Predict Waist and Weight Size on 4+ Years of Tracked Data

Dean Chao-Kai Chung, April 2025

1 Executive Summary:

Dataset was acquired through self gathering, and manual tracking of weight and waist measurmets daily. Along with the daily caloric and macronutrient (protein, fat, carbohydrates) consumption. All of these values are tracked on a messy semi-organized data.

This is my first personal project, utilizing what I've learnt so far from self teaching and online resources. So really, its testing the waters on working on a workbook/project without so many guardrails/hand holding from online courses, but also curious if a model is able to predict fluctuations of my weight and waist.

Instead of predicting weight/waist measurement of the next day, I did the difference of weight/waist measurement for the next day from the data given, as i thought it'd cause data leakage issues to use weight/waist measurements in y. Regardless, I still ran into a lot of data leakage problems during training, due to the nature of having both weight and waist related information existing in both X and y datasets.

Through the exploration and visualization of the data/contents, there's a lot more nuance and external factors that play into waist and weight change besides caloric and macro count. Prediction showed how sensitive the model is to any sort of change in life style and diet, where it ends up over generalizing the weight and waist, aka predicting near 0.

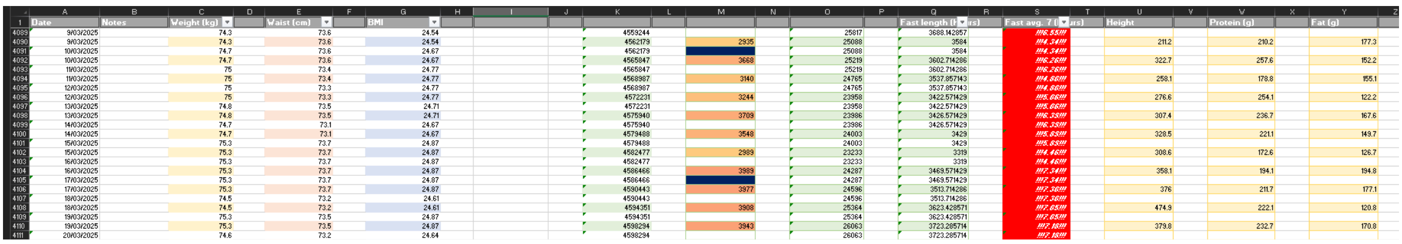
2 Data Exploration:

2.1 Semi-Organized Data/Cleaning:

Data is semi-organized on an excels sheet, with collection of data began on 20th Aug 2019 but only contained weight/waist data with missing days. However starting from 30th March 2020 till present began daily collection of data in weight, calories, and macros. The sheet contains 3000+ cels over 4+ years of records. Data required some cleaning with empty columns or singular non-null columns that's used for presentation/communication purposes on the excel sheet.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30637 entries, 0 to 30636
Data columns (total 68 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Date                  30172 non-null  object
 1   Notes                 15 non-null     object
 2   Weight (kg)           3853 non-null   float64
 3   Unnamed: 3            2 non-null      object
 4   Waist (cm)            3853 non-null   float64
 5   Unnamed: 5            0 non-null      float64
 6   BMI                   3853 non-null   float64
 7   Unnamed: 7            0 non-null      float64
 8   Unnamed: 8            74 non-null     object
 9   Unnamed: 9            1 non-null      object
 10  Unnamed: 10           8490 non-null   object
 11  Unnamed: 11           0 non-null      float64
 12  Unnamed: 12           1959 non-null   object
 13  Unnamed: 13           0 non-null      float64
 14  Unnamed: 14           3916 non-null   object
 15  Unnamed: 15           0 non-null      float64
 16  Fast length (hours)   4156 non-null   object
 17  Unnamed: 17           1 non-null      object
 18  Fast avg. 7 (hours)   4150 non-null   object
 19  Unnamed: 19           0 non-null      float64
...
 66  Unnamed: 66           0 non-null      float64
 67  Unnamed: 67           1 non-null      object
dtypes: float64(43), object(25)
memory usage: 15.9+ MB
```

A change in data, was tracking weight/waist used to be 2 per day, one at morning, and one at night. This wouldn't work with macro/calcoric data, which is tracked one per day, meaning we can only use data where macro data is available, halfing number of cels.



2.2 Data Features/Descriptions:

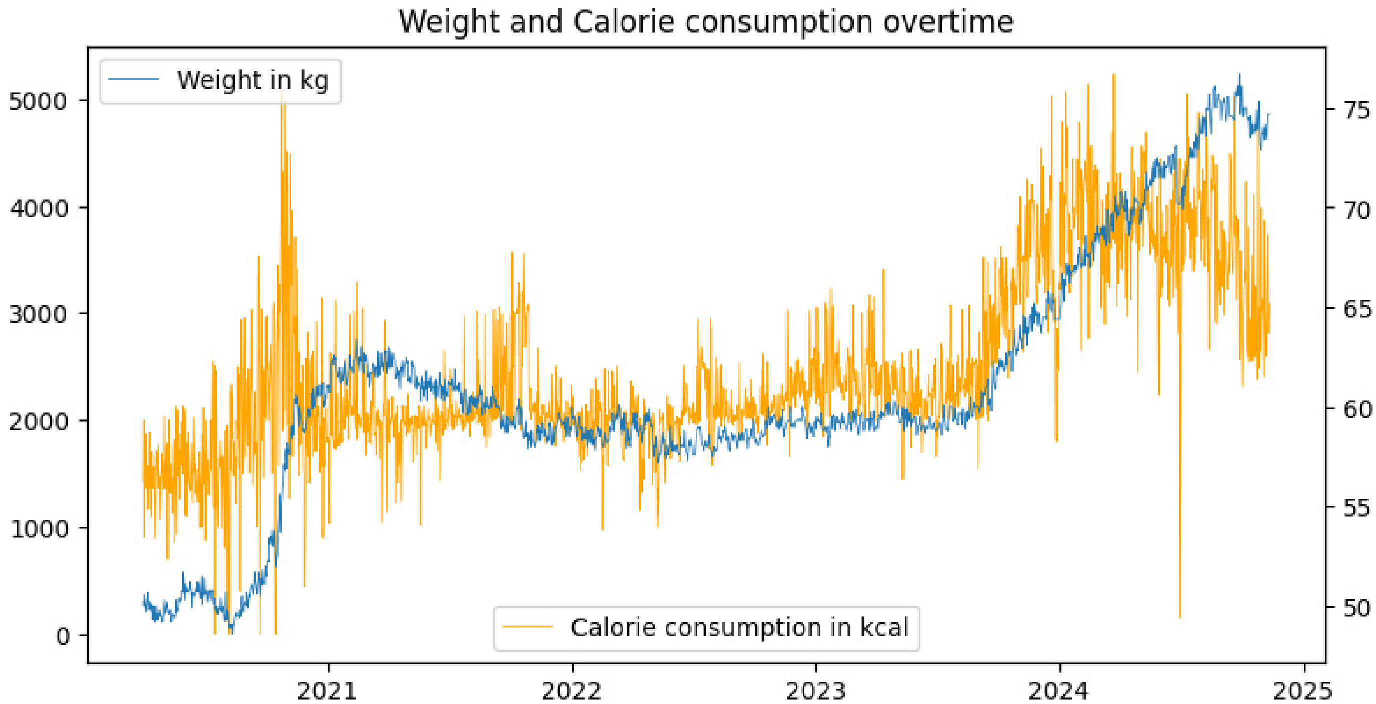
Post data cleaning, the list of features are the weight, waist, caloric intake by day/week average, macro consumption by grams/percentage of calories, and weight/waist difference.

Here's a summary of the list of feature statistics for min, max, standard deviation, and mean. We have 1688 data points in total to use for the model training.

	Date	Weight (kg)	Waist (cm)	Daily intake (kcal)	Kcal avg. week (day)	Carbohydrate (g)	Protein (g)	Fat (g)	Carbohydrate %	Protein %	Fat %	Weight Difference	Waist Difference
count	1688	1688.000000	1688.000000	1688.000000	1688.000000	1688.000000	1688.000000	1688.000000	1688.000000	1688.000000	1688.000000	1687.000000	1687.000000
mean	2022-07-21 11:59:59.999999744	60.912915	67.264502	2480.717417	2477.837438	258.473904	106.511671	111.140604	42.339985	16.806491	40.567946	0.014523	0.008477
min	2020-03-30 00:00:00	48.600000	58.200000	0.000000	1102.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-2.500000	-5.600000
25%	2021-05-25 18:00:00	58.600000	64.600000	1991.000000	2006.357143	184.550000	61.200000	82.100000	34.665653	11.893795	34.012749	-0.300000	-0.400000
50%	2022-07-21 12:00:00	59.600000	66.350000	2187.000000	2199.071429	245.700000	91.650000	105.600000	43.072585	16.402331	40.153975	0.000000	0.000000
75%	2023-09-16 06:00:00	62.300000	70.300000	3001.500000	2890.357143	316.925000	146.800000	138.100000	50.688393	20.810325	47.398633	0.300000	0.500000
max	2024-11-11 00:00:00	76.700000	77.900000	5231.000000	4453.428571	696.600000	374.400000	318.600000	100.000000	57.626769	78.334271	1.500000	4.700000
std	NaN	6.187260	4.024980	840.246992	744.614903	112.001458	60.502474	45.456362	13.190117	6.866947	11.365782	0.469796	0.824128

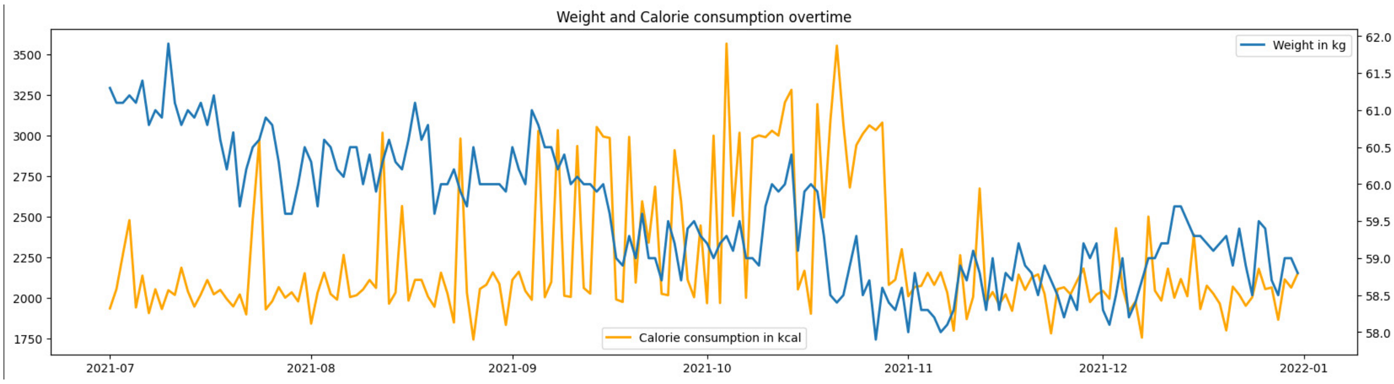
2.3 Data visualization and anomalies:

Visualizing the data, there is a period of time where it outlies most other data surrounding it. An increase of caloric consumption, while maintaining/decreasing in weight. You can find it around Q3 of 2021.



Here's a closer look at the Q3 2021 section of outlier data. The cause in anomaly was due to me, taking DNP2-4. A chemical known for its weight loss effect, and often utilized by bodybuilders for cutting.

This period of data is removed from the training data, as I had external influence that's directly affecting my weight/waist on a section of data.



3 Methodology:

```
yweight_train = yweight_train*100
ywaist_train = ywaist_train*100
yweight_test = yweight_test*100
ywaist_test = ywaist_test*100
```

3.1 Data Preprocessing:

Y data was multiplied by 100. During initial trainings, data hovering at 0 had the model struggling with predictions.

Function for time sequence was created for X. This pulls in a number of indexes from previous days (7 in our case), into every index as extra column data. This allows the model to gain better pattern recognition, through evaluating on the past 7 day trends from our list of features in X. I did accidentally apply it to y data, which caused data leakage.

```
def create_sequences(X, y, window_size):
    Xs, ys = [], []
    for i in range(len(X) - window_size):
        Xs.append(X[i:i+window_size])
        ys.append(y.iloc[i])  # Fix the lag by predicting current time step. It predicts the first value of the window_size, rather than a future value, since we added the window_size to the index.
    return np.array(Xs), np.array(ys)

window_size=7 # size of how far our data will catch trends for (in days)
Xwaist_sequence_train, ywaist_sequence_train = create_sequences(Xwaist_train, ywaist_train, window_size)
Xwaist_sequence_test, ywaist_sequence_test = create_sequences(Xwaist_test, ywaist_test, window_size)
```

Data was normalized with StandardScalar() to ensure the features are in comparable scales, and heavy outliers don't dominate model training.

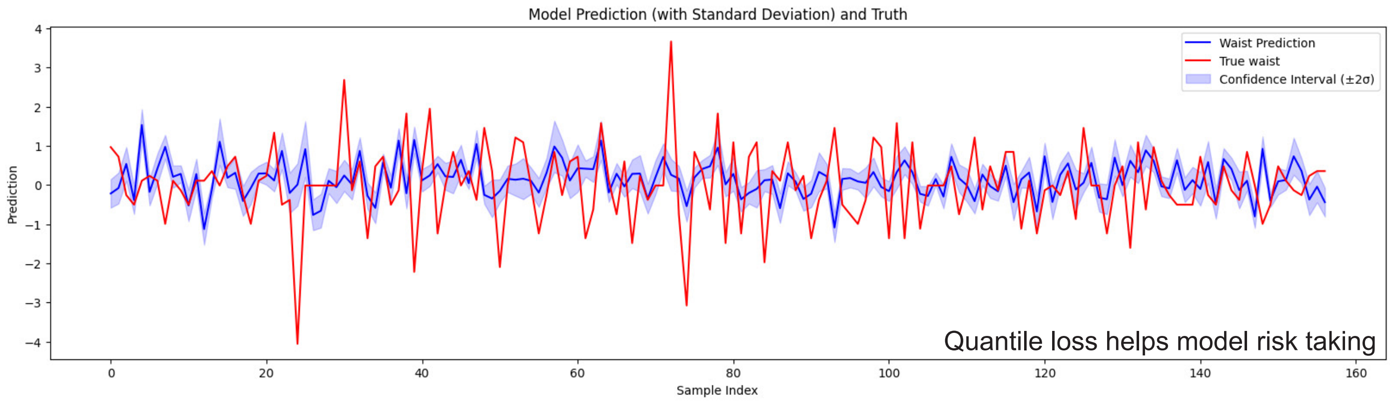
```
Xwaist_sequence_train = Xwaist_sequence_train.reshape(Xwaist_sequence_train.shape[0], -1)
scaler_X_waist = StandardScaler()
normX_waist_train = scaler_X_waist.fit_transform(Xwaist_sequence_train)
scaler_y_waist = StandardScaler()
normY_waist_train = scaler_y_waist.fit_transform(ywaist_sequence_train.reshape(-1, 1))
```

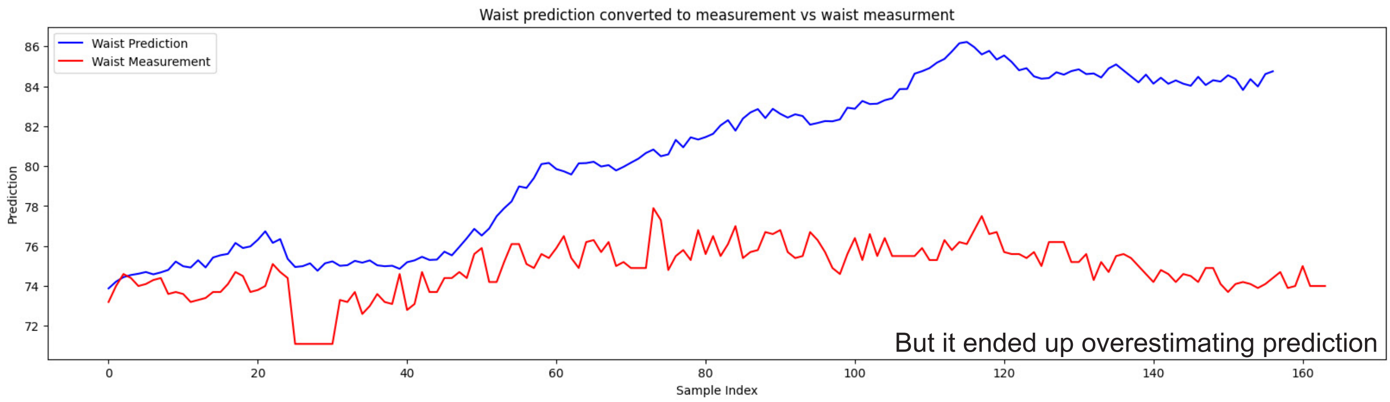
3.2 Model Architecture:
Here’s a layer-by-layer breakdown of model architecture.

Layer	Details
Input	7-day sequenced data that’s been flattened to operate with model properly
GaussianNoise(0.1)	Adding small noise to X-data to regularize
Dense(512) + l2(0.001)	High neuron to add model complexity, and l2 to make model more flexible
BatchNorm + LeakyReLU + Dropout	Norm+Drop helps with model regularization, LeakyReLU works better
Dense(256) - l2	Progressive feature compression
Dense(128) - BatchNorm	Removed BatchNorm to prevent over generalization
Dense(64) - BatchNorm - Dropout	Norm and Dropout removed to prevent over generalization
Output (1)	Funnel everything to a single value to predict weight/waist

3.3 Loss Function, Quantile Loss:
Quantile loss was used in aspect to have model take riskier predictions, due to it sitting around value 0 too often, where higher q loss value looks like the model is taking risk. This had the inadvertent affect, where high q loss, only made the model overpredict on all aspects. It was evident after displaying predictions, where the change in values is added onto the actual weight/waist measurement.

```
def quantile_loss(q):
    def loss(y_true, y_pred):
        e = y_true - y_pred # calculate error differe
        return K.mean(K.maximum(q * e, (q - 1) * e))
    return loss
```





3.4 Learning Rate Strategy, CosineDecayRestarts:
This learning rate strategy, allows the reduction of learning rate overtime to refocus itseld. After an amount of time, the learning ‘restarts’ back up, but learning rate is reduced slightly each time.

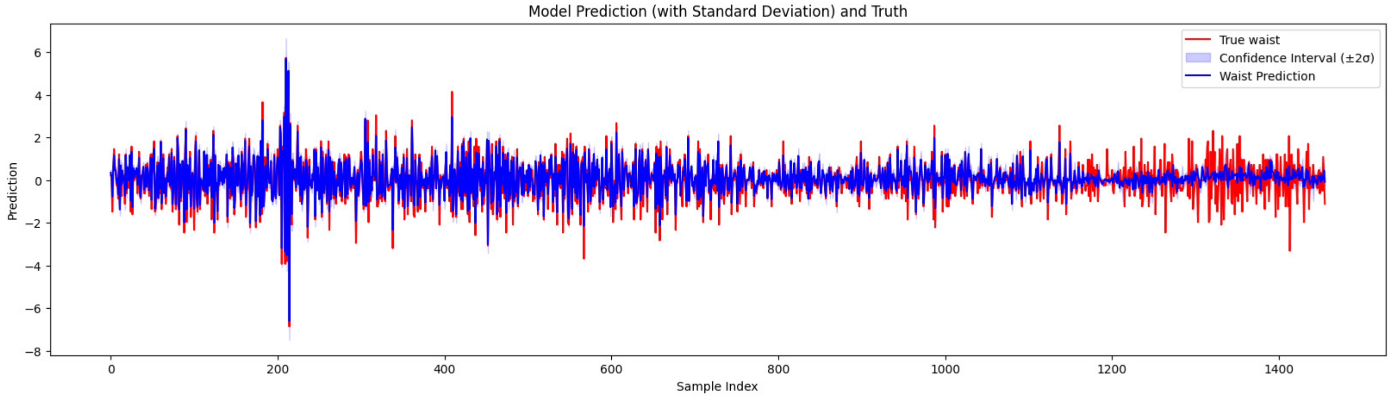
This helps with reducing the chance of model getting stuck in a local minima.

```
lr_schedule = tf.keras.optimizers.schedules.CosineDecayRestarts(
    initial_learning_rate=0.001,
    first_decay_steps=200, # Will start decaying learning rate, after 200 epochs
    t_mul=2.0,  # Reduces decay overtime, multiplying by 2 on the first decay steps (200*2 = 400 is the new epoch to decay)
    m_mul=0.8,  # Reduces peak LR over time, where our start is 0.001, multiply by 0.8, gives us 0.0008 for new start of lr
    alpha=1e-6  # Minimum LR
)
```

3.5 Experimental X-train Design:

Full dataset model

The first try on the model, was done on the entirety of the train dataset. After visualizing the predictions through measurements rather than the change in weight/waist data. Investigating what causes the model to overgeneralize but predict so high in value, revealed in training data. My change in sedentary lifestyle to active lifestyle created a problem for the model. I didn't have any indicators that track my active hours, steps etc. Only calories and macronutrients, so the model had no clue of such change, and treated the numbers as if I was still in my sedentary lifestyle.



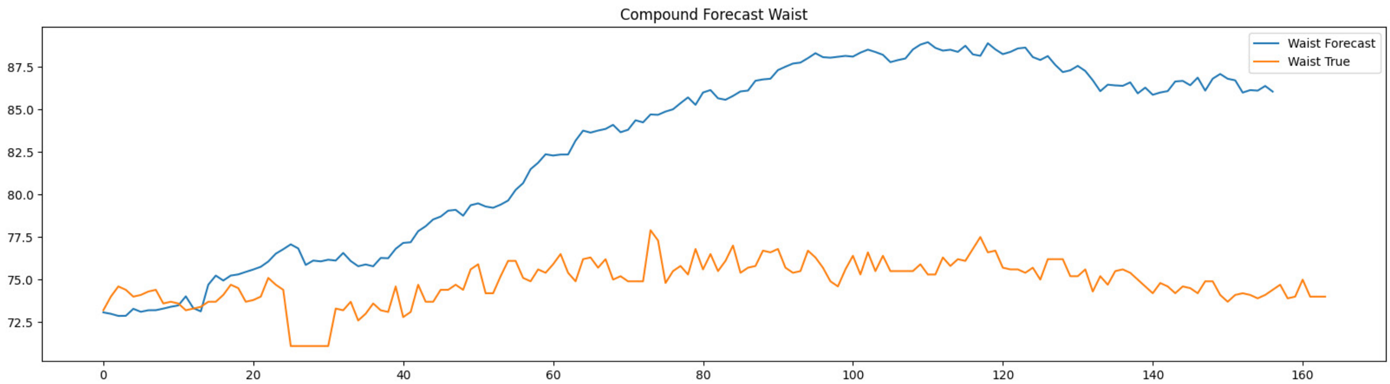
Where you see the model starts underpredicting all values in the train data set, is where the change in lifestyle began.

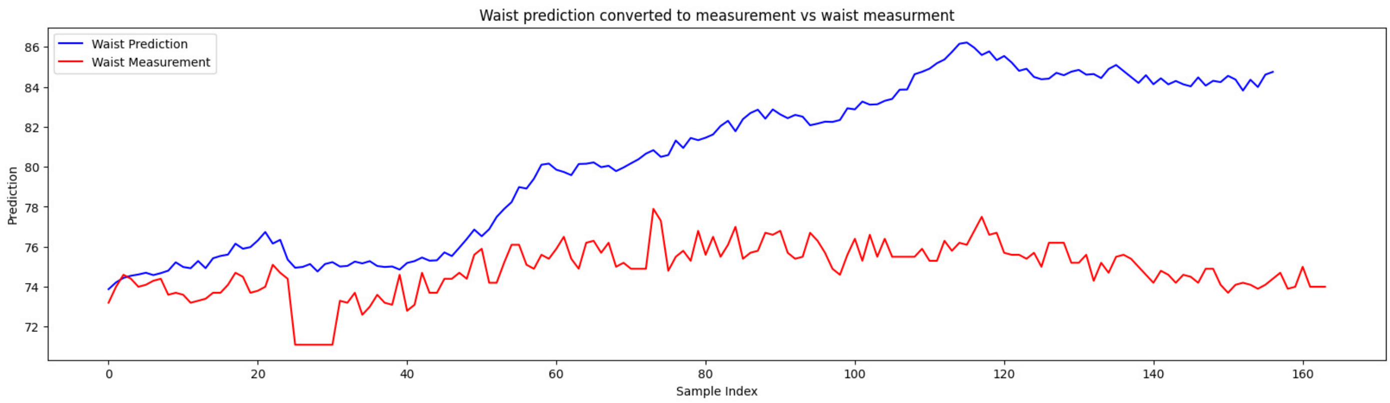
Active lifestyle model

From the results above, I tried retraining the model only on the active lifestyle portion of data, from index 1150, cutting the majority of training data out. Having the model learn only the active lifestyle portion generated favourable results.

3.6 Forecast Weight and Waist Function:

I tried visualizing prediction by adding the differences on top of the first weight and waist data. Then I wanted to create a function, where the prediction of weight and waist is changed inside the X data. Aka a form of forecasting function, using previous model's prediction to make another prediction, where prediction compounds on top of each other. The addition version was not accurate to reality. This is because, the model has access to true weight and waist values after each prediction it's made. Naturally, you won't have access to measurment data, if you're predicting an entire portion of data. So, you have to rely on your previous predictions, and make another guess from that.





The model does a pretty good job when predicting only by it's predictions, compared to having access to true weight.

3.7 Last bit of changes (Q loss):

I lacked knowledge on Quartile Loss, and learnt that it predicts such as the upper quartile of the data spread for high q loss, or vice versa for low q loss. I made changes to q loss that better fits with each weight and waist model.

4 Results:

4.1 Performance Metrics:

I created a custom evaluation called MAPE (Mean Average Percentage Error). Typical MSE and R^2 don't help too much with evaluation, due to the prediction values are small around 0. MAPE states the absolute percentage of how far the prediction is from truth. If prediction is 1, and truth is 2, the difference is 50%.

```
('Weight Evaluation',
 {'Test MAPE (%)': np.float64(206.4920772698088),
  'Test MSE': 1.77812658512935,
  'Test R^2': -0.2552423856101027})
```

```
('Waist Evaluation',
 {'Test MAPE (%)': np.float64(224.53574301705336),
  'Test MSE': 1.602273135015225,
  'Test R^2': -0.18429371657718363})
```

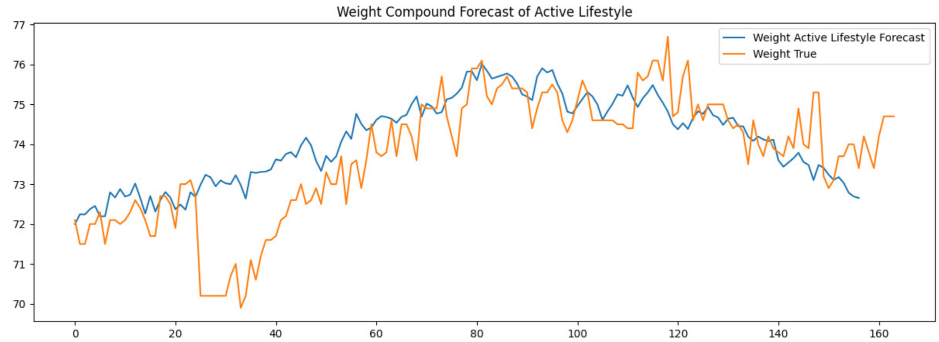
4.2 Graph between prediction and truth:

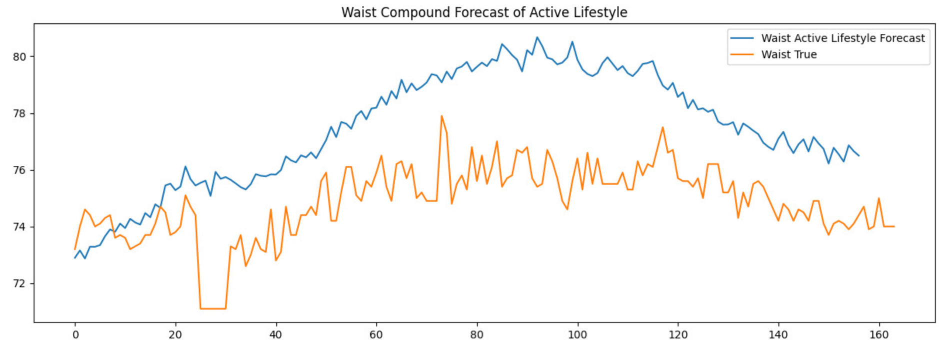
This is the line plot of the best performing models. Trained on the active lifestyle portion of data, and tweaked q loss to better fit each.

Along with its comparison to the true values of weight and waist measurements in orange.

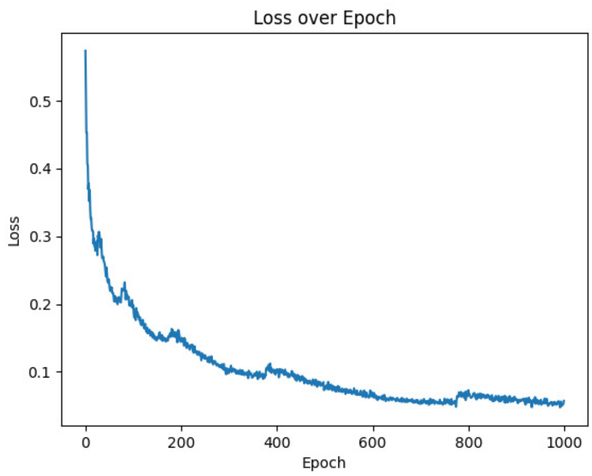
Its evident that the prediction generalizes values, and doesn't have the peaks and troughs of the true data. However makes the best overall accurate predictions.

Only thing I'm unsure is, whether the q loss tailoring would only work with this portion of data, and that it would not function well on a new portion of test data.





5 Analysis and Interpretation:

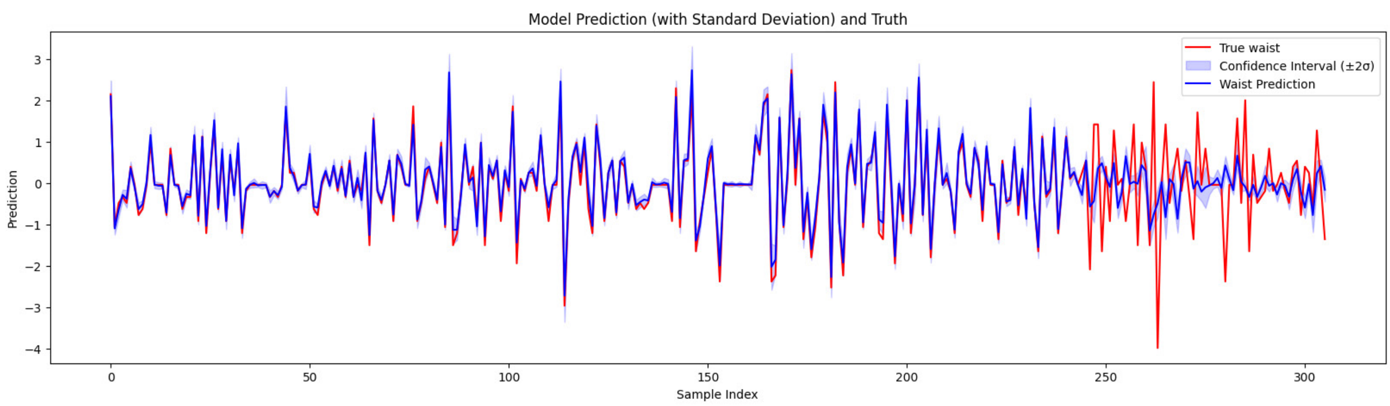


5.1 Model Performance:

During training, the model does a great job at learning patterns of train data.

Predicting the training data withthe model, shows how accurate it can predict, however it still had the same issue with generalizing too much near the last portion of training data, and not take the appropriate risks.

I hypothesise it could be another change in lifestyle/diet, where I did cutting near the end portion of training data. Or either the model did not have enough time to learn every pattern just yet.



5.2 Prediction analysis:
The models seem to be most sensitive to changes in lifestyle. When the model gets to the point where lifestyle has changed, it looks like the model tend to heavily smooth the results and hover much closer to 0, likely due to being unfamiliar with the new lifestyle.

Between active lifestyle model, and full model, the predictions between the two on the weight/waist difference similarly overgeneralize. But, active lifestyle model does a better job with predicting the general patterns of weight/waist measurements than full model. Which often over predicts weight and waist, likely due to being more acquainted to the sedentary lifestyle portion of data, and using it as a frame of reference. (And Q loss value could be another factor.)

5.3 External influences:
There's definitely a ton of different external influences to the change of weight and waist by day. Calorie and macro count is only one piece of the puzzle. One of the most notable external influence is physical activity/lifestyle changes as stated multiple times. Since physical activity plays an affect of how much calories one consume.

Other factors beyond that can also be water, sodium intake (which cause water retention), timing of food intake, body fat percentage, muscle mass, types of supplements etc. etc.

6 Reflections and Learnings:

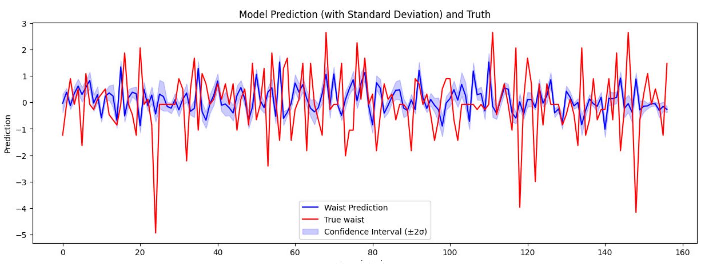
6.1 Discoveries:
I discovered that lifestyle change makes such a strong impact on model performance, as the changes are seen immediately through the model's cautiousness. No matter how many iterations of model was done (over 100), it was still stuck in that over generalization prediction. Plus, the training on sedentary lifestyle, played into the model's overestimation in test dataset, due to how much calories and macros I was consuming during my active lifestyle. It lacked any information regarding physical activity/lifestyle to help it understand why I was eating so much, but not gaining weight.

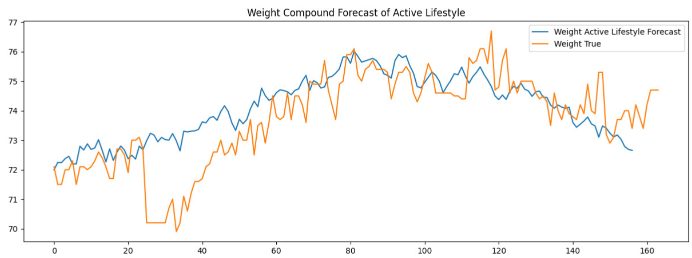
6.2 Learnings:
I've tried RNN's (Recurrent Neural Network), designed for processing sequential data, e.g. a period of time to collectively make a prediction. I thought complexity in models, will lead to better predictions. But that wasn't the case, as it flattens out randomly during training, and refuses to learn. There's a lot of complexity to model creation, and the stuff to implement upon it. I feel like I'm scratching the surface with creating models and engineering it to best optimize weight/waist prediction.

Organization is something I struggled on during this project. Due to sorting out weight, waist, train, test, then full model and active lifestyle model, there was so many reused code where I replaced the variable names. It lead to a lot of times where I get the wrong results, or forgetting which variable name is for which.

I learnt how hyperfixated I get on methods, etc. Such as RNN, where I spent a lot of time trying to make it, despite ReLU model from previous tests, didn't suffer the flattening issue that RNN had. Or hyperfixating on getting the model to take as much risks on predicting weight/waist difference, when it could've proven useful to look at the compounding measurements, and notice the heavy overestimation sooner, so then I can correct it earlier.

Example, the weight difference doesn't provide much visual clues, when compared to compounded forecast.





6.3 What would I do differently?:
I would've tracked the hours I spent on exercise on the day, along with the step count. Maybe even the mean average heart rate during exercise. Few extras could be sodium, as it's a data point that was available to me from the food calorie counting app I use.

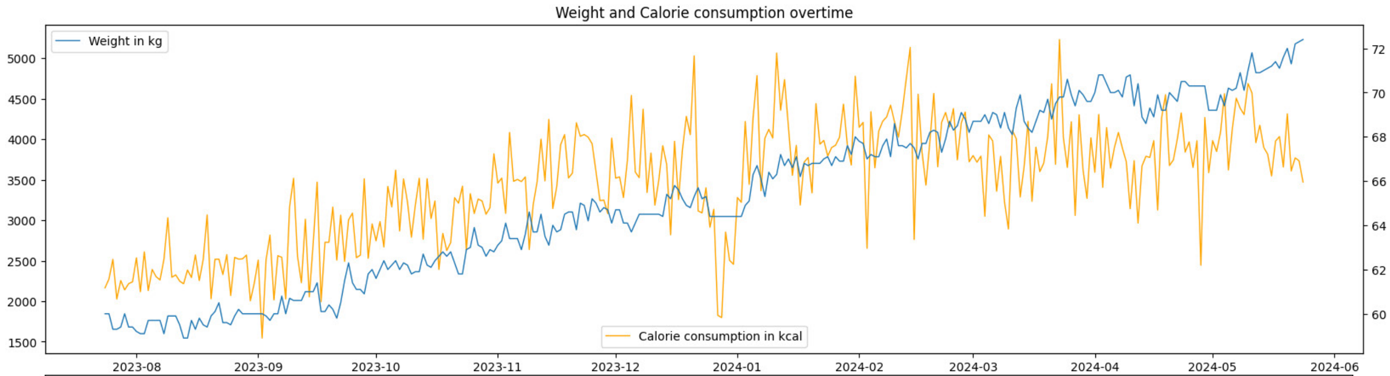
I did prediction based on weight/waist difference. Maybe next time, I could try predicting the weight/waist measurment itself. I initially avoided it due to data leakage, but moving the values so that it predicts the future measurement would solve that issue. Since that was how I solved the weight/waist difference in y data.

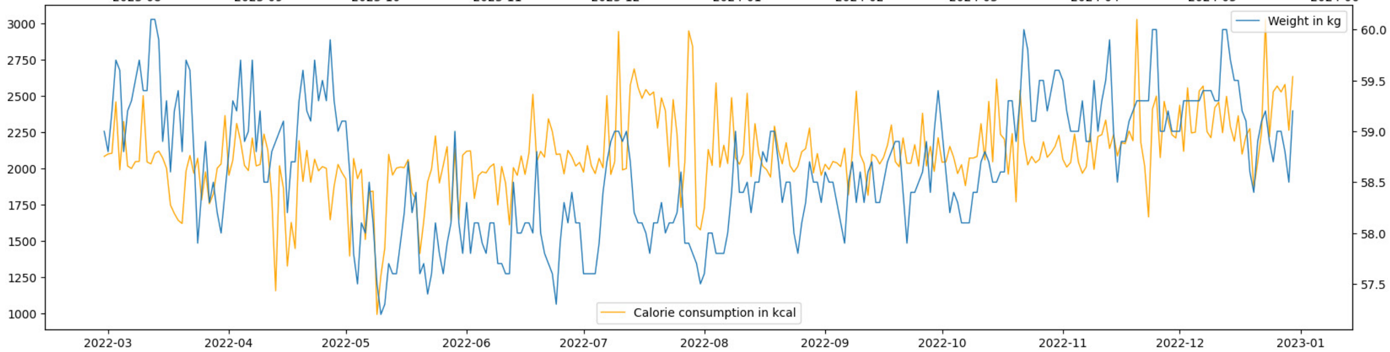
I may simply just need more data points, and track for an even longer time.

7 Appendix:

Here are some extra graphs and visuals, that give a bit more detail on the project.

The graphs underneath, shows the difference between active lifestyle, and sedentary lifestyle. The increase in calories show steady incline in weight, and starts to even out as it progresses. For the sedentary lifestyle, caloric consumption stays relatively stable, but body weight flutuates up and down.





This is the function, used to create the forecasting of weight and waist model. It goes through a loop, updating weight and waist everytime into the X dataset using their predictions. Allowing the model to predict upon it's prediction weight and waist.

```python
# Create sequential, autoregressive prediction loop
def simulation_loop(model_weight, model_waist, norm_X, id_waist, id_weight, X_scaler,
scaler_y_weight, scaler_y_waist, start_weight, start_waist, steps_ahead):
    """
    Forecasting weight and waist values through compounding predictions

    Parameters:
    model_weight, model_waist: trained models
    norm_X: standardized test set
    id_waist, id_weight: indexed location of where waist and weight is located in X_test dataset
    X_scaler, scaler_y_weight, scaler_y_waist: scalers for each respective datasets, to either fit or inverse transform when needed
    start_weight, start_waist: starting value of forecast
    steps_ahead: number of steps to predict/forecast
    """
    current_weight = float(start_weight) # make it float, as to not get rid of decimal predictions
    current_waist = float(start_waist)
    predictions = [] # keep new list for all predictions

    # unstandardize x_test
    X_test = X_scaler.inverse_transform(norm_X)

    for i in range(steps_ahead):
        # prepare current index of x_test in loop to a new variable
        x = X_test[i].copy()

        # replace real measurements with predictions
        x[id_weight] = current_weight
        x[id_waist] = current_waist

        # reattach the current index row to the test dataset
        X_test[i] = x

        # scale test for model training
        norm_X = X_scaler.transform(X_test)

        # predict weight and waist
        weight_pred_scaled = model_weight.predict(norm_X, verbose=0)
        waist_pred_scaled = model_waist.predict(norm_X, verbose=0)

        # inverse scaling of y predictions (including the *100 done on data)
        weight_pred = scaler_y_weight.inverse_transform(weight_pred_scaled)[i][0]/100
        waist_pred = scaler_y_waist.inverse_transform(waist_pred_scaled)[i][0]/100

        # update values for next loop
        current_weight += weight_pred
        current_waist += waist_pred

        predictions.append((current_weight, current_waist))

    return predictions
```

This is the full model of weight and waist model. The only difference between the two, is weight uses q loss at 0.65, and waist uses 0.7.

```python
Xweight_sequence_train_active = Xweight_sequence_train_active.reshape(Xweight_sequence_train_active.shape[0], -1)
scaler_X_weight = StandardScaler()
normX_weight_train_active = scaler_X_weight.fit_transform(Xweight_sequence_train_active)
scaler_y_weight = StandardScaler()
normY_weight_train_active = scaler_y_weight.fit_transform(yweight_sequence_train_active.reshape(-1, 1))

# Define model
modelweight_active = keras.Sequential([
    layers.GaussianNoise(0.1, input_shape=(normX_weight_train.shape[1],)),

    layers.Dense(512, kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.LeakyReLU(alpha=0.1),
    layers.Dropout(0.1),

    layers.Dense(256),
    layers.BatchNormalization(),
    layers.LeakyReLU(alpha=0.1),
    layers.Dropout(0.1),

    layers.Dense(128),
    layers.LeakyReLU(alpha=0.1),
    layers.Dropout(0.1),

    layers.Dense(64),
    layers.LeakyReLU(alpha=0.1),

    layers.Dense(1)
])

lr_schedule = tf.keras.optimizers.schedules.CosineDecayRestarts(
    initial_learning_rate=0.001,
    first_decay_steps=200, # Will start decaying learning rate, after 200 epochs
    t_mul=2.0,  # Reduces decay overtime, multiplying by 2 on the first decay steps (200*2 = 400 is the new epoch to decay)
    m_mul=0.8,  # Reduces peak LR over time, where our start is 0.001, multiply by 0.8, gives us 0.0008 for new start of lr
    alpha=1e-6  # Minimum LR
)

def quantile_loss(q):
    def loss(y_true, y_pred):
        e = y_true - y_pred # calculate error difference between pred and truth
        return K.mean(K.maximum(q * e, (q - 1) * e)) # formula to return the loss, deterministic by q value
    return loss

modelweight_active.compile(optimizer = Adam(learning_rate=lr_schedule),
                loss=quantile_loss(0.65),
                metrics=['mae'])

histweight_active = modelweight_active.fit(normX_weight_train_active, normY_weight_train_active,
                            validation_split=0.2,
                            epochs=1000, verbose=0)
```