ML Obesity Journal

Jonathan Woodruff

July 28, 2025

Introduction

This journal documents in detail the process I used to create my supervised machine learning (ML) model for obesity predictions.

It is useful to show employers the steps I took to arrive at the final product.

Step 1: ETL

- July 29, 2025 — I downloaded the dataset from UC Irvine Machine Learning Repository and used the pandas read_me() function to convert it to a DataFrame.

Step 2: Data Cleaning

- July 30, 2025
  - I learned more about the data by printing df.info()
  - I confirmed there are no null values by using df.columns[df.isnull().sum() > 0]
  - I noticed the "Age" feature is sometimes listed as an integer and is sometimes listed as a float. For consistency, I took the floor of the Age values and converted the data type to int.

    ```
    take_floor = lambda x: np.floor(x)
    df['Age'] = df['Age'].apply(take_floor).astype(int)
    ```

  - I researched the meaning of the acronym for each feature name in the dataset.
- July 31, 2025
  - I realized the dataset is populated with mostly synthetic data, which explains why some of the Age values are of type float, so I reverted the Age data back to the original float values.
  - I looked at the unique values, min, and max of each feature to see if I could spot anything unexpected.

```
for feature in df:
    print('######FEATURE: ' + feature)
    print(df[feature].unique())
    print(df[feature].min())
    print(df[feature].max())
```

Step 3: Exploratory Data Analysis

- July 31, 2025
  - I gave an order to the ordinal variables.

```
df['CAEC'] = pd.Categorical(df['CAEC'], ['no', 'Sometimes', 'Frequently', 'Always'], ordere
df['CALC'] = pd.Categorical(df['CALC'], ['no', 'Sometimes', 'Frequently', 'Always'], ordere
```

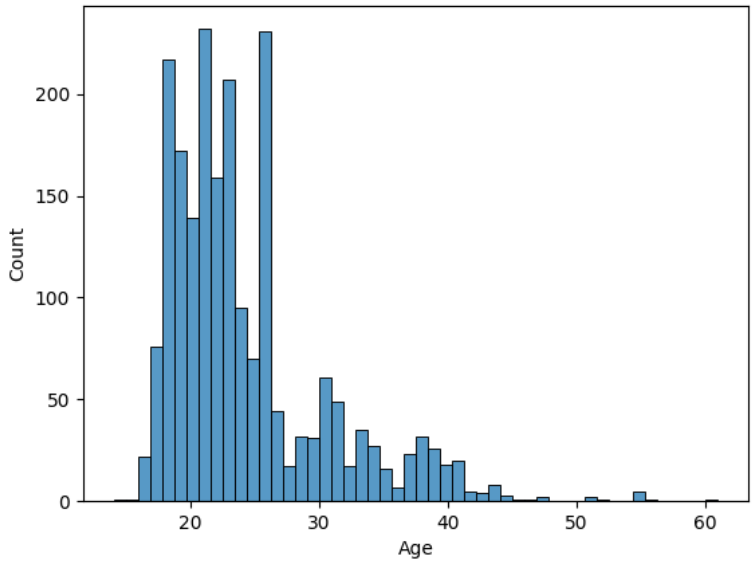

Figure 1: Age distribution

male	1068
female	1043

Table 1: Gender (frequencies)

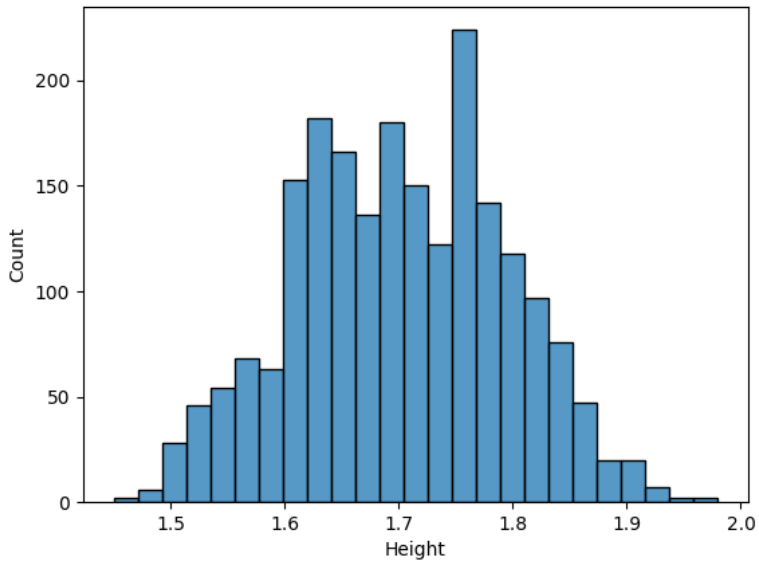

Figure 2: Height Distribution

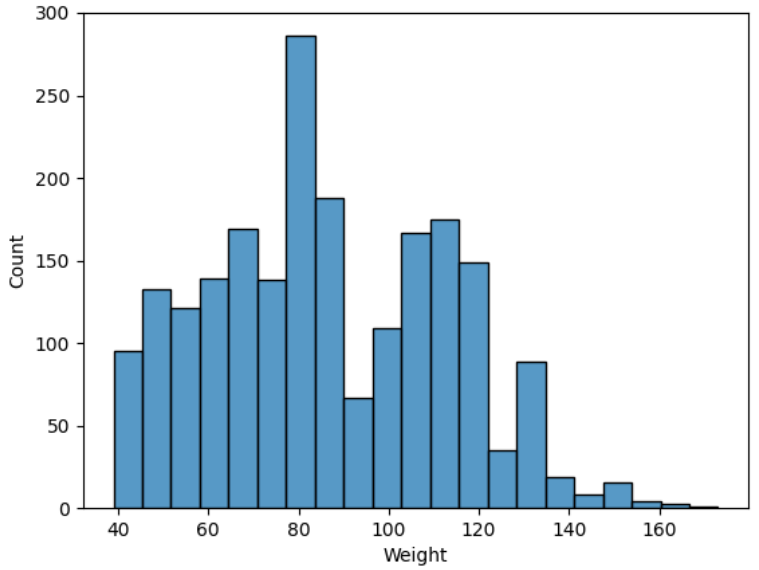

Figure 3: Weight Distribution

yes	1726
no	385

Table 2: Family history with overweight (frequencies)

yes	1866
no	245

Table 3: FAVC (frequencies)

no	51
Sometimes	1765
Frequently	242
Always	53

Table 4: CAEC (frequencies)

yes	44
no	2067

Table 5: SMOKE (frequencies)

yes	96
no	2015

Table 6: SCC (frequencies)

no	639
Sometimes	1401
Frequently	70
Always	1

Table 7: CALC (frequencies)

Public Transportation	1580
Automobile	457
Walking	56
Motorbike	11
Bike	7

Table 8: MTRANS (frequencies)

Obesity Type I	351
Obesity Type II	297
Obesity Type III	324
Overweight Level I	290
Overweight Level II	290
Normal Weight	287
Insufficient Weight	272

Table 9: NObeyesdad (label frequencies)

Step 4: Model Selection

- August 4, 2025
  - I followed the diagram from scikit-learn to select a model. I will be predicting a category, I have labeled data, and I have less than 100,000 samples, so I will start with **linear SVC**.
  - Per the documentation on SVC, I realized I need to normalize the features before fitting the model.

Step 5: Feature Engineering

- August 6, 2025
  - I realized having users respond to 16 survey questions might be too demanding, so I performed Principal Component Analysis (PCA) to see if I could eliminate some features without sacrificing model accuracy. Unfortunately, it seems almost every feature is required to account for 95% of the variation, so for the sake of my demonstration, I decided to keep all 16 features/survey questions.

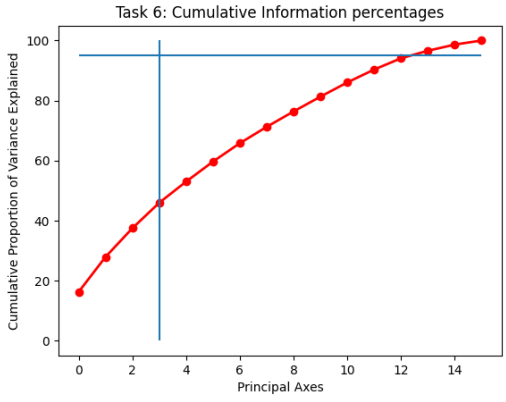

Figure 4: PCA

–

Step 6: Train-Test Split

- August 7, 2025
  - I split the data into 80% training data and 20% testing data.

Step 7: Normalize the Features

- August 7, 2025
  - I z-score normalized the features of the training and test data.

```
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train.values)
```

Step 8: Fit and Score the Model

- August 7, 2025
  - I fit the model using SVC with a linear kernel and otherwise default parameters which yielded a score of 0.966 on the training data and 0.948 on the testing data.

```
classifier = SVC(kernel="linear")
classifier.fit(X_train, y_train)
score_train = classifier.score(X_train, y_train)
score_test = classifier.score(X_test, y_test)
```

Step 9: Hyperparameter Tuning

- August 11, 2025
  - I tuned the hyperparameters using grid search and discovered the best parameters are kernel=linear and C=1. The default gamma value appears to work just as well as other gamma values.

```
# Initialize hyperparameters
parameters = {
    'kernel': ['linear', 'rbf'],
    'C': [.01, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1],
    'gamma': np.logspace(-9, 3, 13)
}

#Set up Grid Search
gs = GridSearchCV(classifier, parameters)
```

```
#fit the grid search classifier to the training data
gs.fit(X_train, y_train)
print(gs.best_estimator_)
print(gs.best_params_)

#compare scores between how the model does on training data (gs.best_score_) and test data
best_score = gs.best_score_
test_score = gs.score(X_test, y_test)
print(best_score)
print(test_score)

#get a nice view of the hyperparameter configurations and their scores
hyperparameter_grid = pd.DataFrame(gs.cv_results_['params'])
grid_scores = pd.DataFrame(gs.cv_results_['mean_test_score'], columns=['score'])
scores = pd.concat([hyperparameter_grid, grid_scores], axis = 1)
print(scores)
```

Conclusion

In conclusion, linear SVC with default parameters provides an excellent score for predictions on the testing data.