Matthias Eder, 11378093
Kai Guetzlaff, 10823590
Julia O'Brien, 8858381

IFN646 - Biomedical Data Science
Wearables Project

QUT, Semester 2, 2022
Group 9
Sunday 16th October, 2022

Abstract

The purpose of this study was to perform classification of sleep/wake cycles on data from an Apple Watch, using classification results from an Actiwatch worn on the same wrist as the ground truth. Participants wore both watches typically for two nights successively, and 27 nights were used in the analysis. Activity counts from the Actiwatch were provided along with the classification results and the method of calculating the total activity counts. Apple Watch data was in the form of ENMO values for every 15 seconds of accelerometry data, these were converted to activity counts using the Phillips algorithm provided. A linear regression model was built to predict activity counts from the Apple Watch data which resulted in a 97.46% accuracy. The analysis shows that the pre-processing and classification methods employed produce accurate results, and that the Apple Watch is able to produce almost the same classifications of sleep and wake cycles as the Actiwatch.

1. Introduction

A variety of algorithms have been used in the literature to classify periods of sleep and wakefulness, similarly a range of different devices have been compared. Epoch periods vary widely, from 1 second to 10 minute intervals, although 1 minute periods are commonly used to classify sleep vs. wake states. Often the algorithm will take into account the activity level immediately before and after the minute currently being classified, resulting in a moving average style classification [4]. Sensitivity thresholds can also vary and can be set independently by the researchers.

The purpose of the current project is to classify sleep and wake patterns using activity data from an Apple Watch, in comparison to the classification of activity counts as sleep and wake periods from an Actiwatch watch. Actigraphy, which is a method of monitoring human activity versus rest cycles [6], is used by the device to determine sleep periods. The devices use accelerometry data to estimate sleep parameters and patterns of sleep [2]. The results from the Actiwatch will be used as the ground truth in this study. The data being used was collected from participants who wore both the Apple Watch and the Actiwatch watch on the same wrist, typically for two consecutive nights.

a) Algorithms

The algorithms used to determine whether the wearer was asleep or awake are underpinned by mathematical expressions which use 'activity counts' as the input and translates them into a binary classification of 0 or 1, for each epoch. There are some algorithms that have been widely used and validated in previous sleep studies. The Cole–Kripke algorithm weighs the activity level at the time periods immediately prior (4 minutes) and post (2 minutes) to each epoch (minute) using predetermined weighting coefficients, the weighted sum is used to classify asleep or awake based on whether the resulting sum is above or below 1 [4]. Comparatively, the Sadeh algorithm (1994) uses an 11-minute window to classify the epoch (1 minute intervals) as asleep or awake, taking into account the 5 minutes previous and 5 minutes post, and a mean activity score and standard deviation for the minute being classified. Again a binary classification is used depending on whether the resulting probability of sleep totals ≥ 0, or below 0. Actiware software by Respironics (2018) again uses the window of time surrounding the current minute, 2 minutes either side, to determine sleep/wake in the algorithm. Coefficients are applied to the activity scores for each minute and the sum of these is compared to a threshold set independently, commonly 20, 40 or 80. Summed scores above the threshold result in a classification of awake versus below for asleep.

2. Approach

The activity counts will be "reproduced" using ENMO (Euclidean Norm Minus One) values from the Apple Watch, which represent 15 seconds of acceleration data. The Philips classification algorithm will then be used to classify the data point as sleep vs. wake.

Longer periods of the same classification will be used to determine the start of a sleep or wake period, for example taking the first 5 minutes of uninterrupted 0's as the point in which the wearer has fallen asleep.

a) Data description and pre-processing

The provided data is split into 27 different CSV files each representing a single night of activity monitoring. Each file contains four columns: the timestamp of an activity recording, Actiwatch activity counts, Actiware classification (1 for wake, 0 for sleep), and ENMO value calculated from the Apple Watch data. The 27 files are first combined into a single dataframe. As the dates have all been altered to preserve privacy, they do not convey any meaning anymore. The records in the first file are assigned to 1 January 2000, those in the second to 2 January 2000, and so on. Thus, a first measure was to extract the still meaningful time values to a new column ("time"). The days were also extracted to a separate column ("day") and contain the generally meaningless day-of-month number the files were assigned during anonymisation. We use these day values to separate the data into training and test sets. At first, we were hesitant to include the column with day values in the dataset used for the reproduction and classification experiments because we wanted to avoid introducing a false hierarchy and thus bias into the data. However, the model used does not take the day values into account. Furthermore, the values are needed for the Philips algorithm to define splits between a continuous stream of observations over night.

i. Split into training and test data

A random selection of 22 out of 27 days are selected as training data. The remaining 5 days are used as test data. Reproducibility and comparability are ensured by using the same seed in the random selection of the days for training and test sets. After all pre-processing steps, we are left with a total of 49,100 rows which are distributed among the two sets in the following way:

- training data: 40,507 rows, 82.5%

- test data: 8,593 rows, 17.5%

The pre-processing is executed after the data splitting in order to avoid leaking bias from the training data into the test data (and vice-versa). Each subset is pre-processed separately.

ii. Outlier analysis

An important step in data preparation is to check the distributions of columns. This should also point out possible outliers that might happen due to erroneous data collection, measurement errors or other reasons. At the same time, it is important to keep potential extreme values in the data. The Interquartile Range (IQR) of each variable is used for outlier detection.

We decided to use a custom IQR interval as reference for outlier detection. The IQR is multiplied with an empirically determined factor (default is 4) for both sides of the interval (i.e. for quantiles at 0.25 and 0.75) to account for natural phenomena and actual outliers like all-time spikes in either direction. The factor 4 is chosen empirically after a lot of testing and is rather large because we want to avoid discarding actual spikes in the data. We decided to focus only on clear outliers and therefore chose this large factor. A possible improvement of this approach could be to choose adaptive factors for each column. The factors might also be tuned with a separate model to improve performance for our target. We noticed there are still some potential outliers left after our strategy had been applied which have a seemingly strong influence on the resulting model. However, we refrained from removing these values as this would have possible amounted to data tuning so that the results would confirm more strongly with our hypothesis.

The outlier analysis shows that only the column "Actiwatch activity counts" contains potential outliers. For the three different sets this amounts to the following distribution for this column:

- full data: 6.85% outlier candidates, amounts to 1.37% of the data
- training data: 6.69% outlier candidates, amounts to 1.34% of the data
- test data: 7.6% outlier candidates, amounts to 1.52% of all data

This is a first approach to get a feeling of the outlier distribution in our data. Of course, the magnitude of outliers is more important than their total or relative numbers. Especially if we were to transform these values later on it would be bad to disregard columns containing only few outlier candidates (which would happen in larger data analyses) because their deviation is the measure that is of interest. However, the magnitude of outliers is already factored into this analysis as a cut-off at a multiple of the IQR is considered. The potential outliers are replaced by the median of the column. This estimator is chosen as it is robust to outliers.

iii. Missing value analysis

The next step is to check for missing values in the data. An initial analysis reveals that 0.26% of all available data are NaN values. The column "Actiware classification" contains 0.8%, "Actiwatch activity counts" 0.43%, and "Apple Watch ENMO" 0.07%. The day and time columns are complete.

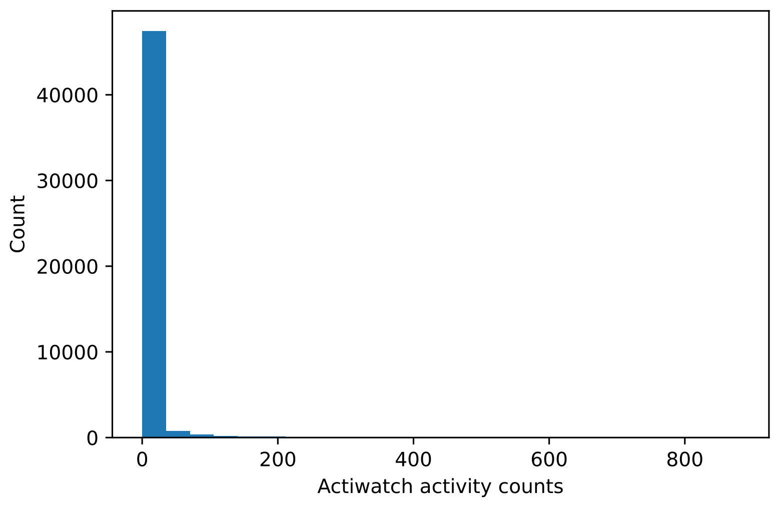


Figure 1: Data distribution of column `Actiwatch activity counts` in the full data

As the origin of missing values in the data is unknown, we initially chose to apply three different strategies to handle them in pre-processing:

- remove missing values
- set missing values to 0
- set missing values to 1

Initial experiments for the reproduction of activity counts indicated that the original strategy seemed to be a replacement with 0 for all missing values. However, we were also interested to see how the results of our classification experiments would change if we handled the missing values differently. Thus, we decided to try out three different ways. All of these strategies are safe because the amount of missing data is very low and would not affect the quality of the resulting models in a significant way. The missing value percentages above are reported for the full data but apply to all subsets as the treatment of missing data does not depend on the distribution of the data.

- full data: 0.26%
- training data: 0.27%
- test data: 0.22%

After several experiments with the Philips algorithm, we noticed that the missing values should be treated differently as the count reproduction depends on previous values. We also observed that every day begins with several rows where both "Actiwatch activity counts" and "Actiware classification" are missing followed by a few rows of missing classifications while the activity counts are present. The latter situation can also be observed for the end of each file. We decided to drop the rows were both columns are missing as nothing can be salvaged from these data (212 rows, 0.43%). Furthermore, we decided to extract the first and last intervals of five minutes

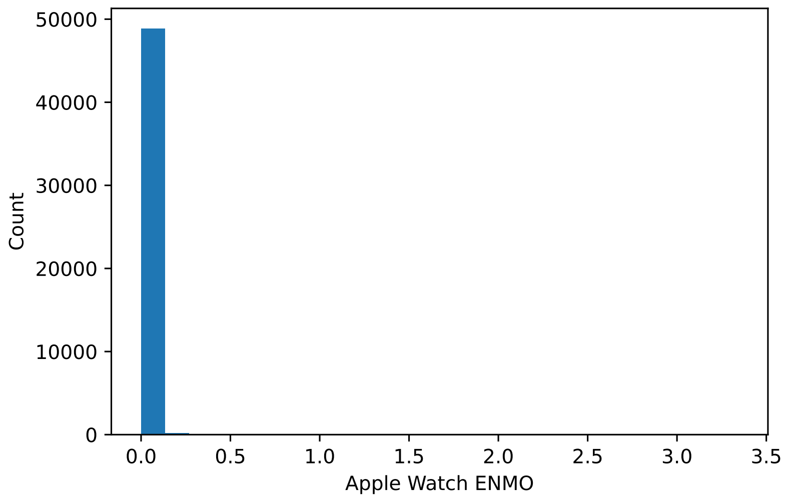


Figure 2: Data distribution of column `Apple Watch ENMO` in the full data

uninterrupted sleep per day which are identified by continuous streams of 0 values. All classification values before or after these intervals are set to awake, respectively. This strategy resulted in 1,436 rows being set 1 which amounts to 2.92% of the data. As stated in the project description, this is a common strategy in actigraphy and worked well for the classification reproduction.

It is important to note that around 98% of all values in the "Actiwatch activity counts" are 0 for the full data. Figure 1 shows the distribution of this column for the full data. Similarly, Figure 2 shows the distribution of the "Apple Watch ENMO" data where the same pattern is noticeable. While this is expected for actigraphy data where the observed patient is supposed to sleep for most of the night interrupted by short intervals of activity, this needs to be taken into account when we interpret the results.

b) Activity count reproduction

As indicated above, we "reproduced" the Philips activity counts from the Apple Watch ENMO data, and then used the Philips classification algorithm which first computes the total activity counts at epoch e using the weighted sum

$$\text{totalCounts}(e) = 0.04 \sum_{i=-8}^{-5} a_{e+i} + 0.2 \sum_{i=-4}^{-1} a_{e+i} + 4a_e + 0.2 \sum_{i=1}^{4} a_{e+i} + 0.04 \sum_{i=5}^{8} a_{e+i}$$

and then classifies the respective epoch e as sleep if $\text{totalCounts}(e) \leq 40$, and as awake if $\text{totalCounts}(e) > 40$.

In the following, we will firstly describe a plausibility check on the data and our classification functions which ensures that our algorithms work flawlessly for the next step: the Linear Regression Classification, which we depict in Section ii..

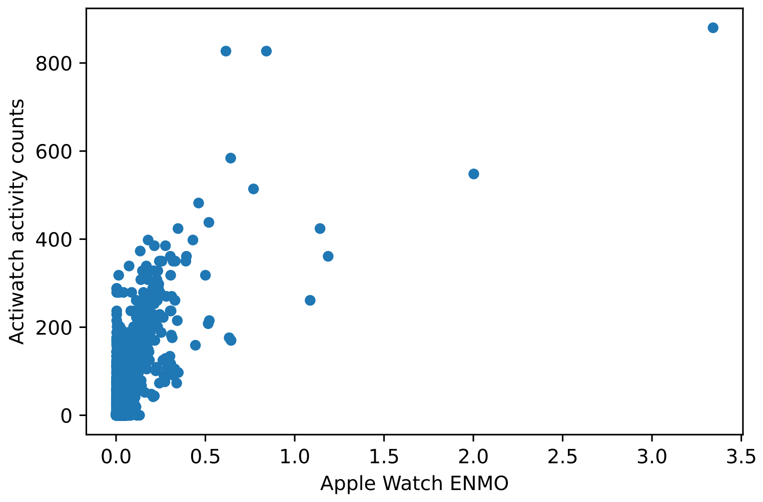


Figure 3: Scatter plot of Apple Watch and Actiwatch data in the train data set

i. Plausibility Check

In order to validate the `Actiware classification` column as well as our implementation of the classifier and determine the used threshold in the data, we calculated the total counts for every epoch e according to the above formula. In the next step, we applied the classification according to a set threshold. Lastly, we looked for the first 5 minutes of uninterrupted sleep, and set all epochs before that to wake. Similarly, we looked for the last 5 minutes of uninterrupted sleep, and set the epochs after that to wake, respectively.
The aim of that data validity check as a whole is to then compare the calculated classification to the already given classification in the data and check whether it is congruent to it. If that is the case, we can be sure that our implementation of the total counts and classification functions are sound and that we chose the correct threshold. The results of the plausbility check are depicted in Section 3..

ii. Linear Regression

In order to get an idea of the data distribution, we first plotted the data. The Apple Watch ENMO data being our independent variable is depicted in x-direction, and the Actiwatch activity counts, which we try to predict, is depicted on the y-axis (cf. Figure 3).
As one can immediately see, the two features are positively correlated. However, that is not surprising as both features depict acceleration data for the same time intervals while one person wearing the two devices on the same arm. Furthermore, the pattern we observe is shaped roughly linearly. Thus, the idea to use a linear classifier that takes the Apple Watch ENMO data as input and returns or “reproduces” the Philips activity counts immediately suggests itself. The benefit of that approach is its simplicity. We get a function with the shape $f(x) = ax + b$ and can therefore see very clearly how the two features depend on one another. The explainability of that low-level Machine Learning approach is, hence, very high. Further, more sophisticated approaches like Neural Networks would not necessarily yield better results as there is no need for more involved

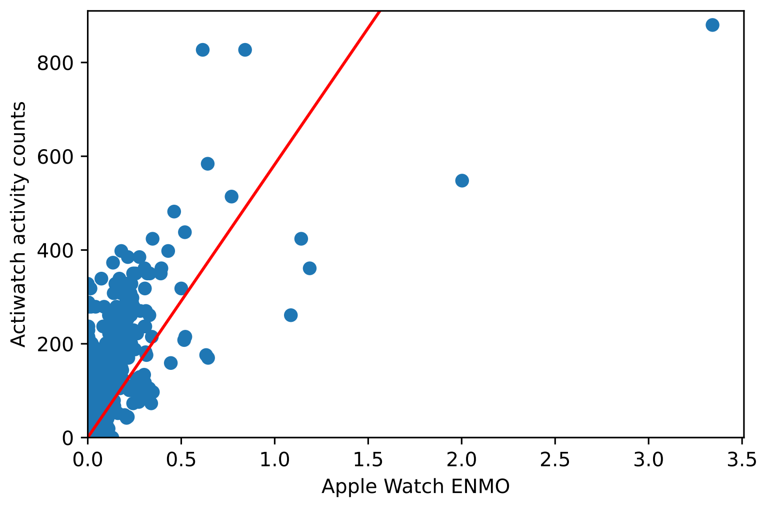


Figure 4: Scatter plot of Apple Watch and Actiwatch data in the train data set with linear regression line $f(x) = 582.3x + 0.2081$

approaches when having just two simple features with a clear linear correlation. The use of more technical approaches would therefore merely add more overhead and reduce the explainability of the model.

The next step is then to fit our linear regression model onto the training data while using the sum of squared errors as our error metric. There are also other metrics like the mean squared error or root mean squared error, however, as they all express the same idea while the first one has the simplest derivative, we chose to use that one as it runs more efficiently and terminates faster. The output of that approach is a linear function which we added to the scatter plot in Figure 3 and depict in the result section in Figure 4.

Having the model, we are now able to predict the activity counts from the Apple Watch data, which we do for yet unseen data: the test data set. We do that to be able to validate the model completely unbiased. From that predictions, again, the total counts can be derived according to our function which we plausibility checked earlier on. Having that, we classify into sleep and wake according to the same threshold used above and lastly handle the uninterrupted sleep for 5 minutes topic and set all before the first and after the last uninterrupted interval to wake.

To sum up, we reproduced the activity counts from Apple Watch ENMO data using a linear classifier. In the next step, the total counts could be derived. From that, a classification could be made. Lastly, we took the uninterrupted sleep topic into consideration for our classifier and reclassified the corresponding epochs accordingly. In the next section, we depict the results of our plausibility check and classification.

3. Results

During the plausibility check, we encountered several issues that helped us debug the code faster and in the end, we were able to develop a fairly accurate reproduction of the original classification. The threshold that minimised the error rate was 40. Ultimately, 19 misclassifications (i.e., 0.05%)

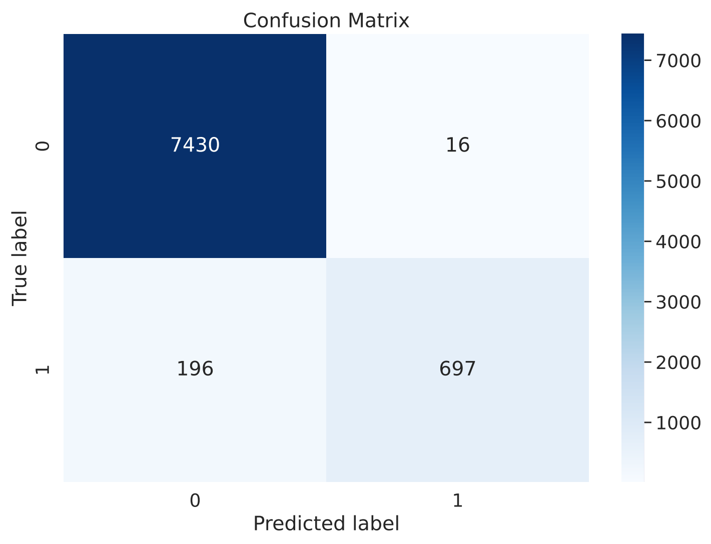


Figure 5: Confusion matrix for the test data set, 0 is sleep, 1 is wake

remain, which we would further try to minimise in the next iteration. Those 19 items might be due to some NA values handled improperly, or a bug in the uninterrupted sleep algorithm. This can be investigated in future work.

In the following, we give the results of our main classification. In Figure 4, the linear classification graph is depicted, having the function $f(x) = 582.3x + 0.2081$. Intuitively, when one would like to predict the activity count from the Apple Watch ENMO data, one just needs to insert the Apple Watch variable into the function. The line clearly shows the linear relationship and portrays a sound classifier.

The confusion matrix of our classification is depicted in Figure 5. It shows, that 7430 epochs are correctly classified as sleep. Further, 697 epochs are correctly classified as wake. Only 16 epochs are classified as wake, whereas they are in fact sleep. Lastly, 196 epochs are falsely classified as sleep, whereas they are in fact wake. Looking just at those numbers, we intuitively can see that the classification we made worked out very good as the misclassified epochs are not too frequent and the numbers are quite low compared to the correctly classified items. In Table 1, we depict different metrics and scores percentage-wise in order to get a better and more profound understanding and feeling for the performance of our model.

Metric	Score
Accuracy	97.46%
Misclassification rate	2.54%
Precision	97.43%
Recall	99.79%
F1	98.59%

Table 1: Performance of classification

The model accuracy is very high with 97.46%, which results in a very low misclassification rate of 2.54%. The precision is 97.43%, and the recall sums up to 99.79%. Note that those metrics rely on true/false positives/negatives and therefore require a definition of what is positive and negative. We decided to define the state of sleep as the positive class as that is what we are classifying. Lastly, the F1-score is 98.59%. All metrics being in the high 90's provide a stunning result of our classification approach. In the next section, we will discuss and interpret our findings and approach.

4. Discussion

The method that was used to classify the Apple Watch data as either sleep or wake was similar to the methods previously discussed in the literature review. Both the Sadeh, and Cole and Kripke algorithms assess wrist actigraph data and classify sleep/wake cycles based on a moving average window for each epoch [4]. In this analysis, a function was applied to calculate total activity counts according to Phillips Actiware software specification (seen in Section b)) that considers the value of the current epoch point being classified, the previous 4 epochs and the 4 epochs immediately following it. Each of these points are assigned a weight and the resulting sum is used as the activity count for that epoch.
The results that were obtained were not unexpected as there was a clear relationship shown between the variables. Some errors were also expected as the data set was not balanced between the two classes, with the majority of the classifications being '0' as seen in Figure 2. The incorrect classifications could also be due to the fit of the linear regression model to the data. It can be seen in Figure 3 that there are outliers in the data, and although some were removed as part of the preprocessing, some outliers, which were deemed important to preserve the nature of the data as they may be providing valuable information, were kept and may introduce errors. The Interquartile Range (IQR) was used to determine which outliers should be removed, which considers the unbalanced classes as it is skewed according to the distribution of the data. We introduced an empirically chosen scaling factor for the IQR to account for the data hand. Different scaling factor values were trialed to remove the outliers while keeping naturally occurring spikes in the data and for now, we settled on 4-times the IQR. However, we intend to perform further experiments to improve the quality of our models while still preserving the true nature of the data in future work.
The linear regression model was chosen as the nature of the data was quite limited, with only 2 features and 2 classes, as well as a small sample of data. Polynomial and quadratic equations were also trialed, but at the risk of overfitting the limited data, these were not seen as the best options. Again, more complicated methods of classification such as more sophisticated machine learning techniques or neural nets were considered but deemed unnecessary for the simplistic nature of this data, and the task.
An assumption was made to account for missing data in the classification column for the Actiwatch data, which could have potentially also introduced some errors. It was assumed that all epochs prior to the first 5 minutes and last 5 minutes of uninterrupted sleep (i.e. 5 minutes of zeros in classification) could be set to awake, as this was before and after the wearer fell sleep and woke up again. Imputing this missing data helped in the classification as the algorithm used a weighted sum of the epochs before and after the instance in question each time. Other methods of imputing this missing data were trialed, using 0's instead of 1's – although this hindered the performance

of the model to a greater extent and did not make as much sense practically. Leaving the missing data as NaN values was not possible due to the nature of the Philip's algorithm to reproduce the counts which relies on previous and following data points. As we observed patterns of NaN values at the beginning and end of each recording session, leaving these entries in the data would thus lead to consequential errors severely affecting the quality of our model.
The main limitations of this analysis concerned the data itself, which had a small number of variables and contained missing values in the ground truth. The data was also separated by 'days' and this information needed to be preserved for classification, making it difficult to use other methods of classification such as a Neural Network or other more sophisticated models.

5. Conclusion

This research was focused on different wearable technology and aimed to determine the utility of the Apple Watch in predicting sleep and wake cycles when compared to a 'ground truth', in this study the Actiwatch activity count data and subsequent classification. We were able to replicate activity counts for the Apple Watch data based on the Phillips Actiware software specification, classify the scores based on a given threshold, and from this build a model to predict classification on unseen data. This was evaluated against the ground truth data and the results were very accurate (97%). Therefore, the Apple Watch can be considered accurate at classifying sleep and wake cycles, as there were no large discrepancies between the classification results from each device. This also signifies that the method employed to perform the classification, by converting ENMO to activity counts can be considered suitable.
We did not have a hypothesis in this study as we expected the counts to line up quite closely, and therefore have a high accuracy in prediction, considering the watch was worn by the same person on the same hand, and both used acceleration data. We have noted that there are clear limitations of both the data and the method employed in this study that could be addressed in future work, namely the limited features of data and the size of the data set, which resulted in a simple linear model being used. Recommendations for future research are as follows:

- Participants wear the device while awake as well as when going to bed/sleeping. This will provide a larger data set and even out the class imbalance, which may allow the algorithm to be able to build a better representation of what is being classified as awake versus asleep.
- If obtainable, more variables could be included in the data such as heart rate, breath rate or body temperature while the participant is sleeping.
- More data in general, either more nights of sleep per participant (as there was only 2) or more participants included in the study.

Obtaining a larger data set would allow the use of more advanced classification techniques such as machine learning.

References

[1] Alexander J. Boe et al. “Automating Sleep Stage Classification Using Wireless, Wearable Sensors”. In: *npj Digital Medicine* 2.1 (1 Dec. 20, 2019), pp. 1–9. ISSN: 2398-6352. DOI: 10.1038/s41746-019-0210-1. URL: https://www.nature.com/articles/s41746-019-0210-1 (visited on 09/21/2022).

[2] Joseph Cheung et al. “Validation of Minute-to-Minute Scoring for Sleep and Wake Periods in a Consumer Wearable Device Compared to an Actigraphy Device”. In: *Sleep Science and Practice* 2.1 (Sept. 2, 2018), p. 11. ISSN: 2398-2683. DOI: 10.1186/s41606-018-0029-8. URL: https://doi.org/10.1186/s41606-018-0029-8 (visited on 09/21/2022).

[3] *Comparing GENEActiv against Actiwatch-2 over Seven Nights Using a Common Sleep Scoring Algorithm and Device-Specific Wake Thresholds*. URL: https://www.tandfonline.com/doi/epub/10.1080/15402002.2021.1924175?needAccess=true (visited on 09/21/2022).

[4] Desta Fekedulegn et al. “Actigraphy-Based Assessment of Sleep Parameters”. In: *Annals of Work Exposures and Health* 64.4 (Apr. 30, 2020), pp. 350–367. ISSN: 2398-7308, 2398-7316. DOI: 10.1093/annweh/wxaa007. URL: https://academic.oup.com/annweh/article/64/4/350/5735350 (visited on 09/21/2022).

[5] Paul H. Lee and Lorna K. P. Suen. “The Convergent Validity of Actiwatch 2 and ActiGraph Link Accelerometers in Measuring Total Sleeping Period, Wake after Sleep Onset, and Sleep Efficiency in Free-Living Condition”. In: *Sleep and Breathing* 21.1 (Mar. 1, 2017), pp. 209–215. ISSN: 1522-1709. DOI: 10.1007/s11325-016-1406-0. URL: https://doi.org/10.1007/s11325-016-1406-0 (visited on 09/21/2022).

[6] *Medical Definition of ACTIGRAPHY*. URL: https://www.merriam-webster.com/medical/actigraphy (visited on 09/22/2022).

[7] Jean Paquet, Anna Kawinska, and Julie Carrier. “Wake Detection Capacity of Actigraphy During Sleep”. In: *Sleep* 30.10 (Oct. 1, 2007), pp. 1362–1369. ISSN: 0161-8105. pmid: 17969470. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2266273/ (visited on 09/21/2022).

Appendices

preprocess.py

```python
import pandas as pd
import numpy as np
import pickle
import time
import random
from os import listdir, makedirs
from os.path import isfile, join, exists
import warnings

warnings.filterwarnings('ignore')

DATA_SETS = ["Full data", "Training data", "Test data"]


def load_data(
        data_path="raw_data",
        cache_file_name="combined_sleep_data",
        cache_dir_name="cached_data"
):
    """
    Loads all files from the given path. Performs preprocessing and returns the cleaned data as three DataFrames:
        training data, test data, and validation data. Extracts time and day from the timestamp column and puts them
        into new columns. Removes timestamp column. Splits data into training, test, and validation sets.

    :param cache_dir_name: The name of the directory used to keep cached files.
    :param cache_file_name: The name of the pickle file (without ending) to contain the cached DataFrame.
    :param data_path: The path to the directory containing the raw input data.
    :return: Three DataFrames containing the concatenated sleep records separated into full, train, test, and validation
        sets.
    """
    cache_file_names = [
        f"{cache_dir_name}/{cache_file_name}_full.pkl",
        f"{cache_dir_name}/{cache_file_name}_train.pkl",
        f"{cache_dir_name}/{cache_file_name}_test.pkl"
    ]

    if not exists(cache_dir_name):
        makedirs(cache_dir_name)

    if __cache_files_exist(cache_file_names):
        # load existing pickle files
        print("Loading cached files.")
        data = []
        for file in cache_file_names:
            with open(file, "rb") as f:
                data.append(pickle.load(f))
    else:
        # find files in input directory
        data_files = [f for f in listdir(data_path) if isfile(join(data_path,
        ↪  f))]
        print(f"Found {len(data_files)} input files in directory
        ↪  '{data_path}'.")

        # load and concatenate input data
        dataframes = []
        for day, file in enumerate(data_files):
            frame = pd.read_csv(f"{data_path}/{file}")
            frame.insert(0, 'day', day + 1)
            dataframes.append(frame)
        data_concat = pd.concat(dataframes, ignore_index=True)

        # day/time extraction
        data_concat["timestamp"] = pd.to_datetime(data_concat["timestamp"],
        ↪  format="%d/%m/%Y %H:%M:%S")
        data_concat["time"] = data_concat.timestamp.dt.time
        data_concat.drop("timestamp", axis=1, inplace=True)

        print(f"Data has {data_concat.shape[0]} entries before
        ↪  preprocessing.\n")

        # split data into training, test, validation sets
        training_data, test_data = __split_data(data_concat)

        # run preprocessing and pickle data
        data = [data_concat, training_data, test_data]
        for data_set, name, file in zip(data, DATA_SETS, cache_file_names):
            with open(file, "wb") as f:
                processed = __preprocess_data(data_set, name)
                print(f"{name} has {processed.shape[0]} entries after
                ↪  preprocessing.\n")
                pickle.dump(processed, f)

    return data


def inform(df):
    """
    Prints some information about the given DataFrame.

    :param df: The DataFrame to analyse.
    """
    print(f"Shape of data: {df.shape}")
    cl_0 = (df["Actiware classification"] == 0).sum()
    ac_0 = (df["Actiwatch activity counts"] == 0).sum()
    print(f"There are {(cl_0 / len(df)) * 100.0:.2f}% 0 values in column
    ↪  'Actiware classification'.")
    print(f"There are {(ac_0 / len(df)) * 100.0:.2f} 0 values in column
    ↪  'Actiwatch activity counts'.")


def __split_data(df, num_train=22, num_test=5):
    """
    Splits the combined DataFrame into training and test sets. Takes a random
↪   set of days into account per set that can
    be specified via parameters.

    :param df: The original combined DataFrame.
    :param num_train: The number of days to include in the training set.
    :param num_test: The number of days to include in the test set.
    :return: Two DataFrames for training and test sets.
    """
    if num_train + num_test > 27:
        raise Exception("Ratio for train/test split is incorrect!")
    seq = range(1, 28)
    random.seed(123456)

    train_days = random.sample(seq, k=num_train)
    seq = [x for x in seq if x not in train_days]
```

```python
    test_days = random.sample(seq, k=num_test)

    train = df[df["day"].isin(train_days)]
    test = df[df["day"].isin(test_days)]

    return train, test


def __preprocess_data(data, name):
    """
    Performs preprocessing steps on the data. The steps are:
        1) Perform outlier detection.
        2) Handle entries with NaN values.

    :param name: The name of the DataFrame.
    :param data: The DataFrame containing input data.
    :return: The DataFrame with the preprocessed data.
    """

    print(f"===== {name} =====")

    # outlier detection, replace outliers with median
    outlier_detection_set = data
    potential_outliers = __detect_outliers(outlier_detection_set)
    potential_outlier_cols = potential_outliers.sum()[potential_outliers.sum() >
    ↪   0]
    potential_outlier_cols = potential_outlier_cols * 100 /
    ↪   len(outlier_detection_set)
    pot_out_perc = potential_outliers.sum() * 100 / len(outlier_detection_set)
    print(f"{len(potential_outlier_cols)} of {outlier_detection_set.shape[1]}
    ↪   columns contain possible outliers.")
    print(f"{pot_out_perc.mean():.2f}% of the data are considered a potential
    ↪   outlier and are replaced by the mean.")
    print(potential_outlier_cols.sort_values(ascending=False))
    removed_outliers = outlier_detection_set.mask(potential_outliers,
    ↪   outlier_detection_set.median(), axis=1)
    # data = removed_outliers

    # missing data analysis
    print('\nMissing values for columns:')
    missing_values, mean_mv = __missing_data_analysis(data)
    print(f"Percentage of missing values in merged sleep data: {mean_mv:.2f}%")
    print("\nMissing data per columns in percent:")
    print(missing_values)

    # handle missing values according to supplied strategy
    data = __handle_missing_values(data, 'Actiware classification')

    return data


def __handle_missing_values(df, classification_col):
    """
    Missing values are handled according to the note for actigraphy in the
    assignment specification. The first and last
    5-minute intervals of uninterrupted sleep per day are extracted and all
    classification values before or after that
    are set to 1 (i.e. awake). Rows where both the columns 'Actiwatch activity
    counts' and 'Actiware classification' are
    safely removed.

    :param df: The DataFrame containing the (sub)set of data.
    :param classification_col: The column containing classification values.
    :return: The DataFrame with handled missing values.
    """
    # remove rows where both activity counts and classification are missing
    length_before = len(df)
    df = df[~(df['Actiwatch activity counts'].isna() & df['Actiware
        classification'].isna())]
    days = df['day'].unique()
    indices = []
    for day in days:
        subset = df[df['day'] == day]

            indices.append(__find_first_and_last_uninterrupted_sleep_intervals(subset,
            classification_col))

    rows_altered = 0
    for start, first_sleep, last_sleep, end in indices:
        rows_altered += first_sleep - start + end - last_sleep
        df.loc[start:first_sleep + 1, classification_col] = 1
        df.loc[last_sleep:end + 1, classification_col] = 1

    print(f"""
    {length_before - len(df)} rows were dropped where both activity counts and
    classification were missing.
    That is roughly {((length_before - len(df)) / length_before) * 100.0:.2f}%
    of the dataset.""")
    print(f"""
    {rows_altered} classifications were set to 1 for the first and last 5
    minutes of uninterrupted sleep.
    That is roughly {(rows_altered / len(df)) * 100.0:.2f}% of the dataset.""")
    return df


def __find_first_and_last_uninterrupted_sleep_intervals(df, classification_col):
    """
    Extracts the first and last 5-minute intervals of uninterrupted sleep (i.e.
    continuous 0 values) from the
    classification column.

    :param df: The sub-DataFrame containing actigraphy for a single day to check
    for sleep intervals.
    :param classification_col: The column containing classification values.
    :return: A 4-tuple: the start index of the sub-DataFrame, the index of the
    beginning of the first 5-minute interval,
    the index of the end of the last 5-minute interval and the end index of
    sub-DataFrame.
    """
    indices = df.index[df[classification_col] == 0].tolist()
    first_sleep = np.nan
    last_sleep = np.nan
    for i, index in enumerate(indices):
        interval = list(range(index, index + 21))
        if indices[i:i + 21] == interval:
            first_sleep = index
            break
    for i, index in enumerate(reversed(indices)):
        interval = list(range(index - 20, index + 1))
        comparison_interval = indices[-i - 21:None if i == 0 else -i]
        if comparison_interval == interval:
            last_sleep = index
            break
    if first_sleep == np.nan or last_sleep == np.nan:
        raise Exception("Something went wrong while extracting the first and
        last sleep indices.")
    return df.index[0], first_sleep, indices[-1], df.index[-1]


def __detect_outliers(df, iqr_factor=4):
    """
    Performs an outlier detection with regard to an interval of a multiple of
    ↪ the IQR.

    :param df: The DataFrame to check for outliers.
    :param iqr_factor: The factor for the IQR.
    :return: A boolean mask indicating outliers where True.
    """
    q1s = df.quantile(0.25)
    q3s = df.quantile(0.75)
    iqr = (q3s - q1s) + 1
    lo = q1s - iqr_factor * iqr
    hi = q3s + iqr_factor * iqr
    outliers_mask = (df <= lo) | (df >= hi)
    return outliers_mask


def __missing_data_analysis(df):
    """
    Checks a given DataFrame for missing values. Returns a DataFrame with the
    ↪ percentage of missing values per column.

    :param df: The original DataFrame to check.
    :return: A DataFrame containing the percentage of missing values per column
    ↪ and the mean percentage of missing
    values of the whole data.
    """
    perc_missing = df.isnull().sum() * 100 / len(df)
    mv = pd.DataFrame({'column_name': df.columns, 'percent_missing':
        ↪ perc_missing})
    mv.sort_values(by="percent_missing", ascending=False, inplace=True)
    mv.reset_index(drop=True, inplace=True)
    return mv, mv['percent_missing'].mean()


def __cache_files_exist(cache_file_names):
    for file in cache_file_names:
        if not exists(file):
                return False
        return True


if __name__ == '__main__':
    start_time = time.time()
    full_d, training_d, test_d = load_data()
    print(f"{time.time() - start_time:.2f} seconds elapsed for data
    ↪  loading/preprocessing.")
```

IFN646 - Biomedical Data Science — Wearables Project

Import of necessary packages

In [1]:
```
import matplotlib.pyplot as plt
import numpy
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score, precision_score, recall_score
from preprocess import load_data, inform, __handle_missing_values
import warnings

# create image directory
from pathlib import Path
Path("img").mkdir(parents=True, exist_ok=True)
warnings.filterwarnings('ignore')
```

Load the data

In [2]:
```
full, train, test = load_data()
```

```
Loading cached files.
```

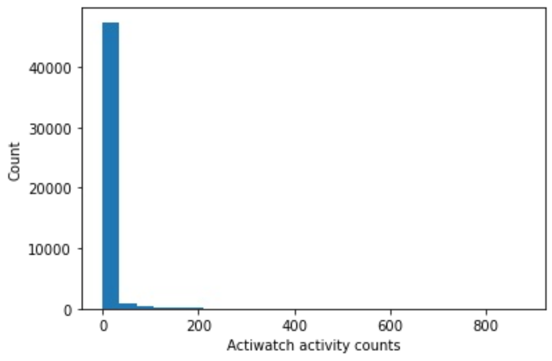
Print statistics of the datasets

In [3]:
```
print("Full data:")
inform(full)
print("Training data:")
inform(train)
print("Test data:")
inform(test)
```

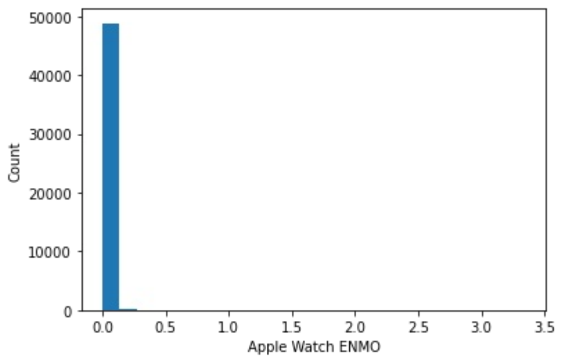
```
Full data:
Shape of data: (49100, 5)
There are 90.74% 0 values in column 'Actiware classification'.
There are 92.14 0 values in column 'Actiwatch activity counts'.
Training data:
Shape of data: (40761, 5)
There are 91.03% 0 values in column 'Actiware classification'.
There are 92.13 0 values in column 'Actiwatch activity counts'.
Test data:
Shape of data: (8339, 5)
There are 89.29% 0 values in column 'Actiware classification'.
There are 92.18 0 values in column 'Actiwatch activity counts'.
```

Gain overview of data

In [4]:
```
# plot histogram of Actiwatch activity counts for the whole data set
plt.hist(full['Actiwatch activity counts'], bins=25)
plt.ylabel('Count')
plt.xlabel('Actiwatch activity counts');
plt.savefig('img/actiwatch_histogram.pdf', bbox='tight')
```



In [5]:
```
# plot histogram of Apple Watch ENMO for the whole data set
plt.hist(full['Apple Watch ENMO'], bins=25)
plt.ylabel('Count')
plt.xlabel('Apple Watch ENMO');
plt.savefig('img/apple_watch_histogram.pdf', bbox='tight')
```



Function to calculate total counts according to Philips’ Actiware software specification

In [6]:

```
def total_counts(df, src_col, dest_col):
    day = df['day'].values
    cts = df[src_col].values
    total = []
    for i in range(len(cts)):
        div_by_25_sum = 0
        div_by_5_sum = 0
        for j in range(-8, -4):
            if i + j >= 0 and day[i + j] == day[i]:
                div_by_25_sum += cts[i + j]
        for j in range(-4, 0):
            if i + j >= 0 and day[i + j] == day[i]:
                div_by_5_sum += cts[i + j]
        for j in range(1, 5):
            if i + j < len(cts) and day[i + j] == day[i]:
                div_by_5_sum += cts[i + j]
        for j in range(5, 9):
            if i + j < len(cts) and day[i + j] == day[i]:
                div_by_25_sum += cts[i + j]
        calculation = 0.04 * div_by_25_sum + 0.20 * div_by_5_sum + 4.00 * cts[i]
        total.append(calculation)
    df[dest_col] = total
```

In [7]:

```
# call total_counts function and add a column for total counts from Actiwatch
total_counts(train, 'Actiwatch activity counts', 'Actiwatch Total Counts')

# print first 30 items
train.head(30)
```

Out[7]:

	day	Actiwatch activity counts	Actiware classification	Apple Watch ENMO	time	Actiwatch Total Counts
1	1	109.0	1.0	0.227648	20:58:15	555.20
2	1	170.0	1.0	0.217089	20:58:30	812.40
3	1	91.0	1.0	0.267528	20:58:45	548.68
4	1	101.0	1.0	0.222397	20:59:00	607.12
5	1	125.0	1.0	0.262205	20:59:15	727.64
6	1	105.0	1.0	0.283417	20:59:30	673.96
7	1	176.0	1.0	0.314253	20:59:45	954.84
8	1	105.0	1.0	0.328872	21:00:00	689.72
9	1	159.0	1.0	0.444264	21:00:15	897.32
10	1	215.0	1.0	0.521921	21:00:30	1110.12
11	1	208.0	1.0	0.515725	21:00:45	1095.32
12	1	91.0	1.0	0.318492	21:01:00	637.72
13	1	97.0	1.0	0.348385	21:01:15	651.84
14	1	134.0	1.0	0.301678	21:01:30	773.68
15	1	125.0	1.0	0.292101	21:01:45	762.48
16	1	117.0	1.0	0.306116	21:02:00	692.08
17	1	76.0	1.0	0.273415	21:02:15	517.52
18	1	73.0	1.0	0.242683	21:02:30	484.24
19	1	385.0	1.0	0.276460	21:02:45	1639.80
20	1	2.0	1.0	0.004816	21:03:00	156.08
21	1	0.0	1.0	0.002099	21:03:15	126.12
22	1	0.0	1.0	0.002393	21:03:30	110.08
23	1	0.0	1.0	0.002089	21:03:45	93.04
24	1	0.0	1.0	0.002052	21:04:00	26.44
25	1	0.0	1.0	0.001939	21:04:15	21.44
26	1	0.0	1.0	0.001993	21:04:30	18.40
27	1	0.0	1.0	0.002051	21:04:45	15.48
28	1	0.0	1.0	0.001956	21:05:00	0.08
29	1	0.0	1.0	0.002015	21:05:15	0.00
30	1	0.0	1.0	0.001976	21:05:30	0.00

Helper functions that classifies into sleep/wake according to threshold 40

In [8]:

```
def classify(row, col):
    if row[col] > 40:
        return 1
    else:
        return 0
```

Plausibility Check

Perform classification of actiwatch total counts for plausibility check

In [9]:

```
train['Actiware classification calculated'] = train.apply(lambda x: classify(x, 'Actiwatch Total Counts'), axis

# set uninterrupted sleep values
train = __handle_missing_values(train, 'Actiware classification calculated')

#print first 30 elements
train.head(30)
```

```
0 rows were dropped where both activity counts and classification were missing.
That is roughly 0.00% of the dataset.

1125 classifications were set to 1 for the first and last 5 minutes of uninterrupted sleep.
That is roughly 2.76% of the dataset.
```

Out[9]:

	day	Actiwatch activity counts	Actiware classification	Apple Watch ENMO	time	Actiwatch Total Counts	Actiware classification calculated
1	1	109.0	1.0	0.227648	20:58:15	555.20	1
2	1	170.0	1.0	0.217089	20:58:30	812.40	1
3	1	91.0	1.0	0.267528	20:58:45	548.68	1
4	1	101.0	1.0	0.222397	20:59:00	607.12	1
5	1	125.0	1.0	0.262205	20:59:15	727.64	1
6	1	105.0	1.0	0.283417	20:59:30	673.96	1
7	1	176.0	1.0	0.314253	20:59:45	954.84	1
8	1	105.0	1.0	0.328872	21:00:00	689.72	1
9	1	159.0	1.0	0.444264	21:00:15	897.32	1
10	1	215.0	1.0	0.521921	21:00:30	1110.12	1
11	1	208.0	1.0	0.515725	21:00:45	1095.32	1
12	1	91.0	1.0	0.318492	21:01:00	637.72	1
13	1	97.0	1.0	0.348385	21:01:15	651.84	1
14	1	134.0	1.0	0.301678	21:01:30	773.68	1
15	1	125.0	1.0	0.292101	21:01:45	762.48	1
16	1	117.0	1.0	0.306116	21:02:00	692.08	1
17	1	76.0	1.0	0.273415	21:02:15	517.52	1
18	1	73.0	1.0	0.242683	21:02:30	484.24	1
19	1	385.0	1.0	0.276460	21:02:45	1639.80	1
20	1	2.0	1.0	0.004816	21:03:00	156.08	1
21	1	0.0	1.0	0.002099	21:03:15	126.12	1
22	1	0.0	1.0	0.002393	21:03:30	110.08	1
23	1	0.0	1.0	0.002089	21:03:45	93.04	1
24	1	0.0	1.0	0.002052	21:04:00	26.44	1
25	1	0.0	1.0	0.001939	21:04:15	21.44	1
26	1	0.0	1.0	0.001993	21:04:30	18.40	1
27	1	0.0	1.0	0.002051	21:04:45	15.48	1
28	1	0.0	1.0	0.001956	21:05:00	0.08	1
29	1	0.0	1.0	0.002015	21:05:15	0.00	1
30	1	0.0	1.0	0.001976	21:05:30	0.00	1

Compare classification to calculated classification

In [10]:

```
classification_stats = train.groupby(["Actiware classification", "Actiware classification calculated"]).size()
print(classification_stats)
```

```
Actiware classification  Actiware classification calculated
0.0                      0                                     37106
1.0                      0                                        19
                         1                                      3632
dtype: int64
```

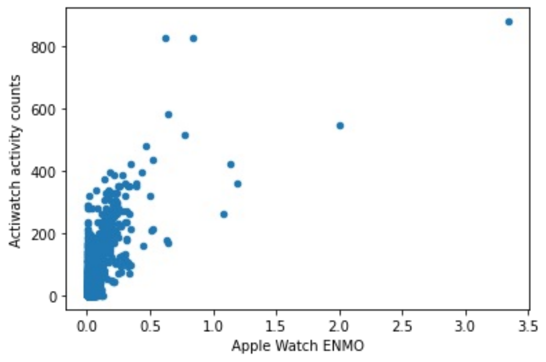
The plausibility check in which we re-classified the sleep/wake state according to Philip's software specification almost yielded a perfect result. Merely 19 values are misclassified. This might be due to some NA values handled improperly or a bug in the uninterrupted sleep algorithm. We will investigate that further in the next iteration.

Fit Machine Learning Model

Draw scatter plot from Apple Watch and Actiwatch

In [11]:

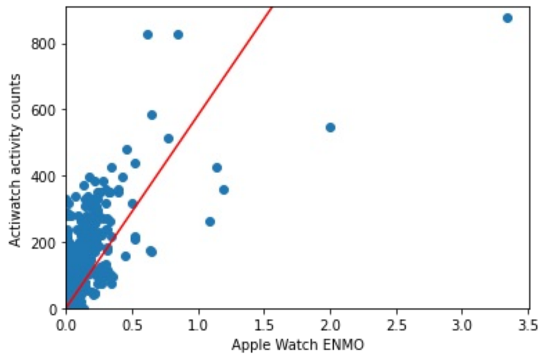
```
train.plot.scatter(x='Apple Watch ENMO', y='Actiwatch activity counts')
plt.savefig('img/scatter_plot.pdf', bbox='tight')
```



Fit linear Regression Model

In [12]:
```
# declare x and y for the model
x = train['Apple Watch ENMO']
y = train['Actiwatch activity counts']

x.fillna(0, inplace=True)

# fit linear model
model = numpy.poly1d(numpy.polyfit(x, y, 1))

# create linspace to draw scatter plot in next step
line = numpy.linspace(0, 3.5, 1000)

# scatter plot
plt.scatter(x, y)

# draw regression graph into plot
plt.plot(line, model(line), color='red')

# set limits
plt.xlim([0,3.51])
plt.ylim([0,910])

# set labels
plt.xlabel("Apple Watch ENMO")
plt.ylabel("Actiwatch activity counts")

plt.savefig('img/scatter_plot_with_regression_line.pdf', bbox='tight')

print('The function of the regression line is:\nf(x) =', str(model).strip())
```

```
The function of the regression line is:
f(x) = 582.3 x + 0.2081
```



Predict if sleep or awake for test data

In [13]:
```
# predict activity counts
test['Predicted activity counts'] = model(test['Apple Watch ENMO'])

# calculate total counts from prediction
total_counts(test, 'Predicted activity counts', 'Predicted Total Counts')

# print first 15 rows
test.head(15)
```

Out[13]:

	day	Actiwatch activity counts	Actiware classification	Apple Watch ENMO	time	Predicted activity counts	Predicted Total Counts
7789	5	91.0	1.0	0.049485	22:11:15	29.023007	317.353564
7790	5	62.0	1.0	0.047339	22:11:30	27.773589	323.652867
7791	5	58.0	1.0	0.069403	22:11:45	40.621203	379.619787
7792	5	154.0	1.0	1.066049	22:12:00	620.965014	2585.415302
7793	5	164.0	1.0	0.503060	22:12:15	293.138021	1340.419208
7794	5	159.0	1.0	0.117267	22:12:30	68.492317	482.563578
7795	5	94.0	1.0	0.075325	22:12:45	44.069387	385.774107
7796	5	0.0	1.0	0.003893	22:13:00	2.474936	221.692133
7797	5	6.0	1.0	0.006534	22:13:15	4.012591	128.706828
7798	5	0.0	1.0	0.003435	22:13:30	2.208075	74.220209
7799	5	0.0	1.0	0.003716	22:13:45	2.372147	63.183453
7800	5	0.0	1.0	0.003637	22:14:00	2.326054	54.829289
7801	5	0.0	1.0	0.004128	22:14:15	2.611610	31.191735
7802	5	0.0	1.0	0.003439	22:14:30	2.210868	17.684373
7803	5	0.0	1.0	0.003159	22:14:45	2.047337	14.349980

In [14]:

```python
# classify
test['Predicted wake'] = test.apply(lambda x: classify(x, 'Predicted Total Counts'), axis=1)

# set uninterrupted sleep values
test = __handle_missing_values(test, 'Predicted wake')

# print first 15 rows
test.head(15)
```

```
0 rows were dropped where both activity counts and classification were missing.
That is roughly 0.00% of the dataset.

268 classifications were set to 1 for the first and last 5 minutes of uninterrupted sleep.
That is roughly 3.21% of the dataset.
```

Out[14]:

	day	Actiwatch activity counts	Actiware classification	Apple Watch ENMO	time	Predicted activity counts	Predicted Total Counts	Predicted wake
7789	5	91.0	1.0	0.049485	22:11:15	29.023007	317.353564	1
7790	5	62.0	1.0	0.047339	22:11:30	27.773589	323.652867	1
7791	5	58.0	1.0	0.069403	22:11:45	40.621203	379.619787	1
7792	5	154.0	1.0	1.066049	22:12:00	620.965014	2585.415302	1
7793	5	164.0	1.0	0.503060	22:12:15	293.138021	1340.419208	1
7794	5	159.0	1.0	0.117267	22:12:30	68.492317	482.563578	1
7795	5	94.0	1.0	0.075325	22:12:45	44.069387	385.774107	1
7796	5	0.0	1.0	0.003893	22:13:00	2.474936	221.692133	1
7797	5	6.0	1.0	0.006534	22:13:15	4.012591	128.706828	1
7798	5	0.0	1.0	0.003435	22:13:30	2.208075	74.220209	1
7799	5	0.0	1.0	0.003716	22:13:45	2.372147	63.183453	1
7800	5	0.0	1.0	0.003637	22:14:00	2.326054	54.829289	1
7801	5	0.0	1.0	0.004128	22:14:15	2.611610	31.191735	1
7802	5	0.0	1.0	0.003439	22:14:30	2.210868	17.684373	1
7803	5	0.0	1.0	0.003159	22:14:45	2.047337	14.349980	1

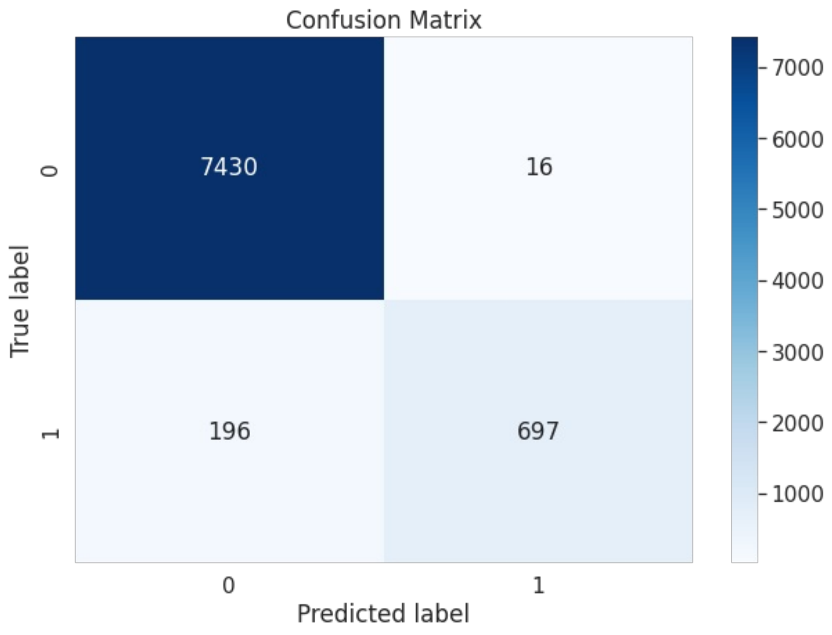
Print statistics of classification

Confusion matrix

In [15]:

```python
# create matrix
conf_mat = confusion_matrix(test['Actiware classification'], test['Predicted wake'])

# put matrix into data frame
df_cm = pd.DataFrame(conf_mat, range(2), range(2))

# plot matrix with blues color style
plt.figure(figsize=(10,7))
```

```
sns.set(font_scale=1.4)

s = sns.heatmap(df_cm, annot=True, cmap='Blues', fmt='g')
s.set(xlabel='Predicted label', ylabel='True label', title='Confusion Matrix')

fig = s.get_figure()
fig.savefig("img/confusion_matrix.pdf", bbox='tight')
```



Metrics: Accuracy, Misclassification Rate, Precision, Recall, F1-score

Note: As we are trying to classify sleep, we consider 0 as the positive class

In [16]:
```
# Accuracy
acc = round(accuracy_score(test['Actiware classification'], test['Predicted wake'])*100, 2)
print("Accuracy score: \t", acc, '%')

# Misclassification Rate
print("Misclassification rate:\t", round(100-acc, 2), '%')

# Precision
print("Precision score:\t",
      round(precision_score(test['Actiware classification'], test['Predicted wake'], pos_label=0)*100, 2), '%'

# Recall
print("Recall: \t\t",
      round(recall_score(test['Actiware classification'], test['Predicted wake'], pos_label=0)*100, 2), '%')

# F1-Score
print("F1 score: \t\t",
      round(f1_score(test['Actiware classification'], test['Predicted wake'], pos_label=0)*100, 2), '%')
```

```
Accuracy score:         97.46 %
Misclassification rate:  2.54 %
Precision score:        97.43 %
Recall:                 99.79 %
F1 score:               98.59 %
```