor'. As we can see, because there was not much computation behind it, the prediction results are terrible.

The next choice was simple 'Linear Regression', also known as 'Ordinary Least Squares'. This model did not require a hyperparameter tunning process as well. Nevertheless, the results out of the box were very promising. As we can see here, there are two results, the upper one with 'edge cases' and at the bottom one without them. We have done the same testing procedure with the rest of the 'Linear Regression' models.

WITH OUTLIERS

```
Best score: 0.9658
Best parameter: {}

TRAIN SET --> R-squared: 0.9754 ... RMSE: 0.157
TEST SET --> R-squared: 0.9918 ... RMSE: 0.071
```

WITHOUT OUTLIERS

```
Best score: 0.961
Best parameter: {}

TRAIN SET --> R-squared: 0.973 ... RMSE: 0.164
TEST SET --> R-squared: 0.994 ... RMSE: 0.071
```

After, it was the 'Ridge' Regression model. This learning algorithm was required to provide an 'alpha' hyperparameter. So, we used the 'Gridsearch' object to try different values for 'alpha'. As the result below, we can see that the best value for 'alpha' is 2.0. We also can see that result is very competitive to the one we got from simple OLS.

WITH OUTLIERS

```
Best score: 0.9662
Best parameter: {'alpha': 1.5}

TRAIN SET --> R-squared: 0.9752 ... RMSE: 0.157
TEST SET --> R-squared: 0.9901 ... RMSE: 0.078
```

WITHOUT OUTLIERS

```
Best score: 0.9618
Best parameter: {'alpha': 2.0}

TRAIN SET --> R-squared: 0.9728 ... RMSE: 0.165
TEST SET --> R-squared: 0.9916 ... RMSE: 0.084
```

Then we gave the shot to the 'Lasso' learning algorithm. We also applied the 'GridSearch' object to tune the hyperparameter and got a result that was pretty much the same as before.

WITH OUTLIERS

```
Best score: 0.9676
Best parameter: {'alpha': 0.01}

TRAIN SET --> R-squared: 0.9747 ... RMSE: 0.159
TEST SET --> R-squared: 0.9943 ... RMSE: 0.059
```

WITHOUT OUTLIERS

```
Best score: 0.9619
Best parameter: {'alpha': 0.001}

TRAIN SET --> R-squared: 0.973 ... RMSE: 0.164
TEST SET --> R-squared: 0.9943 ... RMSE: 0.069
```

The final choice was placed on the 'ElasticNet' learning algorithm. The beauty of this one is the ability to tune multiple hyperparameters that could improve the overall result. Nevertheless, we couldn't get any improvements despite the 'superior' of this algorithm.

WITH OUTLIERS

```
Best score: 0.9682
Best parameter: {'alpha': 0.01, 'l1_ratio': 1.0, 'max_iter': 900, 'selection': 'random'}

TRAIN SET --> R-squared: 0.9747 ... RMSE: 0.159
TEST SET --> R-squared: 0.9938 ... RMSE: 0.062
```

WITHOUT OUTLIERS

```
Best score: 0.9638
Best parameter: {'alpha': 0.01, 'l1_ratio': 0.8, 'max_iter': 900, 'selection': 'random'}

TRAIN SET --> R-squared: 0.9722 ... RMSE: 0.167
TEST SET --> R-squared: 0.9945 ... RMSE: 0.068
```

Compare results and choosing model: by looking back at all results we have got we can conclude that, no matter which model we choose the accuracy is pretty much the same. So, in case to save time on 'computations' and 'complexity we stuck our choice on simple 'Linear Regression' algorithm that mostly outperforms the rest of models.

CONCLUSION

During processes of wrangling, EDA, preprocessing, and modeling we were able to achieve our goal. Our model is able to produce the result with about 95-99% of accuracy that is more than we expected in the beginning. Also, we were able to complete our project within a time frame that helped to save some expense. At this moment, any person who is willing to fight the battle against obesity now is equipped with a proper tool. It will help to refocus effort in the right direction.

Of course, there is still room for improvement. For example, we could target outliers or apply additional 'correlated features or implement the 'mobile' form of this application. The list is going on and on. The main point here is to have the interest to do so. But for now, we could say that project is successfully completed.

Citations:

Holland, Kimberly. “Obesity Facts in America.” *Healthline*, Healthline Media, 29 July 2020, www.healthline.com/health/obesity-facts#:~:text=More%20than%20one%2Dthird%20of,States%20are%20overweight%20or%20obese.