

# TensorFlow

# Time series fundamentals

+

# Milestone Project 3

BitPredict  



# This is not financial advice

(Spoiler: by the end of this module you'll see where even deep learning models fail to predict in open systems)

# Where can you get help?

- Follow along with the code



```
10. Milestone Project 3: Time series forecasting in TensorFlow (BitPredict)

The goal of this notebook is to get you familiar with working with time series data. We're going to be building a series of models in an attempt to predict the price of Bitcoin.

Welcome to Milestone Project 3, BitPredict! 🚀

⚠ Note: This is not financial advice, as you'll see time series forecasting for stock market prices is actually quite terrible.

What is a time series problem?
Time series problems deal with data over time. Such as, the number of staff members in a company over 10 years, sales of computers for the past 5 years, electricity usage for the past 50 years. The timeline can be short (seconds/minutes) or long (years/decades). And the problems you might investigate using can usually be broken down into two categories.

Classification: "Which of these points is an anomaly?"  
Forecasting: "How much will the price of Bitcoin change tomorrow?"
```

- Try it for yourself



"If in doubt, run the code"

- Press SHIFT + CMD + SPACE to read the docstring

```
# Let's get TensorFlow!
import tensorflow as tf

# MASE implemented could
def mean_absolute_scale(*args, **kwargs):
    """Implement MASE (assuming no seasonality)
    mae = tf.reduce_mean(tf.abs(y_true - y_pred))

    # Find MAE of naive forecast (no seasonality)
    mae_naive_no_season = tf.reduce_mean(tf.abs(y_true[1:] - y_true[:-1])) # our seasonality is 1 day

    return mae / mae_naive_no_season
```

Since we're going to be evaluating a lot of models, let's write a function to help us calculate evaluation metrics on their forecasts.

First we'll need TensorFlow.

[20]:

```
def reduce_mean(input_tensor, axis=None, keepdims=False, name=None)
```

And since TensorFlow doesn't have its own?

Reduces `input_tensor` along the dimensions given in `axis` by computing the mean of elements across the dimensions in `axis`. Unless `keepdims` is true, the rank of the tensor is reduced by 1 for each of the entries in `axis`, which must be unique. If `keepdims` is true, the reduced dimensions are retained with length 1.

We'll take inspiration from `sklearn`:

```
# MASE implemented could
def mean_absolute_scale(*args, **kwargs):
    """Implement MASE (assuming no seasonality)
    mae = tf.reduce_mean(tf.abs(y_true - y_pred))

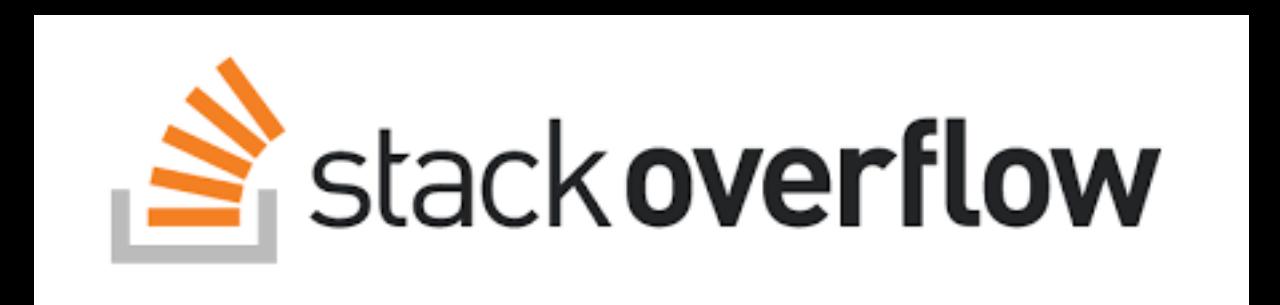
    # Find MAE of naive forecast (no seasonality)
    mae_naive_no_season = tf.reduce_mean(tf.abs(y_true[1:] - y_true[:-1])) # our seasonality is 1 day

    return mae / mae_naive_no_season
```

You'll notice the version of MASE above doesn't take in the training values like sktime's `mae_loss()`. In our case, we're comparing the MAE of our predictions on the test to the MAE of the naive forecast on the test set.

In practice, if we've created the function correctly, the naive model should achieve an MASE of 1 (or very close to 1). Any model worse than the naive forecast will achieve an MASE of >1 and any model better than the naive forecast will achieve <1.

- Search for it



- Try again

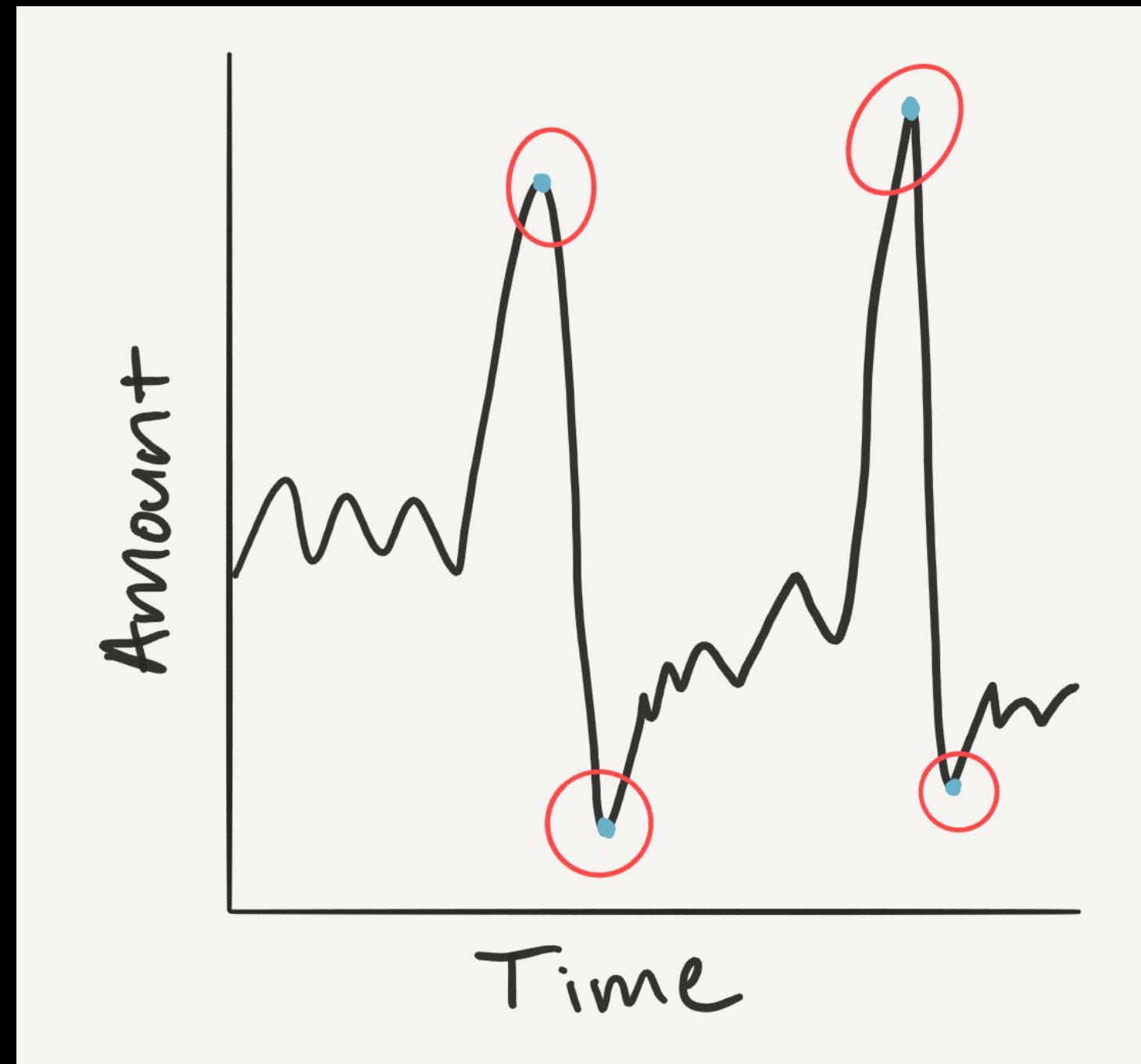
- Ask (don't forget the Discord chat!)

(yes, including the “dumb” questions)

# Example Time Series Problems

## Classification

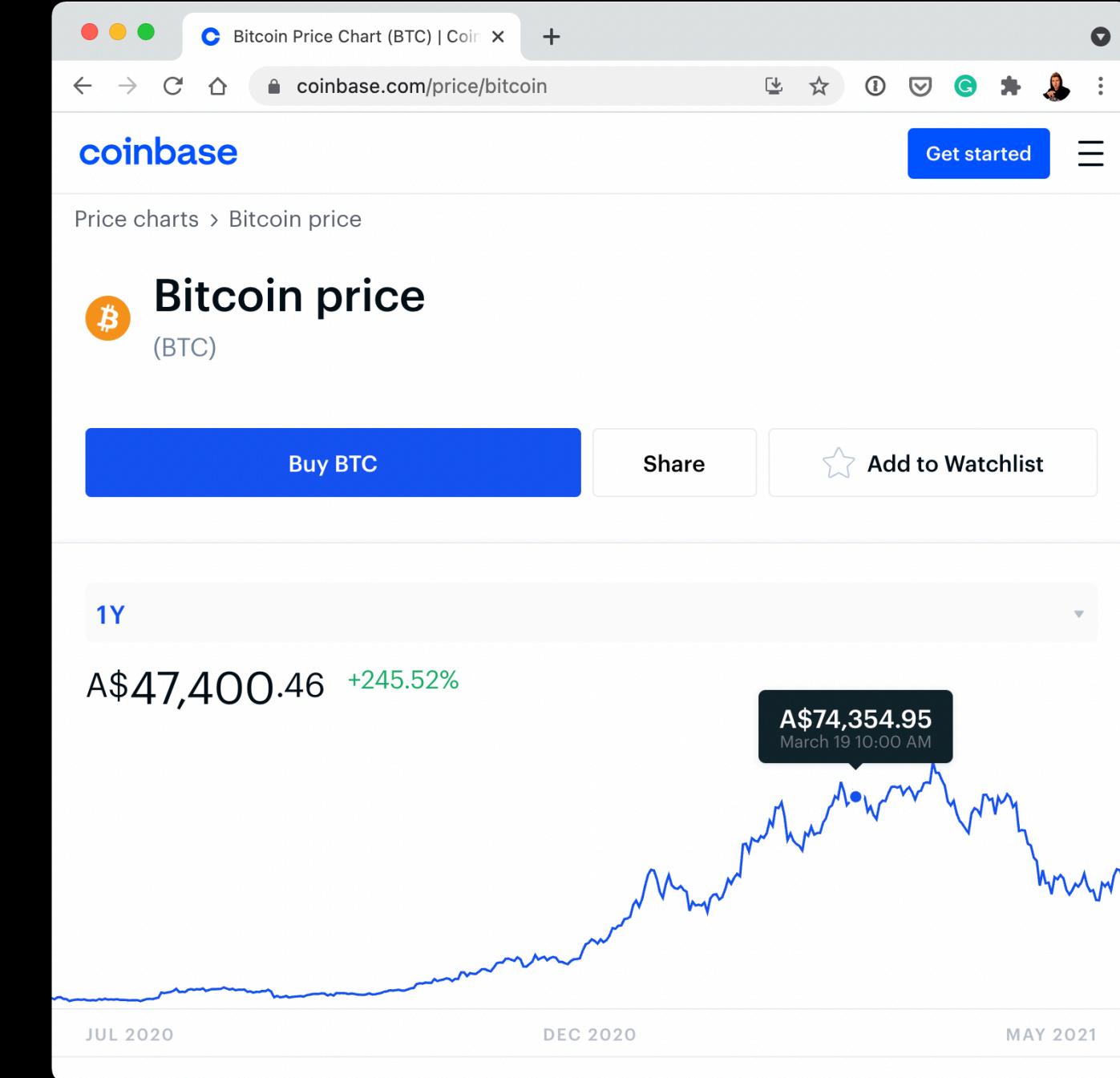
“Which of these points is an anomaly?”



Temporal (time)  
component

## Forecasting

“How much will the price of Bitcoin change tomorrow?”



“What electronic device is this?”

“Are these heartbeats regular?”

Output: discrete

“How many computers will we sell next year?”

“How many staff do we need for next week?”

Output: continuous

# Example Forecasting Problems at Uber



Market demand

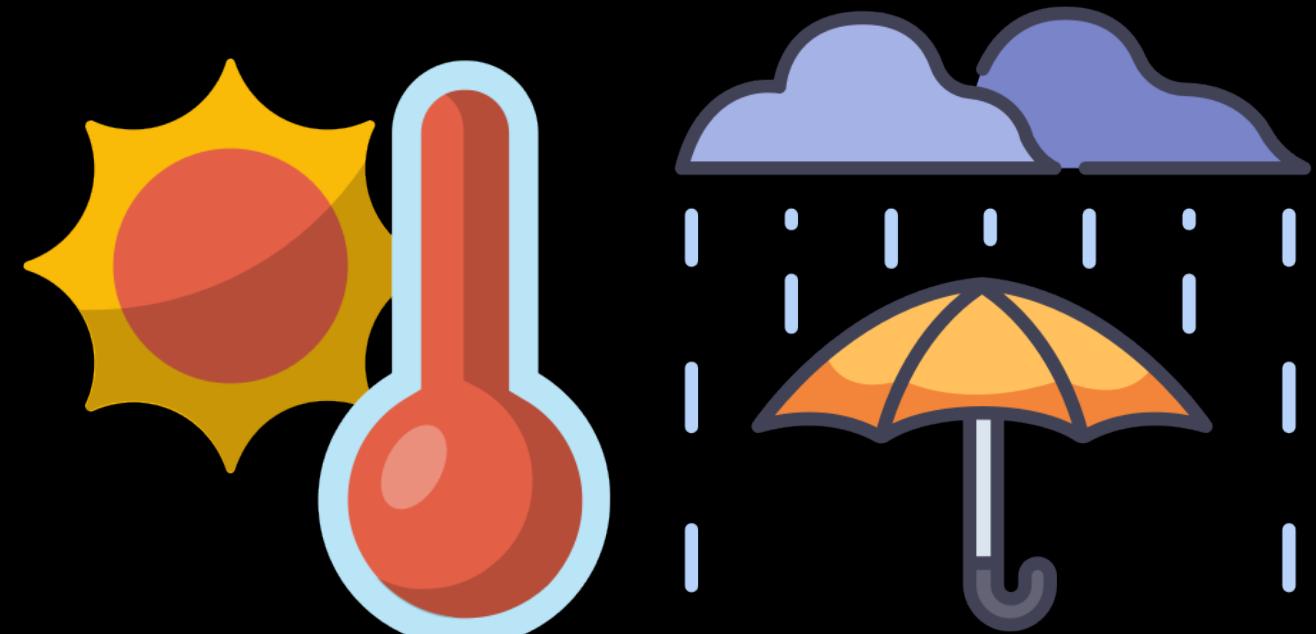


Compute requirements



Marketing campaigns

# Example Forecasting Problems in daily life



TECH \ TRANSPORTATION \ CARS

Lyft's president says 'majority' of rides will be in self-driving cars by 2021

*Also say bye-bye to personal car ownership*

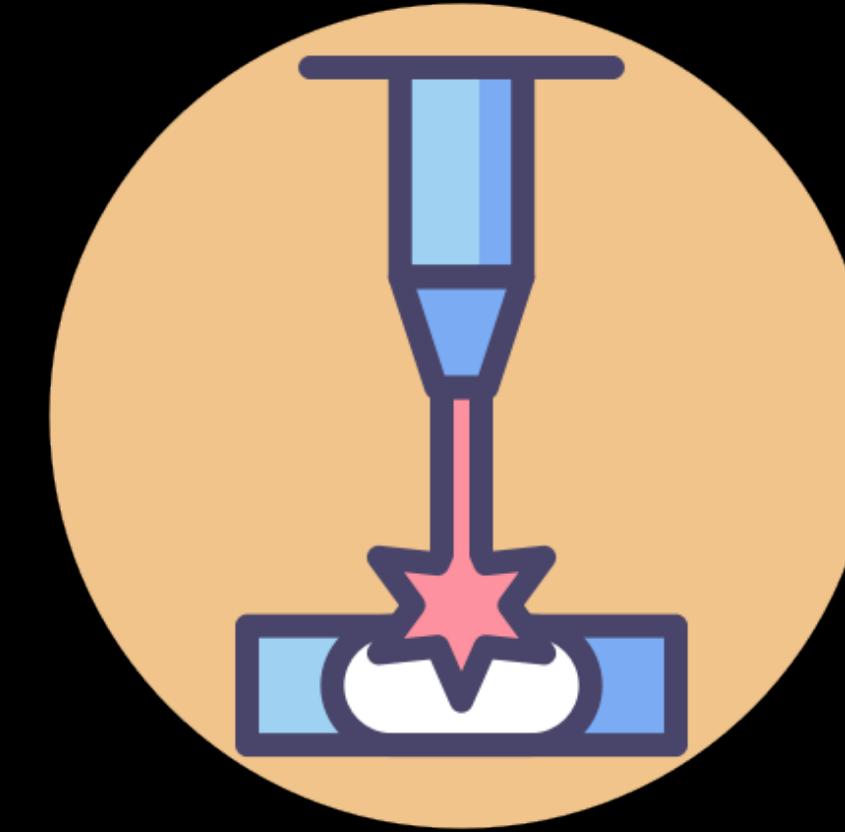
By Andrew J. Hawkins | @andyjayhawk | Sep 18, 2016, 9:15am EDT

“I think there is a world market for maybe five computers.”

— Thomas Watson, president of IBM, 1943

(there are many more of these)

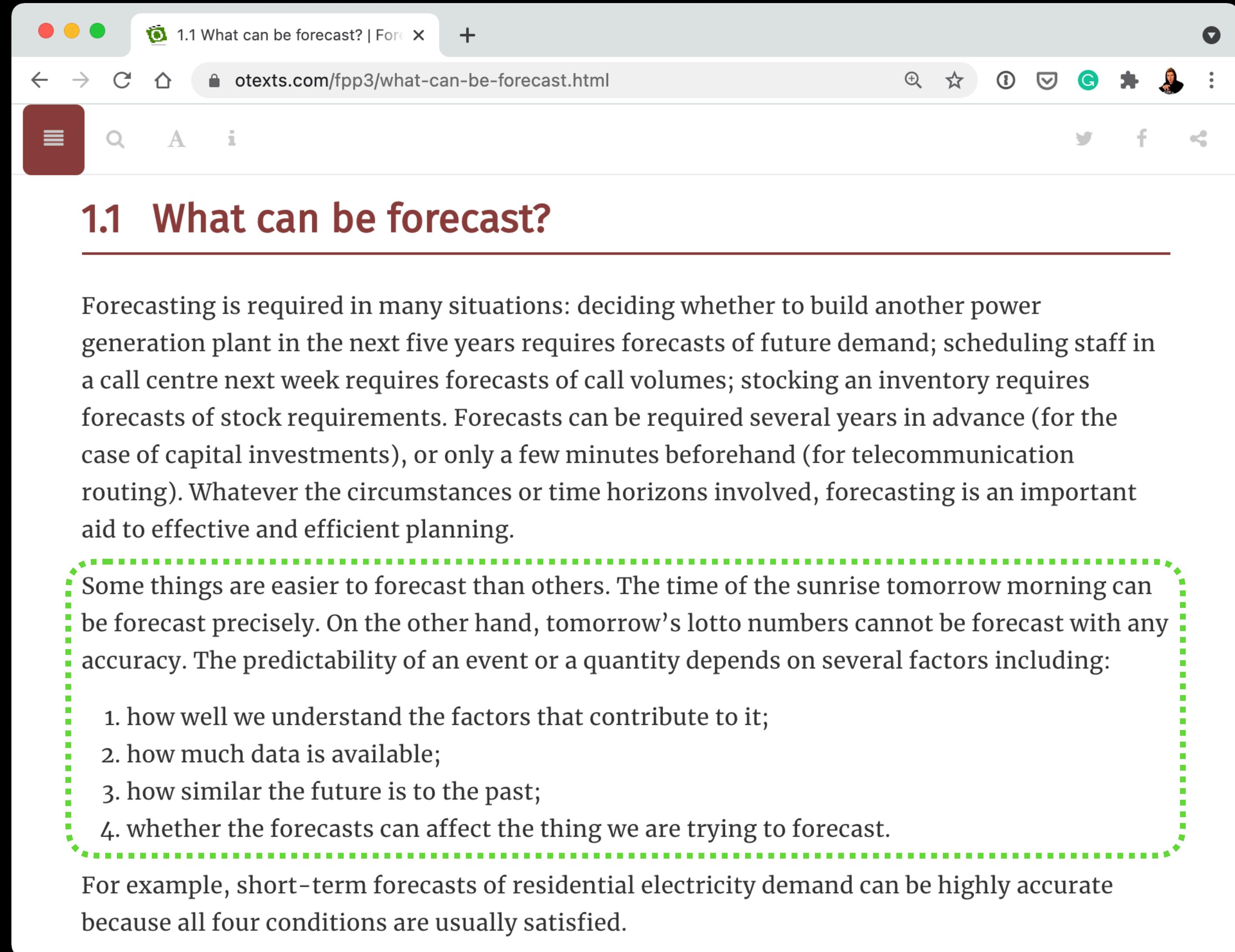
# What can we predict?



What do these things have in common?

(Massive impact but unplanned, unpredicted)

# What can we predict?



The screenshot shows a web browser window with the title "1.1 What can be forecast? | Fore" and the URL "otexts.com/fpp3/what-can-be-forecast.html". The page content is as follows:

## 1.1 What can be forecast?

Forecasting is required in many situations: deciding whether to build another power generation plant in the next five years requires forecasts of future demand; scheduling staff in a call centre next week requires forecasts of call volumes; stocking an inventory requires forecasts of stock requirements. Forecasts can be required several years in advance (for the case of capital investments), or only a few minutes beforehand (for telecommunication routing). Whatever the circumstances or time horizons involved, forecasting is an important aid to effective and efficient planning.

Some things are easier to forecast than others. The time of the sunrise tomorrow morning can be forecast precisely. On the other hand, tomorrow's lotto numbers cannot be forecast with any accuracy. The predictability of an event or a quantity depends on several factors including:

1. how well we understand the factors that contribute to it;
2. how much data is available;
3. how similar the future is to the past;
4. whether the forecasts can affect the thing we are trying to forecast.

For example, short-term forecasts of residential electricity demand can be highly accurate because all four conditions are usually satisfied.

**Source:** Forecasting: Principles and Practice 3rd edition chapter 1.1

# What we're going to cover (broadly)

- Downloading and formatting time series data (the historical price of Bitcoin)
- Writing a **preprocessing function** to prepare our time series data
- Setting up **multiple time series modelling experiments**
- Building a **multivariate model** to take in multivariate time series data
- Replicating the N-BEATS algorithm using TensorFlow
- Making forecasts with **prediction intervals**
- Demonstrating **why time series forecasting can be BS** with the turkey problem 

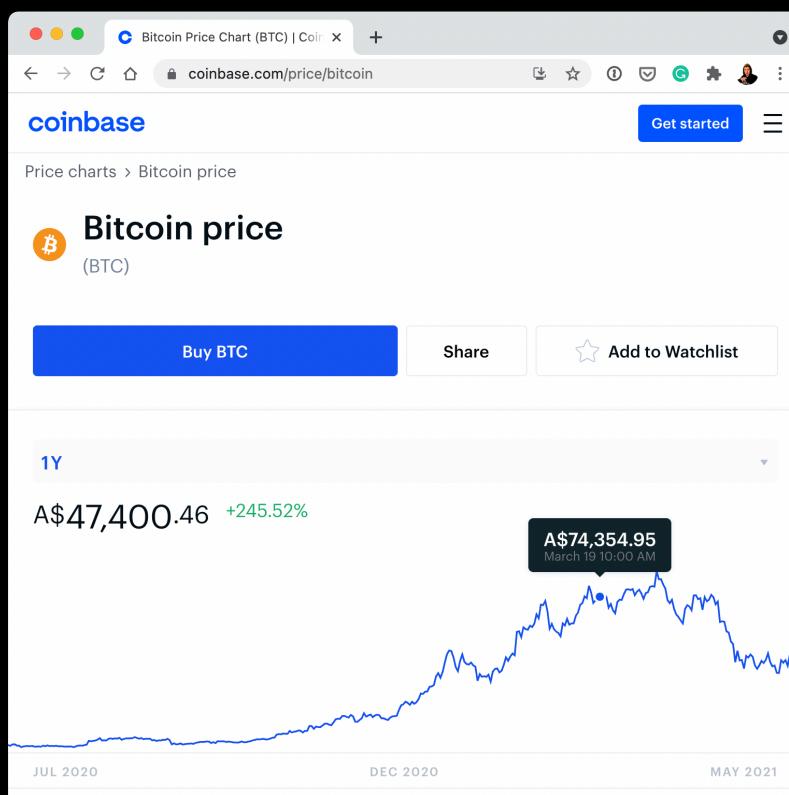
(we'll be cooking up lots of code!)

**How:**



# Time series inputs and outputs

(Forecasting wise)

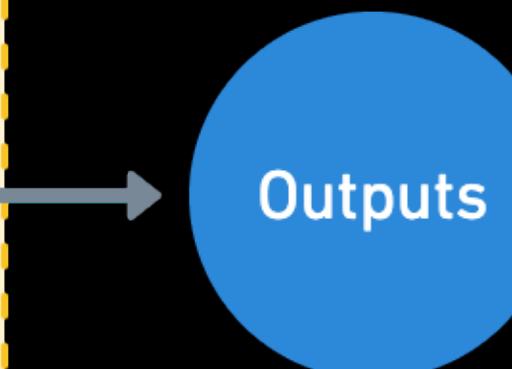
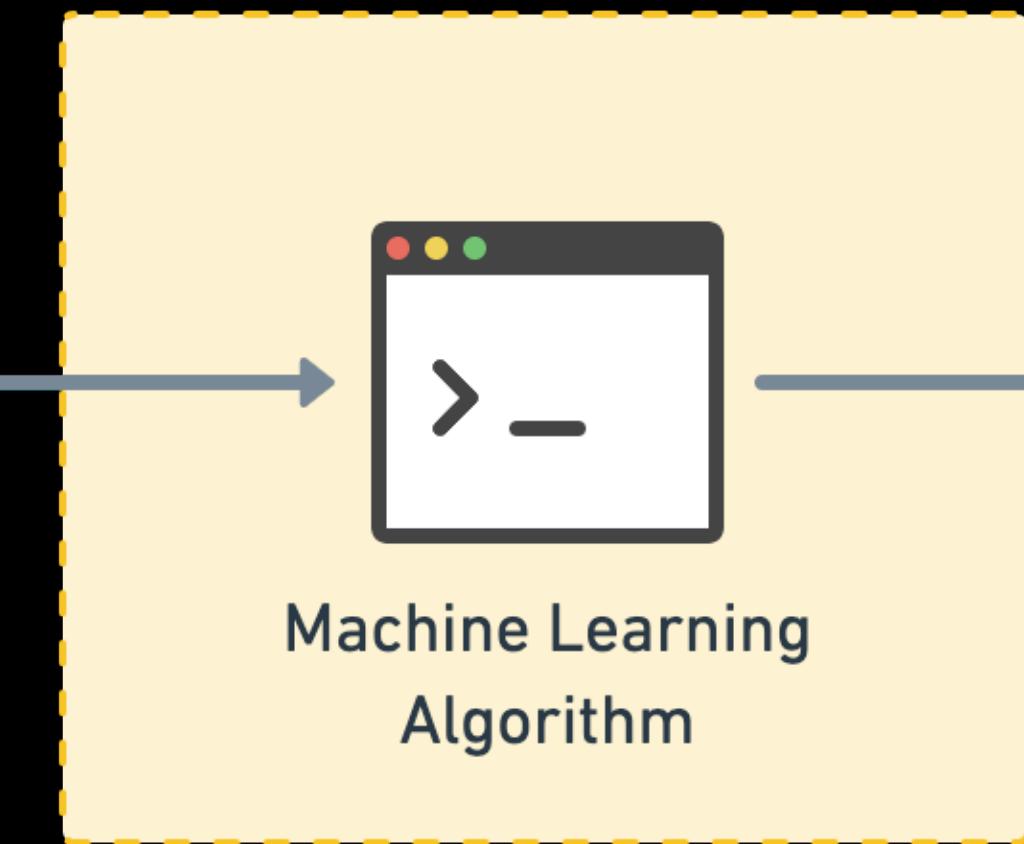


“What will the price of Bitcoin be tomorrow?”

[ [37888, 39124, 40563...],  
[39124, 40563, 42458...], →  
[40563, 42458, 44899...],  
...,

## Numerical encoding

(Time series data is often already numerical)



[ [40669] ,  
[43443] ,  
[46986] ,  
...,

## Predicted output

(comes from looking at lots of these)

(often already exists, if not, you can build one)

BitPredict

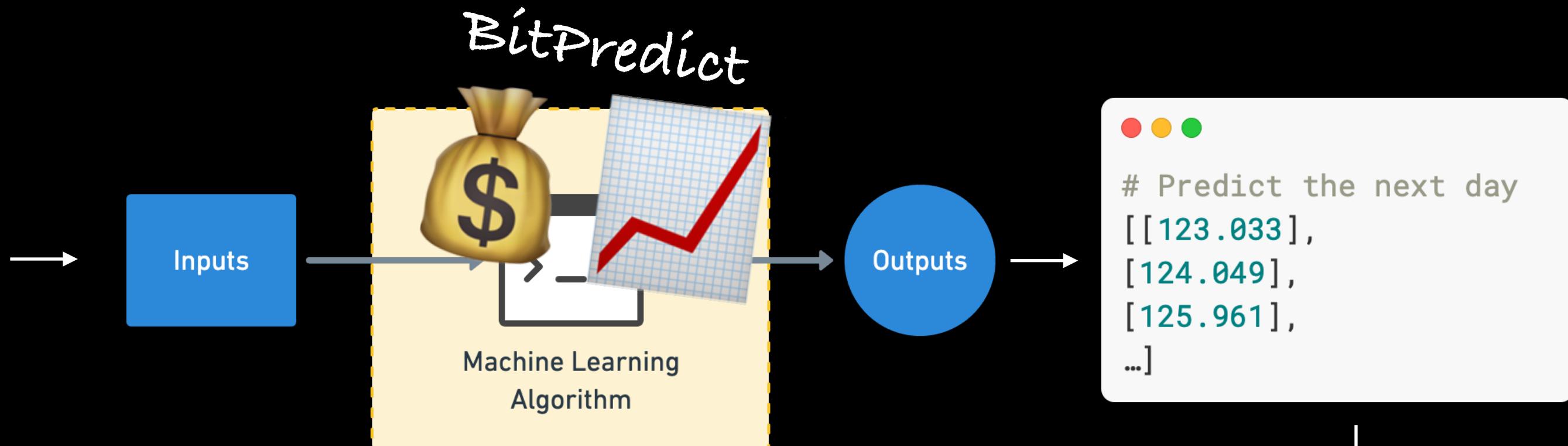


A\$47,744

Actual output

# BitPredict input and output shapes

```
# Window for one week  
[[123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795],  
 [125.455, 108.584, 118.674, 121.338, 120.655, 121.795, 123.033],  
 [108.584, 118.674, 121.338, 120.655, 121.795, 123.033, 124.049],  
 ...]
```



[batch\_size, window\_size]

Shape = [None, 7]

or

Shape = [32, 7]

(32 is a very common batch size)

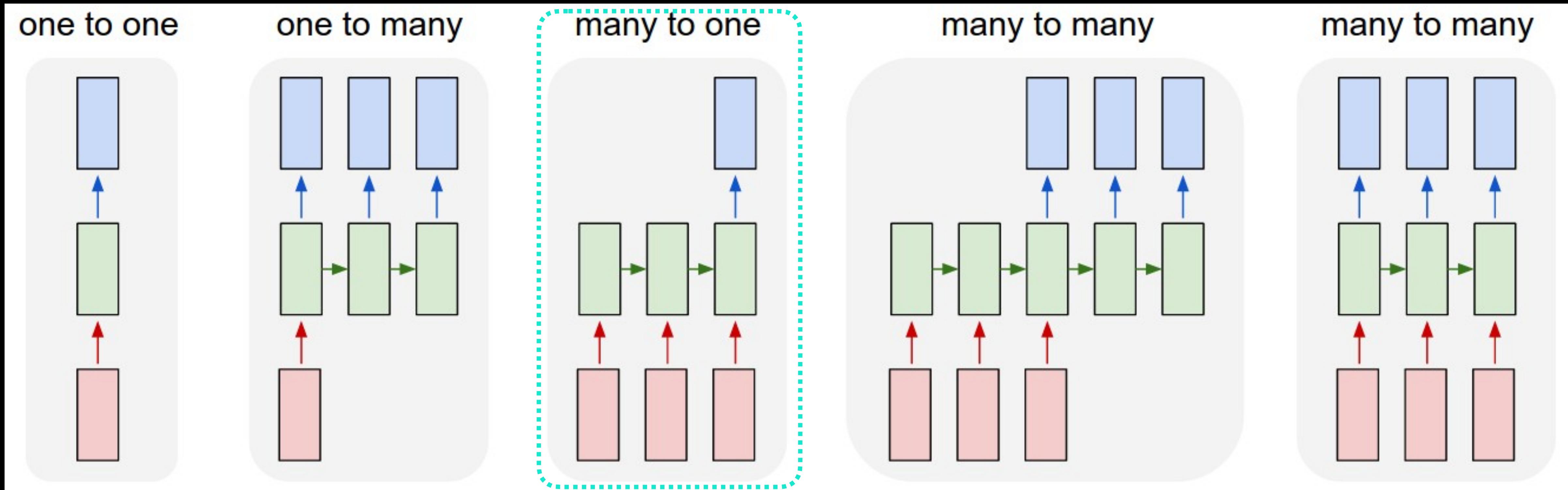
These will vary depending on the problem you're working on/what window size & horizon you decide to use.

[horizon]

Shape = [1]

# Sequence problems

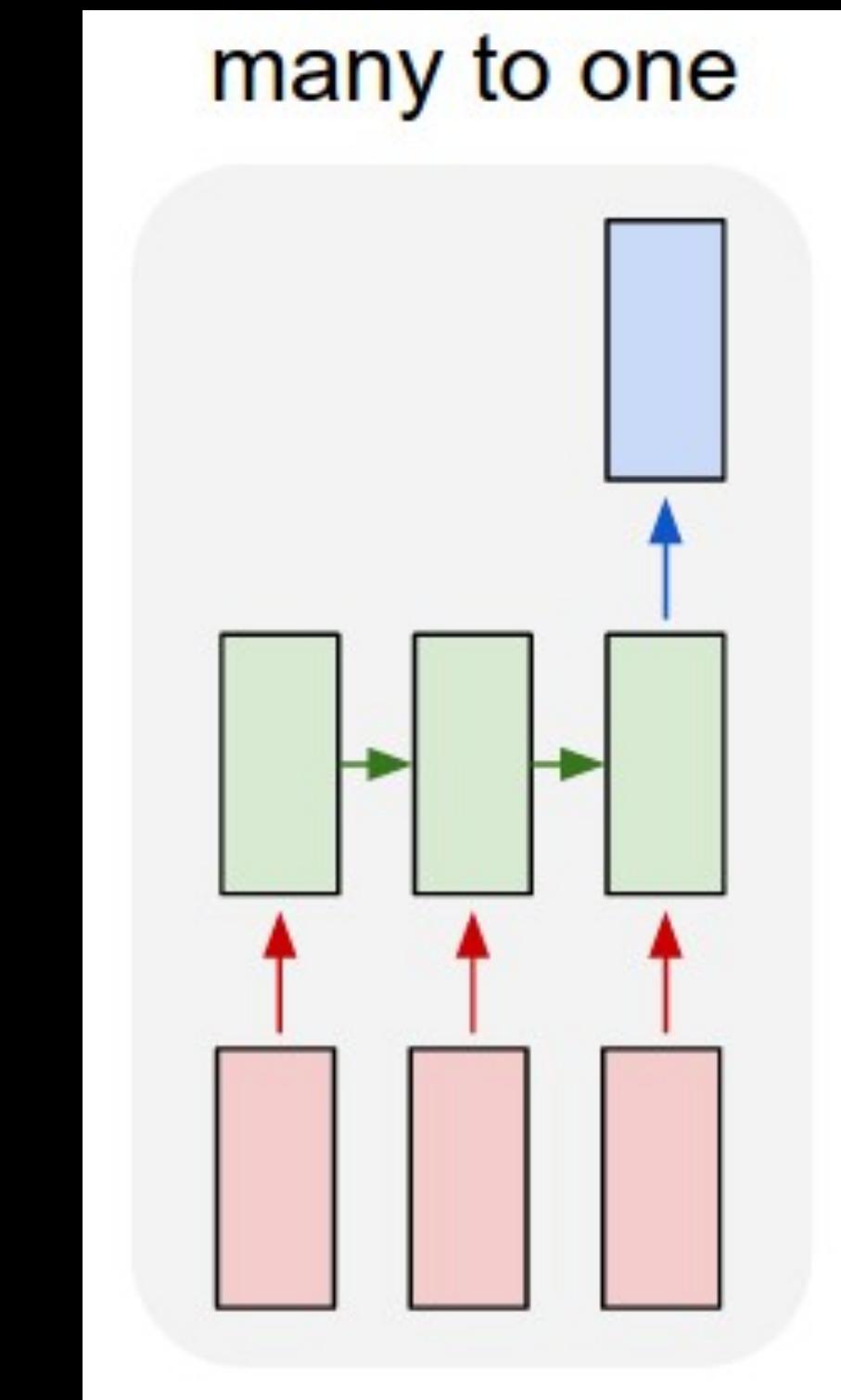
(also called seq2seq)



Source: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# BitPredict: seq2seq problems

Output [123.033] Horizon = 1 *One day forecast for Bitcoin price*

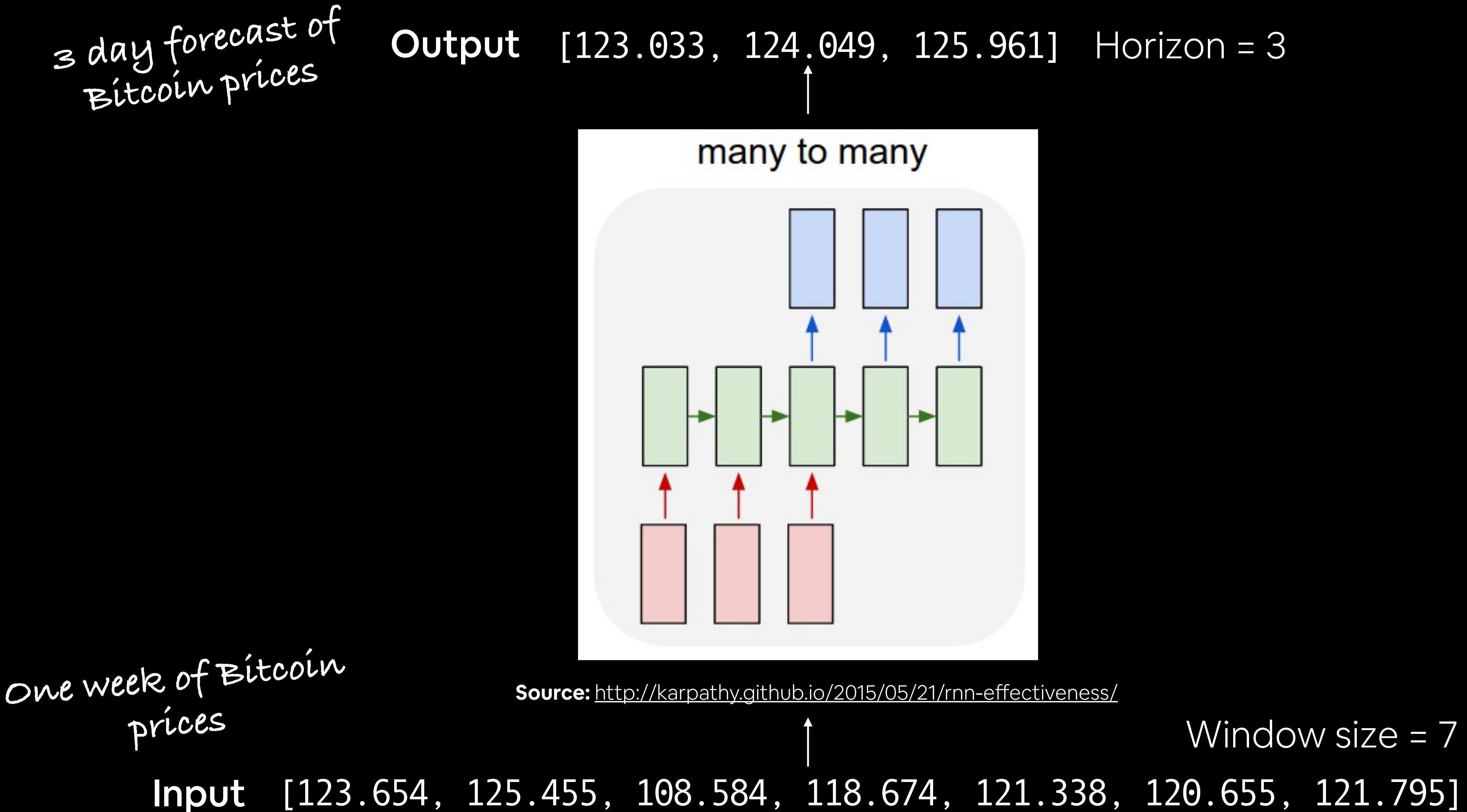


*One week of Bitcoin prices*

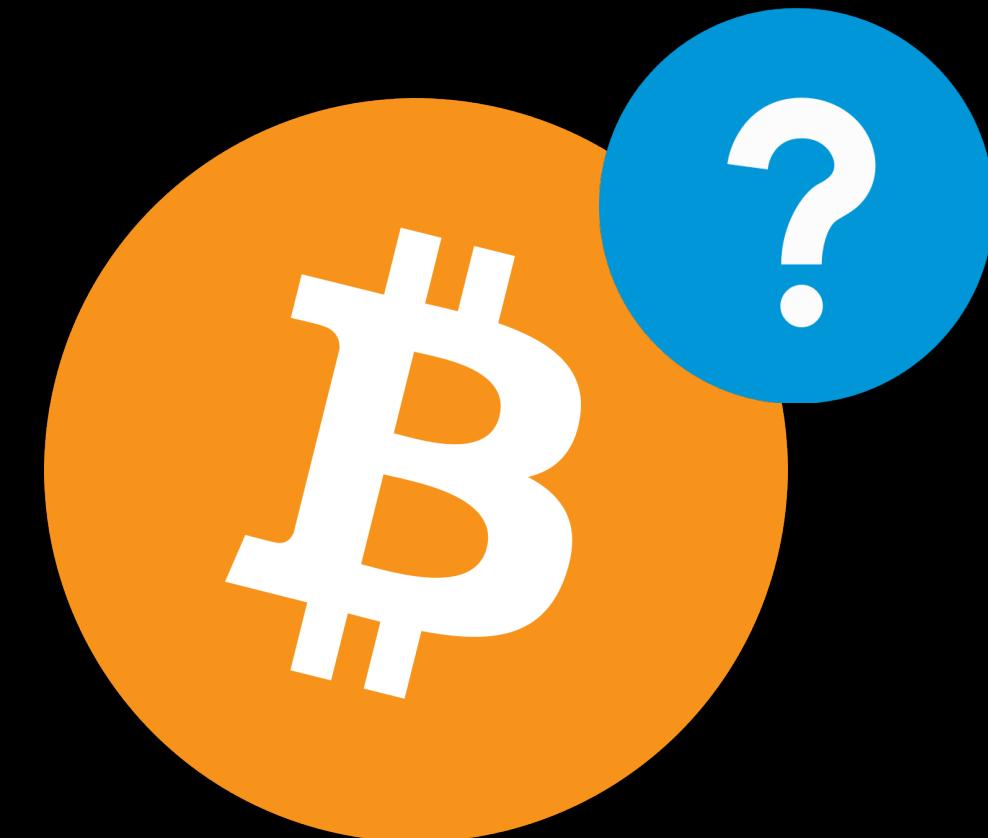
Input [123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795] Window size = 7

Source: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# BitPredict: seq2seq problems



Let's try and predict the price  
of Bitcoin!



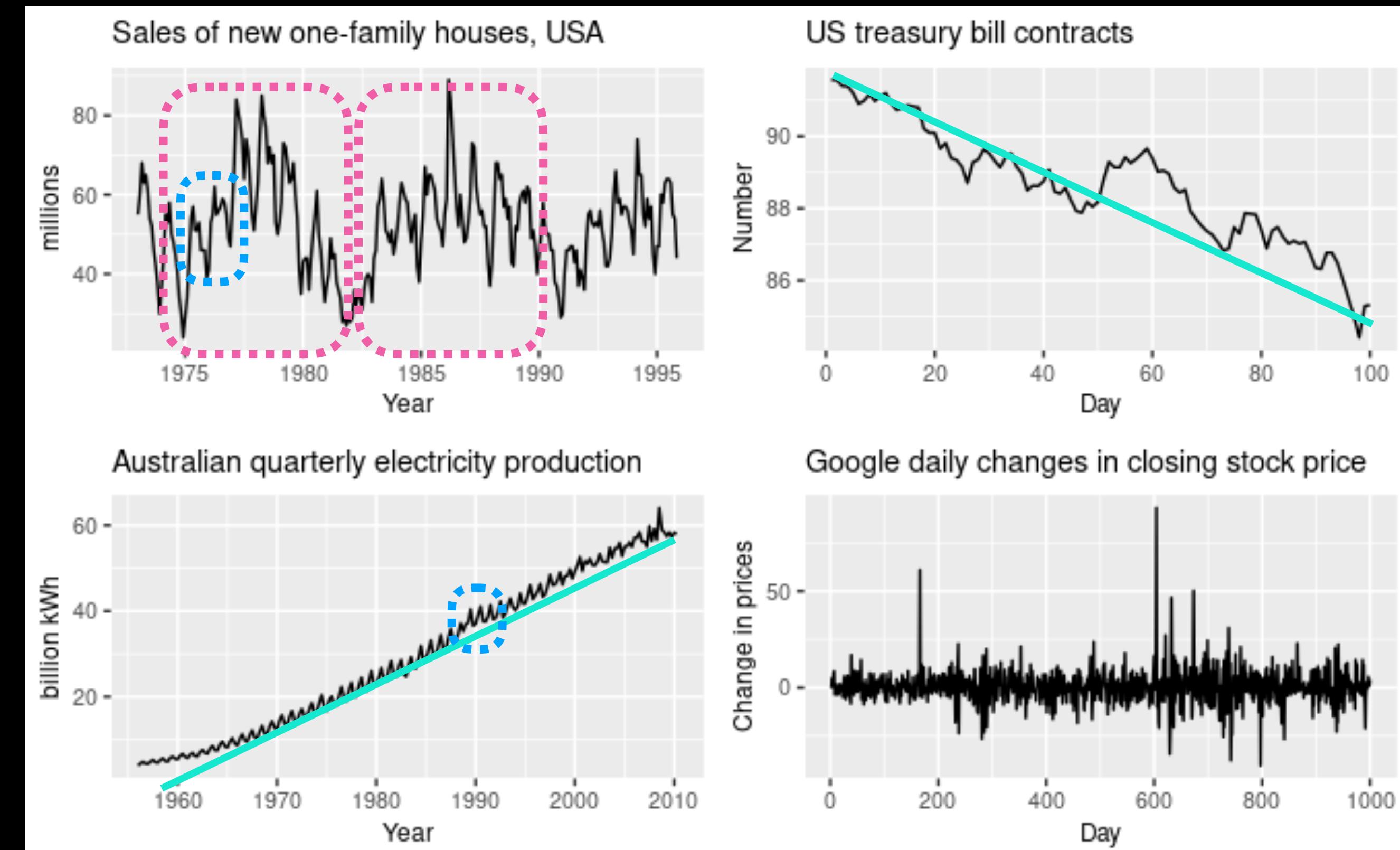
# Types of time series

(various patterns)

**Trend** — time series has a clear long-term increase or decrease (may or may not be linear)

**Seasonal** — time series affected by seasonal factors such as time of year (e.g. increased sales towards end of year) or day of week

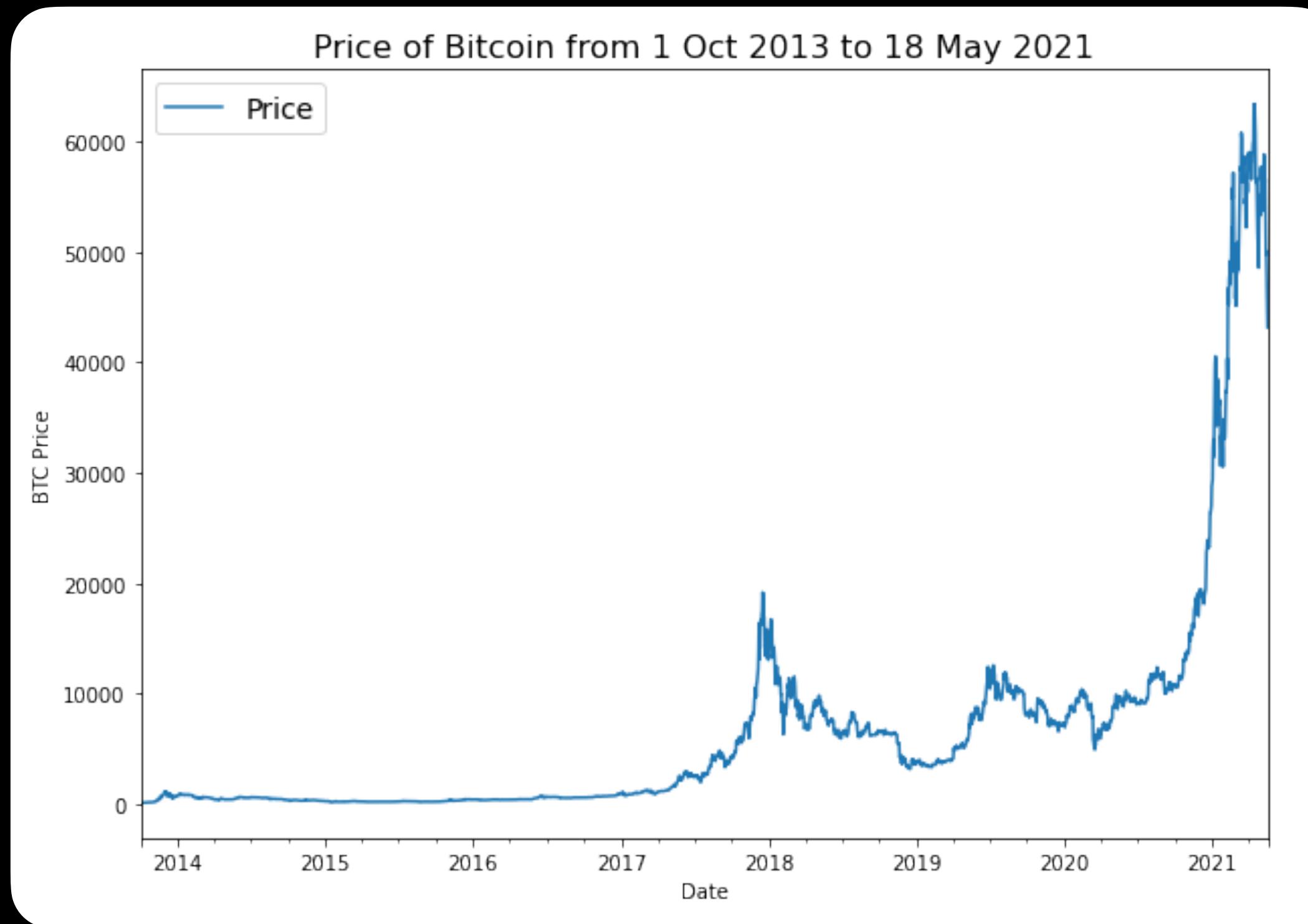
**Cyclic** — time series shows rises and falls over an unfixed period, these tend to be longer/more variable than seasonal patterns



**Source:** Figure 2.3: Four examples of time series showing different patterns from [Forecasting: Principles and Practice 3rd Edition](#).

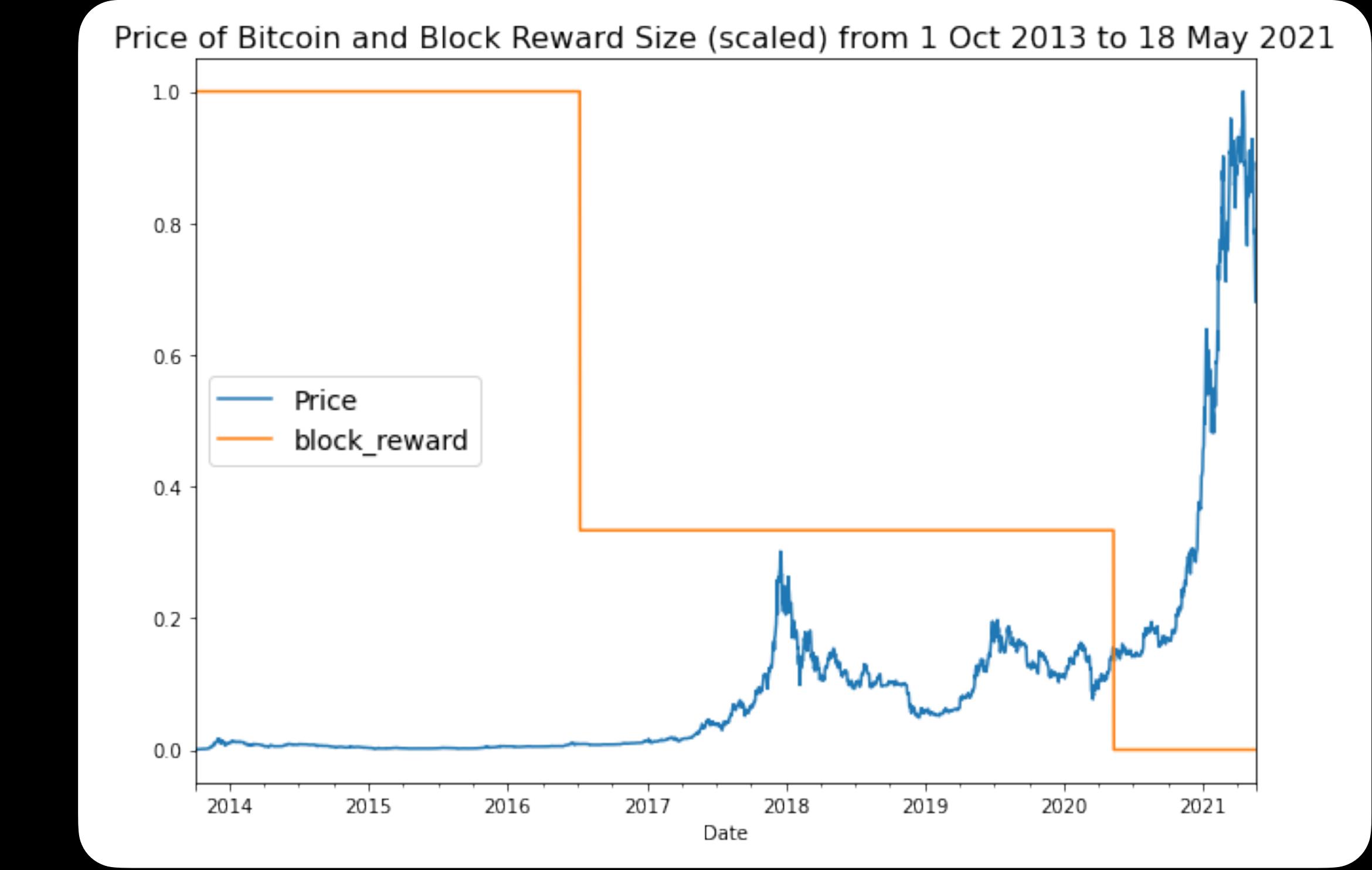
No patterns  
(random, likely not predictable)

# Univariate and Multivariate time series data



## Univariate

**Only one** variable (using the price of Bitcoin to predict the price of Bitcoin).

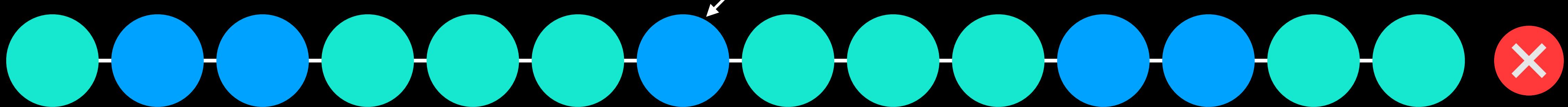


## Multivariate

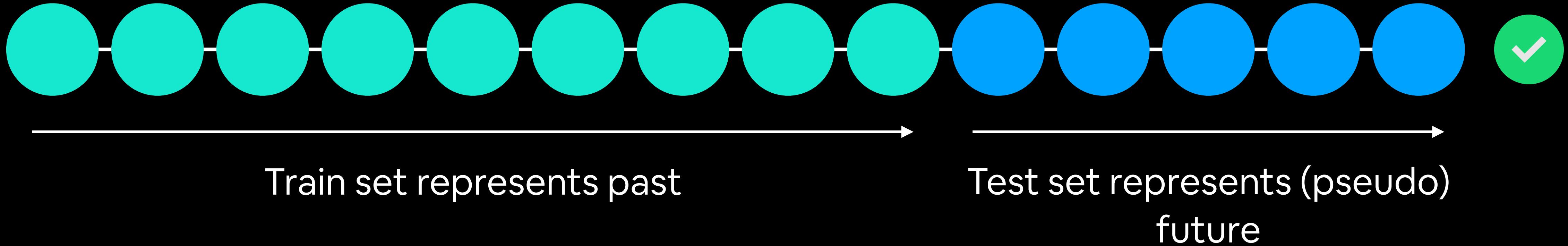
**More than one** variable (using the price of Bitcoin as well as the block reward size to predict the price of Bitcoin).

# Time series train & test sets

**Random split**



**Time series split**



(some common)

# Regression evaluation metrics

Metric Name	Metric Forumla	TensorFlow code	When to use
Mean absolute error (MAE)	$\text{MAE} = \frac{\sum_{i=1}^n  y_i - x_i }{n}$	<code>tf.keras.losses.MAE()</code> or <code>tf.metrics.mean_absolute_error()</code>	As a great starter metric for any regression problem.
Mean square error (MSE)	$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$	<code>tf.keras.losses.MSE()</code> or <code>tf.metrics.mean_squared_error()</code>	When larger errors are more significant than smaller errors.
Huber	$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for }  y - f(x)  \leq \delta, \\ \delta  y - f(x)  - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$	<code>tf.keras.losses.Huber()</code>	Combination of MSE and MAE. Less sensitive to outliers than MSE.

(some common)

# Time series forecasting evaluation metrics

Metric Name	Metric Forumla	TensorFlow code	When to use
Mean absolute error (MAE)	$\text{MAE} = \frac{\sum_{i=1}^n  y_i - x_i }{n}$	<code>tf.keras.losses.MAE()</code> or <code>tf.metrics.mean_absolute_error()</code>	As a great starter metric for any regression problem.
Mean square error (MSE)	$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$	<code>tf.keras.losses.MSE()</code> or <code>tf.metrics.mean_squared_error()</code>	When larger errors are more significant than smaller errors.
Root mean squared error (RMSE)	$\text{RMSE} = \sqrt{\text{MSE}}$	<code>tf.sqrt(tf.metrics.mean_squared_error())</code>	Similar to MSE but as interpretable as MAE (RMSE is in same units as target).
(Symmetric) Mean absolute percentage error (MAPE/ sMAPE)	$\text{MAPE} = \text{mean}( 100e_t/y_t )$	<code>tf.keras.metrics.mean_absolute_percentage_error()</code>	Recommend not to use by <u>Hyndman &amp; Koehler</u> .
Mean absolute scaled error (MASE)	$q_j = \frac{e_j}{\frac{1}{T-1} \sum_{t=2}^T  y_t - y_{t-1} }.$	Custom See <code>sktime's mase_loss()</code>	A scaled error is <b>&gt;1 if the forecast is worse than the naive</b> and <b>&lt;1 if the forecast is better than the naive</b> .

# Windows and horizons



# Windows and horizons



Turning a time series into a supervised learning problem



```
# Window for one week with the target of predicting the next day (Bitcoin prices)
[123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795] -> [123.033]
[125.455, 108.584, 118.674, 121.338, 120.655, 121.795, 123.033] -> [124.049]
[[108.584, 118.674, 121.338, 120.655, 121.795, 123.033, 124.049]] -> [125.961]
```

Window size = 7  
Data

Horizon = 1  
Label

# Windows and horizons

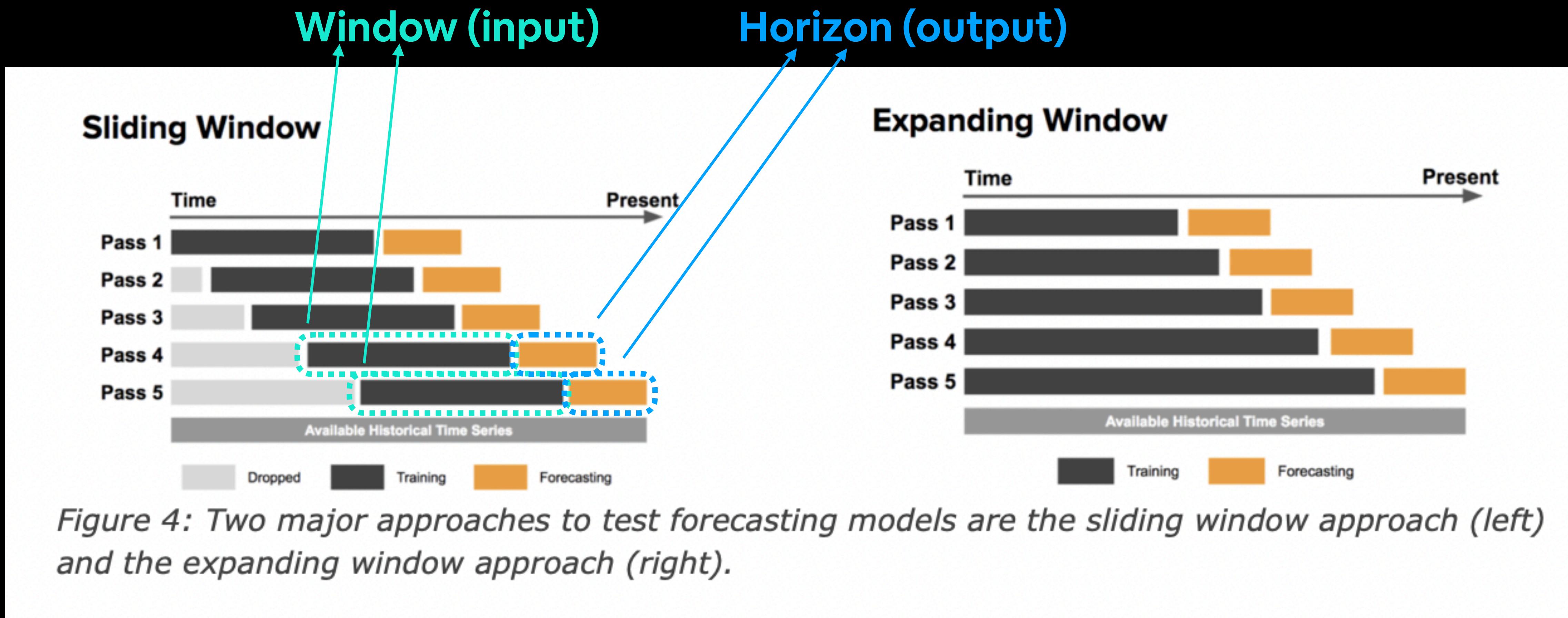


Figure 4: Two major approaches to test forecasting models are the sliding window approach (left) and the expanding window approach (right).

Source: [Forecasting at Uber blog post](#)

# Windows and horizons



```
# Window for one week with the target of predicting the next day (Bitcoin prices)
[123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795] -> [123.033]
[125.455, 108.584, 118.674, 121.338, 120.655, 121.795, 123.033] -> [124.049]
[108.584, 118.674, 121.338, 120.655, 121.795, 123.033, 124.049] -> [125.961]
```

Window size = 7

(These values are tuneable  
hyperparameters)

“Can we predict the **next** \_\_\_\_\_  
given the **past** \_\_\_\_\_?”

Horizon = 1

**Window size (input)** —  
number of time steps of  
historical data used to  
predict horizon

Data

**Horizon (output)** —  
number of time steps to  
predict into the future

Label

# BitPredict: inputs and outputs



```
# Window for one week with the target of predicting the next day (Bitcoin prices)
[123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795] -> [123.033]
[125.455, 108.584, 118.674, 121.338, 120.655, 121.795, 123.033] -> [124.049]
[108.584, 118.674, 121.338, 120.655, 121.795, 123.033, 124.049] -> [125.961]
```

Window size = 7

(These values are tuneable  
hyperparameters)

“Can we predict the **price** of  
**Bitcoin tomorrow** given the past  
week of prices?”

Horizon = 1

**Window size (input)** —  
number of time steps of  
historical data used to  
predict horizon

Data

**Horizon (output)** —  
number of time steps to  
predict into the future

Label

# BitPredict: inputs and outputs



```
# Window for one month with the target of predicting the next day (Bitcoin prices)
[123.654, 125.455, 108.584, 118.674, 121.338, 120.655,
121.795, 123.033, 124.049, 125.961, 125.279, 125.927,
126.383, 135.241, 133.203, 142.763, 137.923, 142.951,
152.551, 160.338, 164.314, 177.633, 188.297, 200.701,
180.355, 175.031, 177.696, 187.159, 192.756, 197.400] -> [196.024]
```

```
[125.455, 108.584, 118.674, 121.338, 120.655, 121.795,
123.033, 124.049, 125.961, 125.279, 125.927, 126.383,
135.241, 133.203, 142.763, 137.923, 142.951, 152.551,
160.338, 164.314, 177.633, 188.297, 200.701, 180.355,
175.031, 177.696, 187.159, 192.756, 197.400, 196.024] -> [198.048]
```

Window size = 30

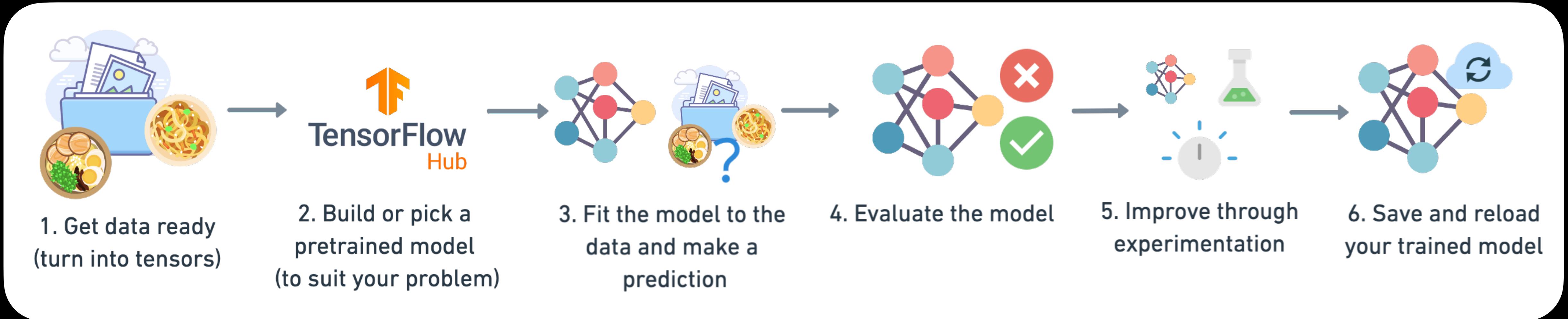
Data

Horizon = 1

Label

(These values are tuneable  
hyperparameters)

# Steps in modelling with TensorFlow



1. Turn all data into numbers (neural networks can't handle text/natural language)
2. Make sure all of your tensors are the right shape (pad sequences which don't fit)

# Experiments we're running

Experiment	Model
0	Naive model (baseline)
1	Dense model (window size = 7, horizon = 1)
2	Same architecture as model 1 (window size = 30, horizon = 1)
3	Same architecture as model 1 (window size = 30, horizon = 7)
4	Conv1D
5	LSTM
6	Same architecture as model 1 (but with multivariate data)
7	<u>N-BEATS algorithm</u>
8	Ensemble (multiple models stacked together)
9	Future prediction model
10	Same architecture as model 1 (but with turkey  data introduced)

```
59 Machine Learning Engineer*
60 -----
61
62 1. Download a paper
63 2. Implement it
64 3. Keep doing this until you have skills
```

- George Hotz

\*Machine Learning Engineering also involves building infrastructure around your models/  
data preprocessing steps

# What we're doing

Replicating this paper

The screenshot shows a web browser displaying an arXiv.org page. The title of the paper is "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting" by Boris N. Oreshkin, Dmitri Carpol, Nicolas Chapados, and Yoshua Bengio. The abstract discusses solving univariate time series point forecasting using deep learning, specifically a deep neural architecture with backward and forward residual links and a stack of fully-connected layers. It highlights state-of-the-art performance on M3, M4, and TOURISM datasets, and demonstrates its interpretability and performance on heterogeneous datasets. The page includes standard arXiv navigation and search features, as well as sections for download (PDF, other formats), references, and citations.

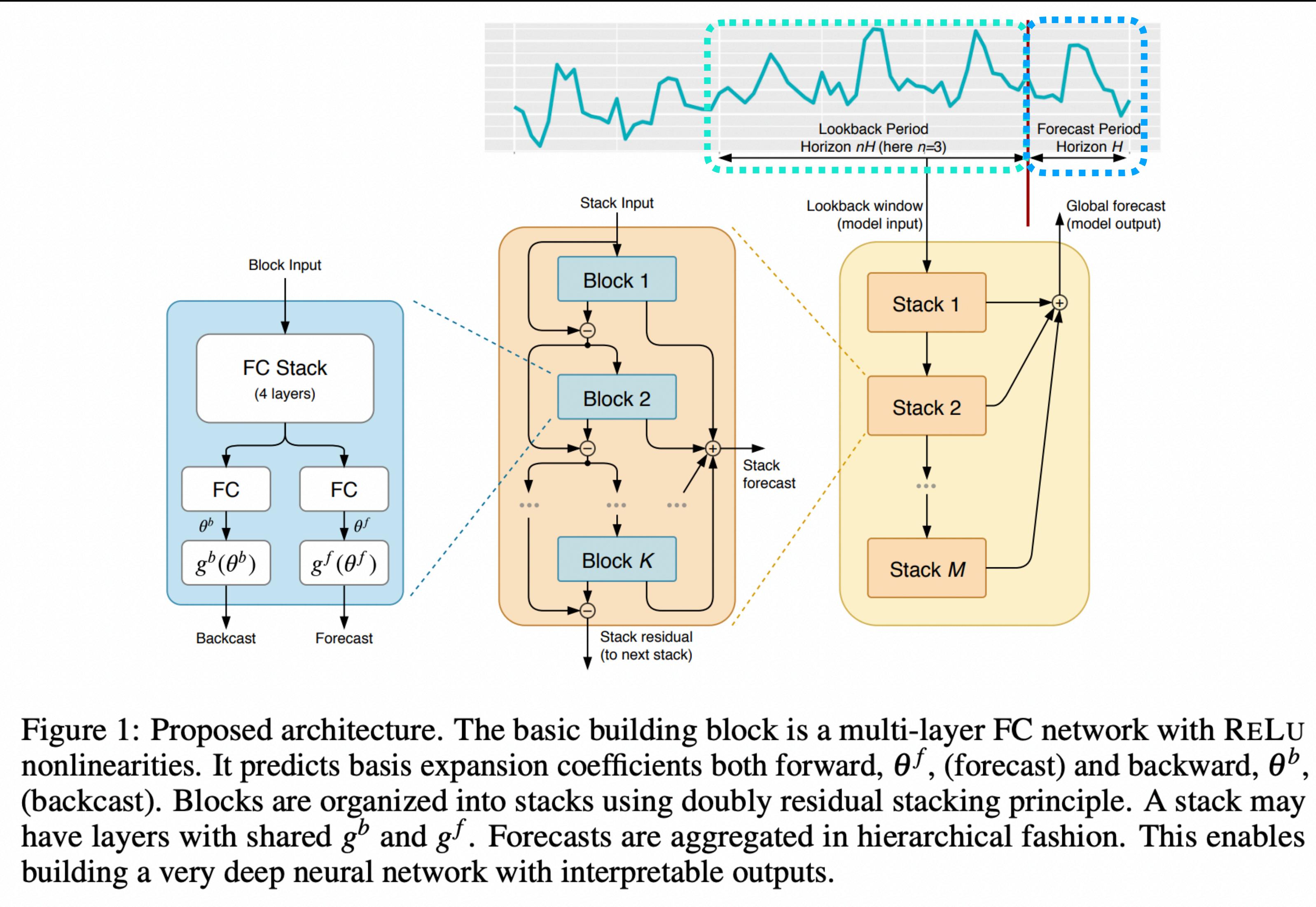
We focus on solving the univariate times series point forecasting problem using deep learning. We propose a deep neural architecture based on backward and forward residual links and a very deep stack of fully-connected layers. The architecture has a number of desirable properties, being interpretable, applicable without modification to a wide array of target domains, and fast to train. We test the proposed architecture on several well-known datasets, including M3, M4 and TOURISM competition datasets containing time series from diverse domains. We demonstrate state-of-the-art performance for two configurations of N-BEATS for all the datasets, improving forecast accuracy by 11% over a statistical benchmark and by 3% over last year's winner of the M4 competition, a domain-adjusted hand-crafted hybrid between neural network and statistical time series models. The first configuration of our model does not employ any time-series-specific components and its performance on heterogeneous datasets strongly suggests that, contrarily to received wisdom, deep learning primitives such as residual blocks are by themselves sufficient to solve a wide range of forecasting problems. Finally, we demonstrate how the proposed architecture can be augmented to provide outputs that are interpretable without considerable loss in accuracy.

Subjects: Machine Learning (cs.LG); Machine Learning (stat.ML)  
Cite as: arXiv:1905.10437 [cs.LG]  
(or arXiv:1905.10437v4 [cs.LG] for this version)

Source: <https://arxiv.org/abs/1905.10437>

Source: Figure 1 (proposed architecture) N-BEATS paper

Window      Horizon



```

# Create NBeatsBlock custom layer
class NBeatsBlock(tf.keras.layers.Layer):
    def __init__(self, # the constructor takes all the hyperparameters for the layer
                 input_size: int,
                 theta_size: int,
                 horizon: int,
                 n_neurons: int,
                 n_layers: int,
                 **kwargs): # takes care of all of the arguments for the parent class (input_shape, trainable, name)
        super().__init__(**kwargs)
        self.input_size = input_size
        self.theta_size = theta_size
        self.horizon = horizon
        self.n_neurons = n_neurons
        self.n_layers = n_layers

        # Block contains stack of 4 fully connected layers each has ReLU activation
        self.hidden = [tf.keras.layers.Dense(n_neurons, activation="relu") for _ in range(n_layers)]

        # Output of block is a theta layer with linear activation
        self.theta_layer = tf.keras.layers.Dense(theta_size, activation="linear", name="theta")

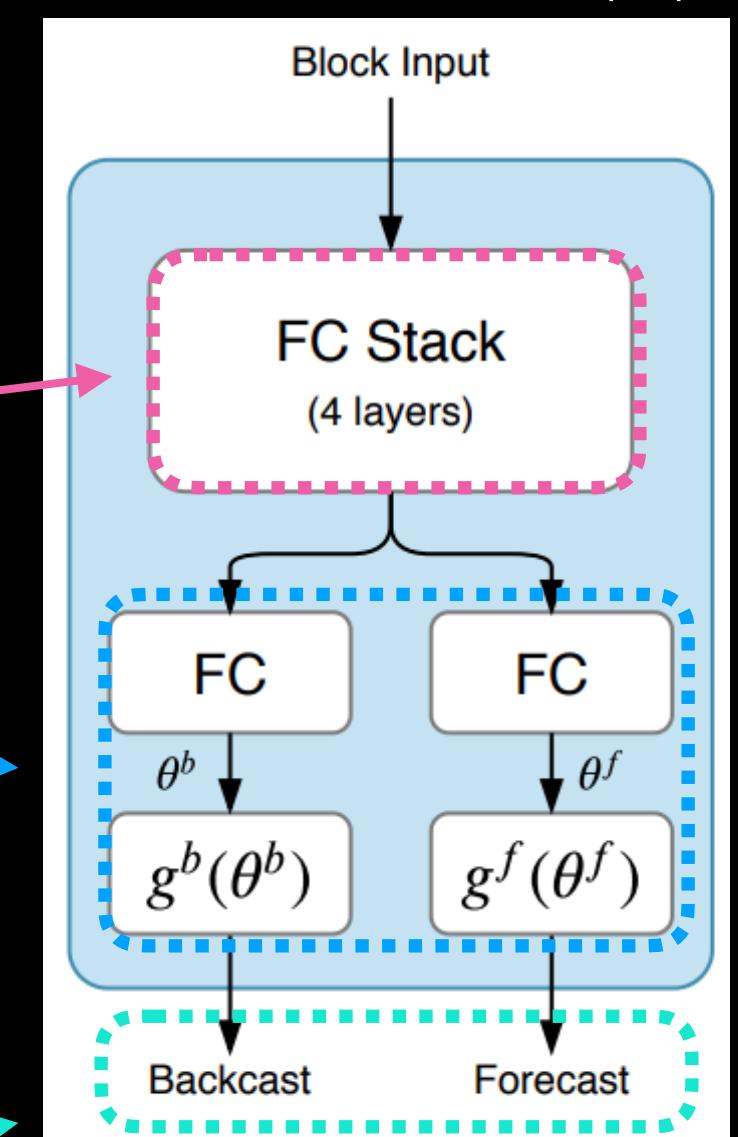
    def call(self, inputs): # the call method is what runs when the layer is called
        x = inputs
        for layer in self.hidden: # pass inputs through each hidden layer
            x = layer(x)
        theta = self.theta_layer(x)

        # Output the backcast and forecast from theta
        backcast, forecast = theta[:, :self.input_size], theta[:, -self.horizon:]

        return backcast, forecast

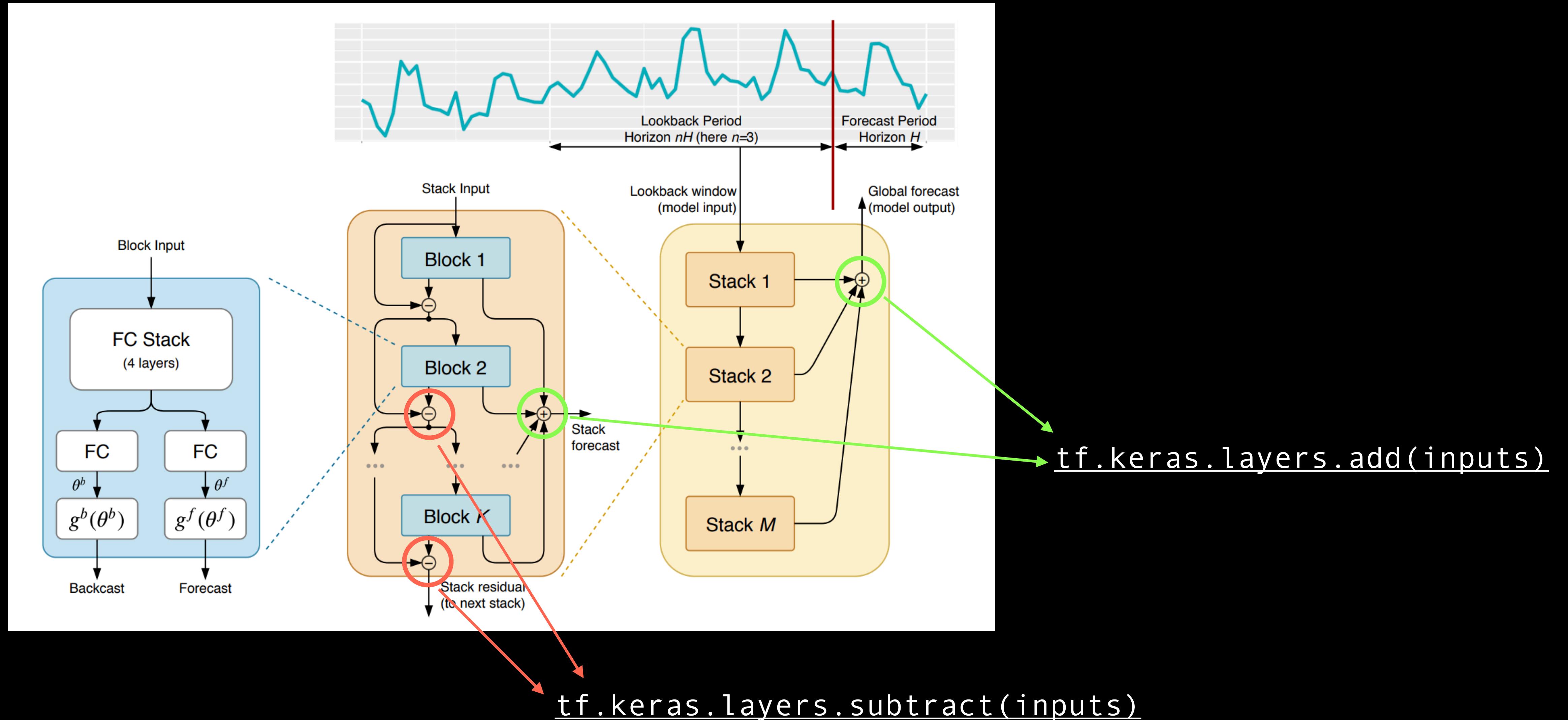
```

**Source:** [Figure 1 \(proposed architecture\) N-BEATS paper](#)

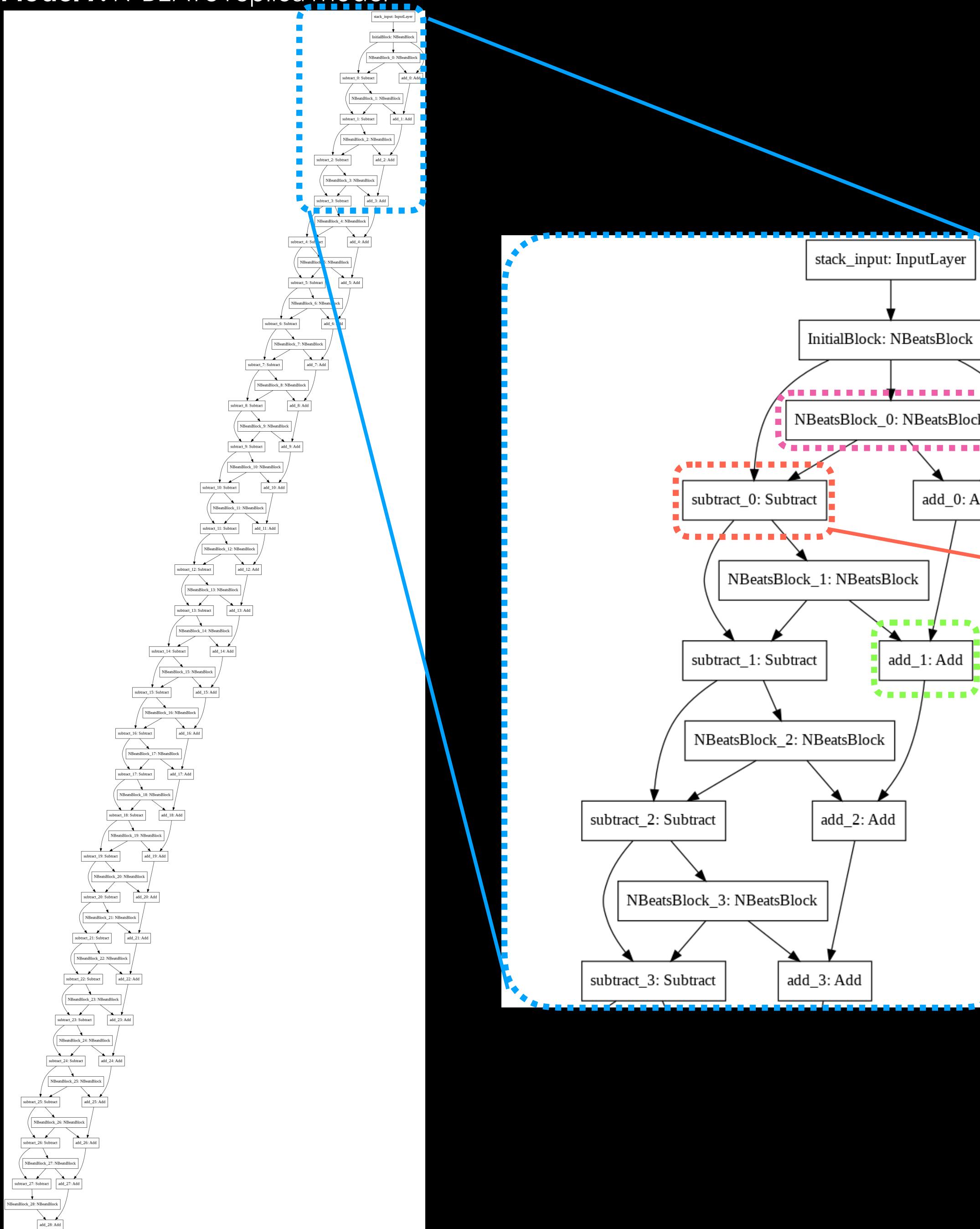


N-BEATS basic block replication with TensorFlow layer subclassing (section 3.1 of N-BEATS paper)

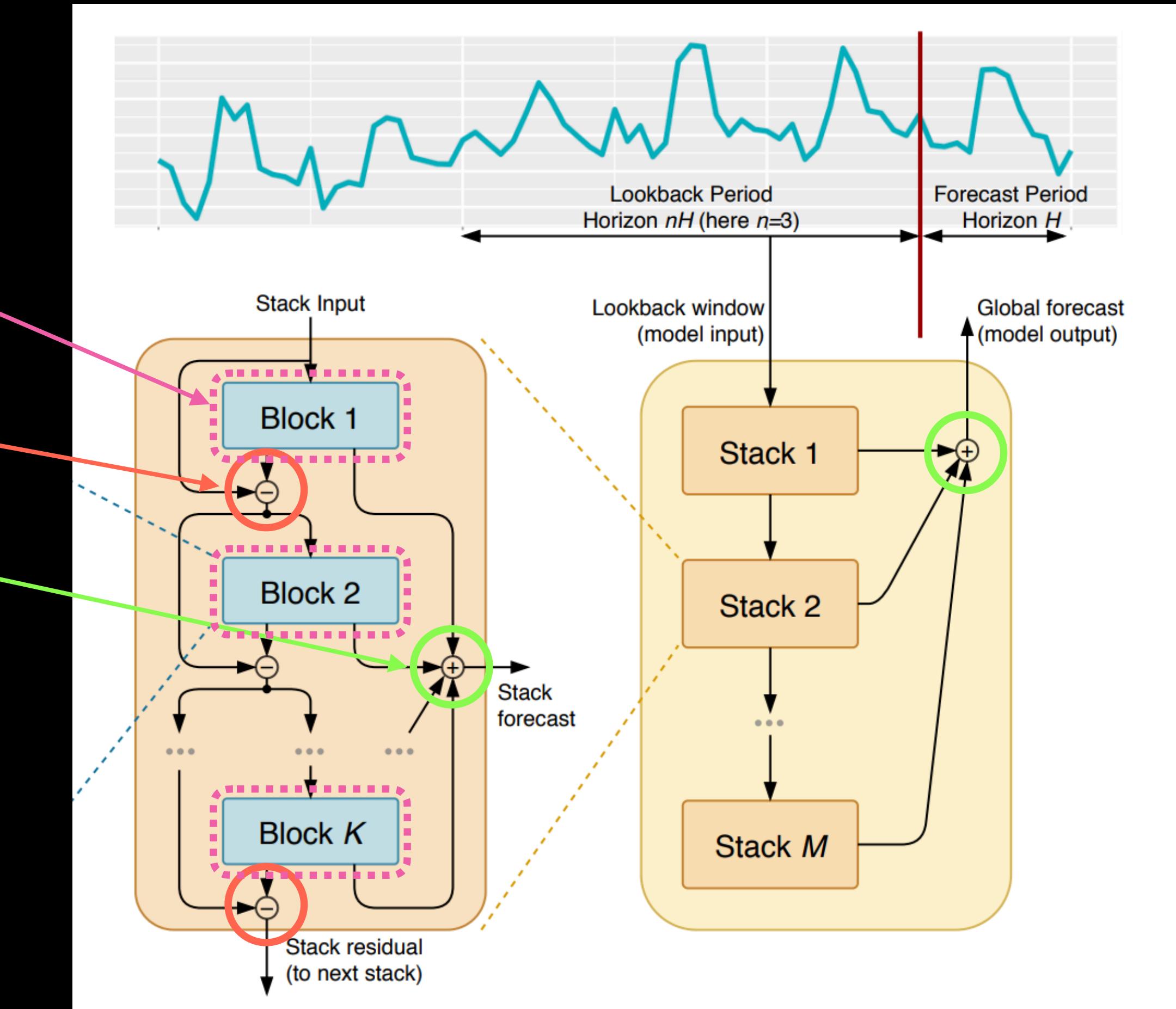
Source: Figure 1 (proposed architecture) N-BEATS paper



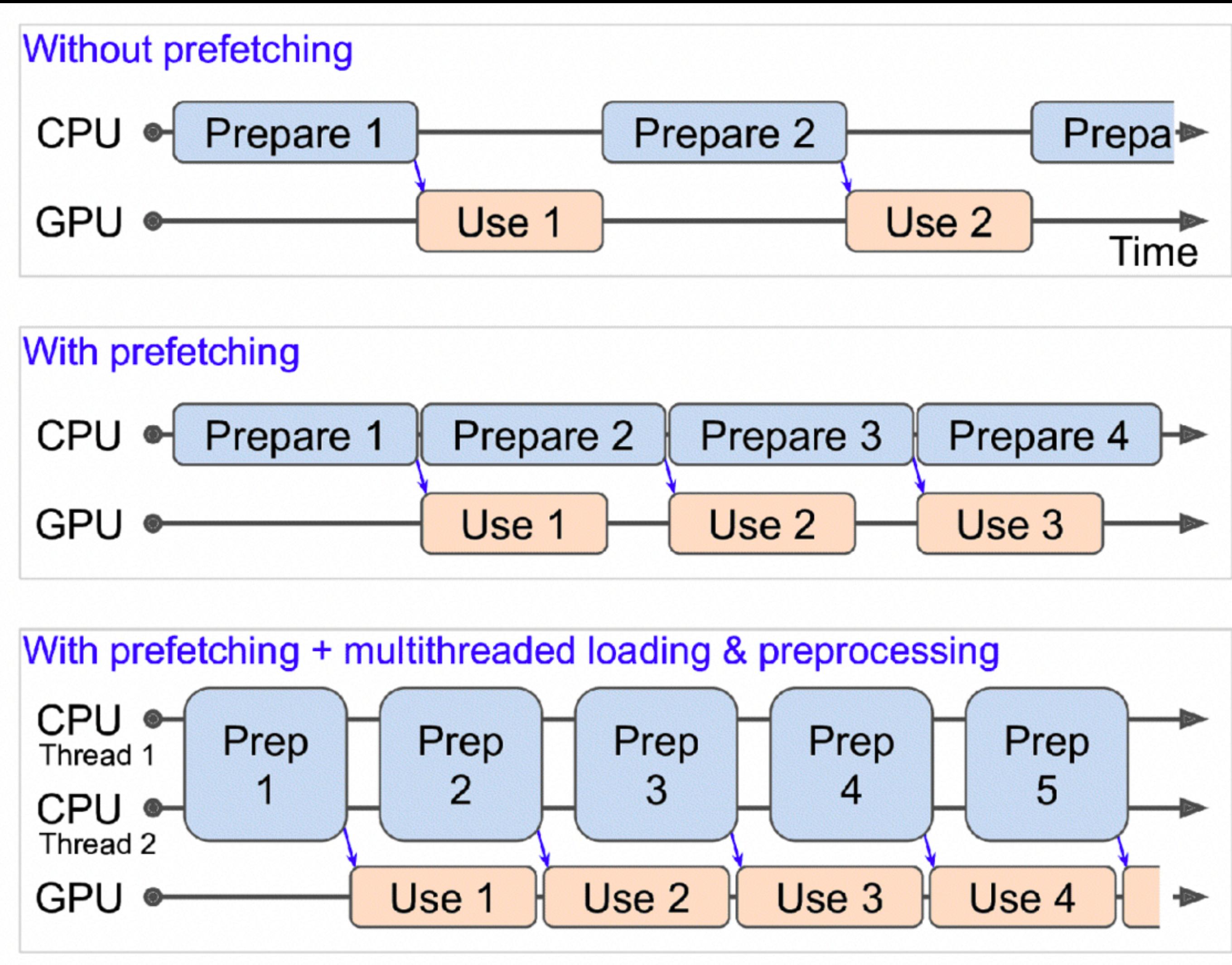
## Model 7: N-BEATS replica model



**Source:** Figure 1 (proposed architecture) N-BEATS paper



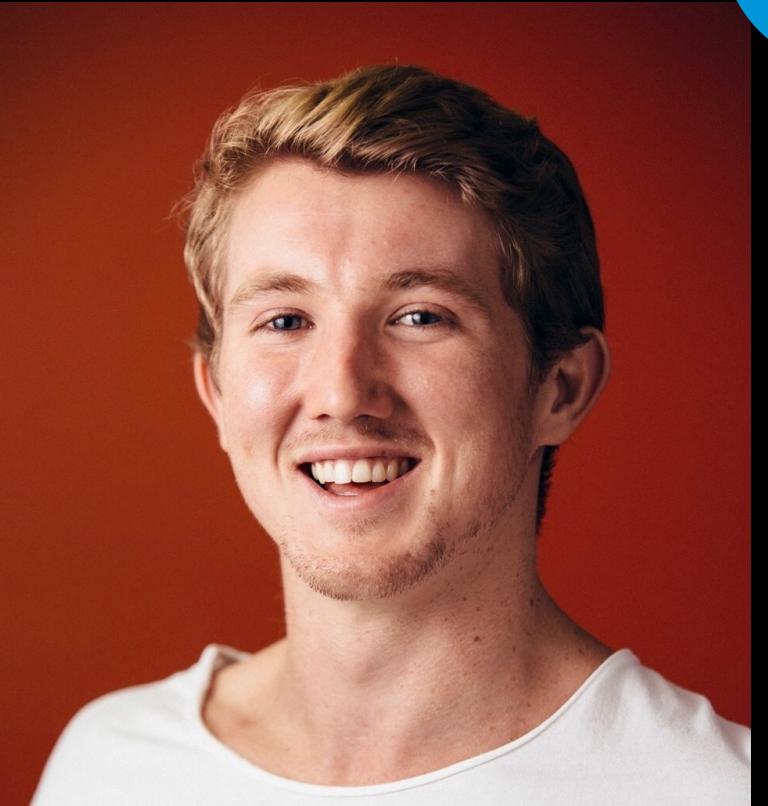
# Prefetching



Source: Page 422 of [Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow Book](#) by Aurélien Géron

# What is an ensemble model?

Single  
Model



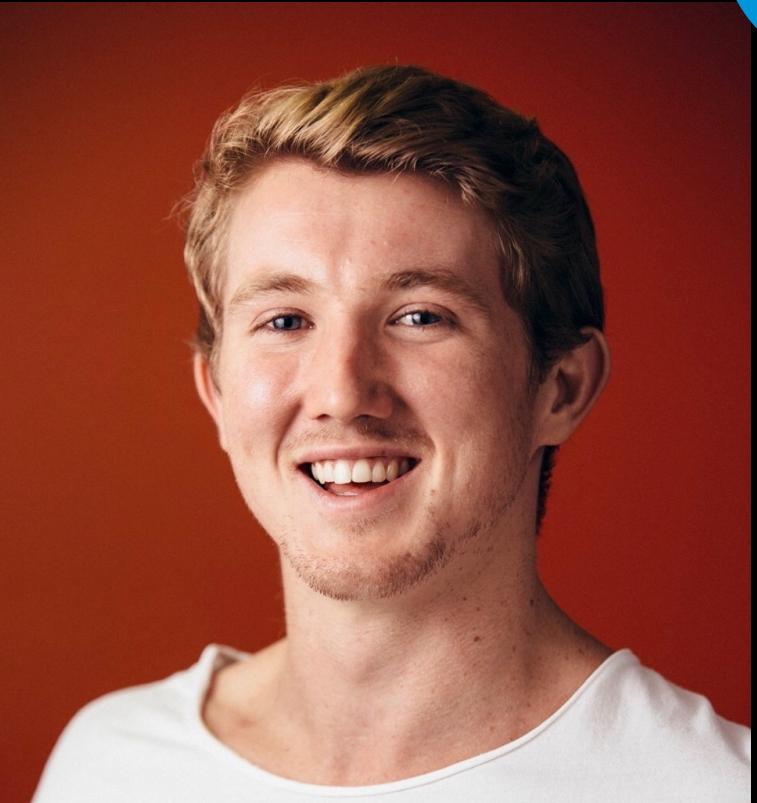
Smart level 7

Ensemble

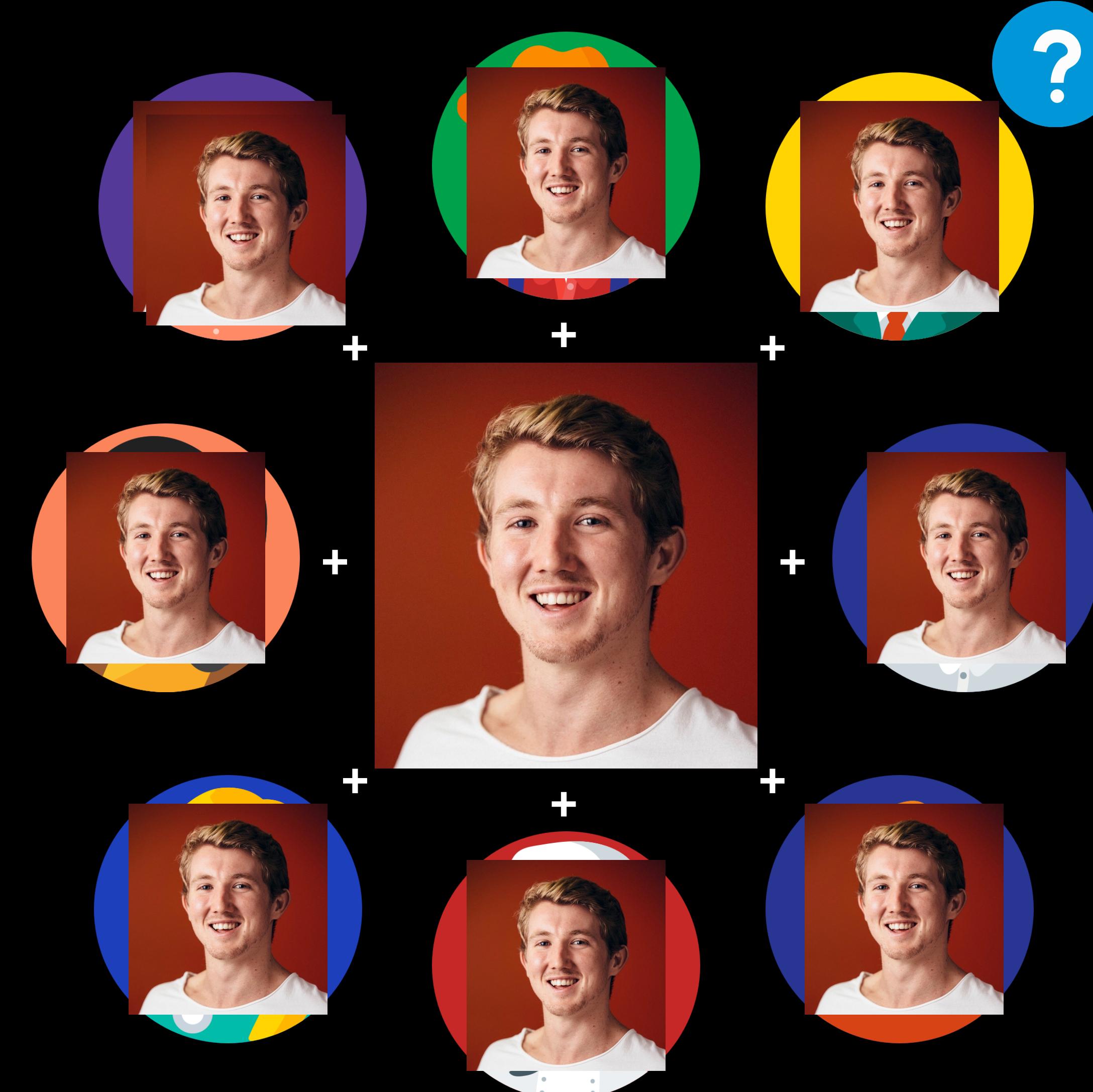


Smart level 10

# What is an ensemble model?



Smart level 7



Smart level 7

# Example Forecasting Problems at Uber



Market demand



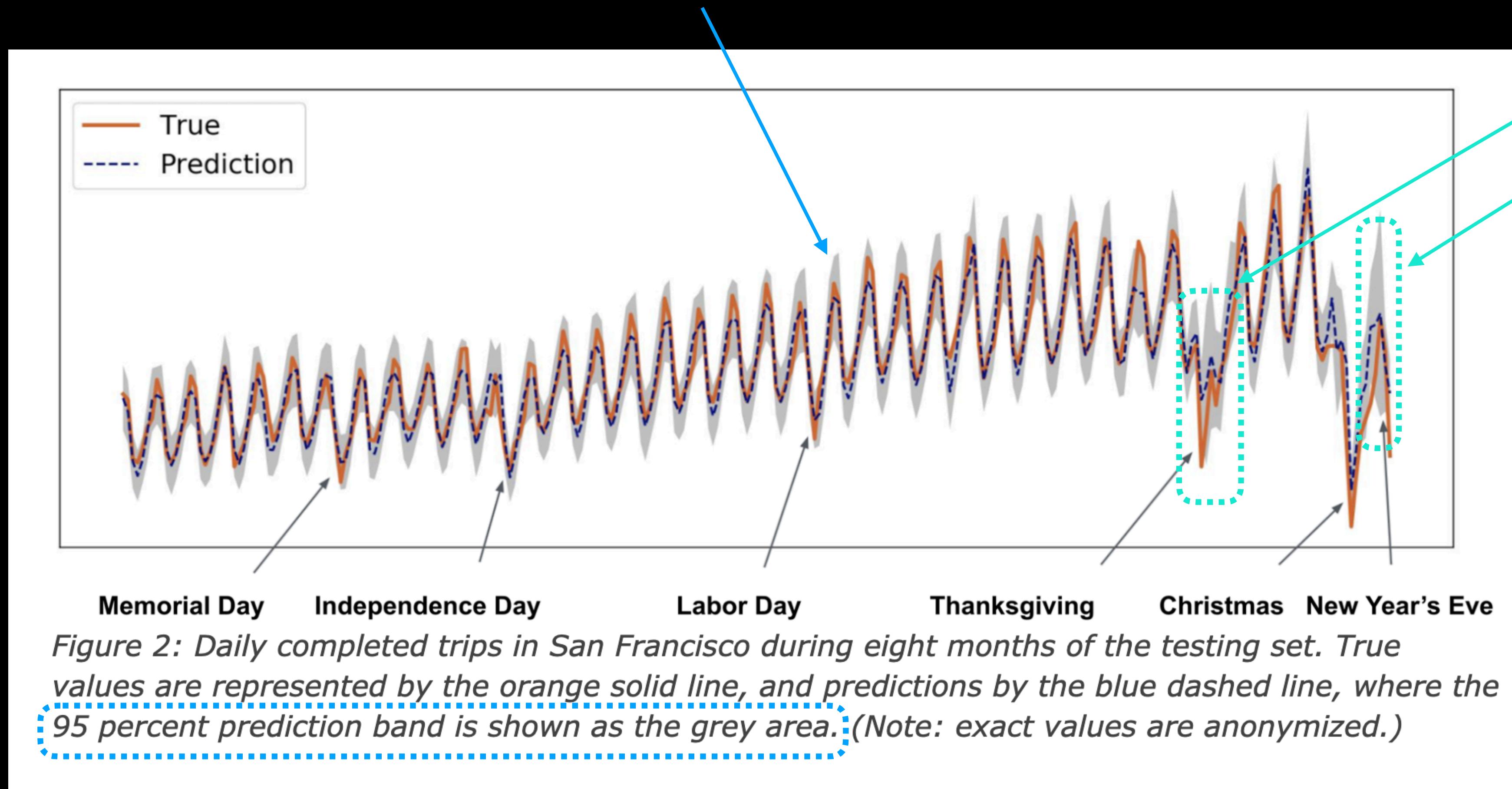
Compute requirements



Marketing campaigns

# Prediction intervals/uncertainty estimates

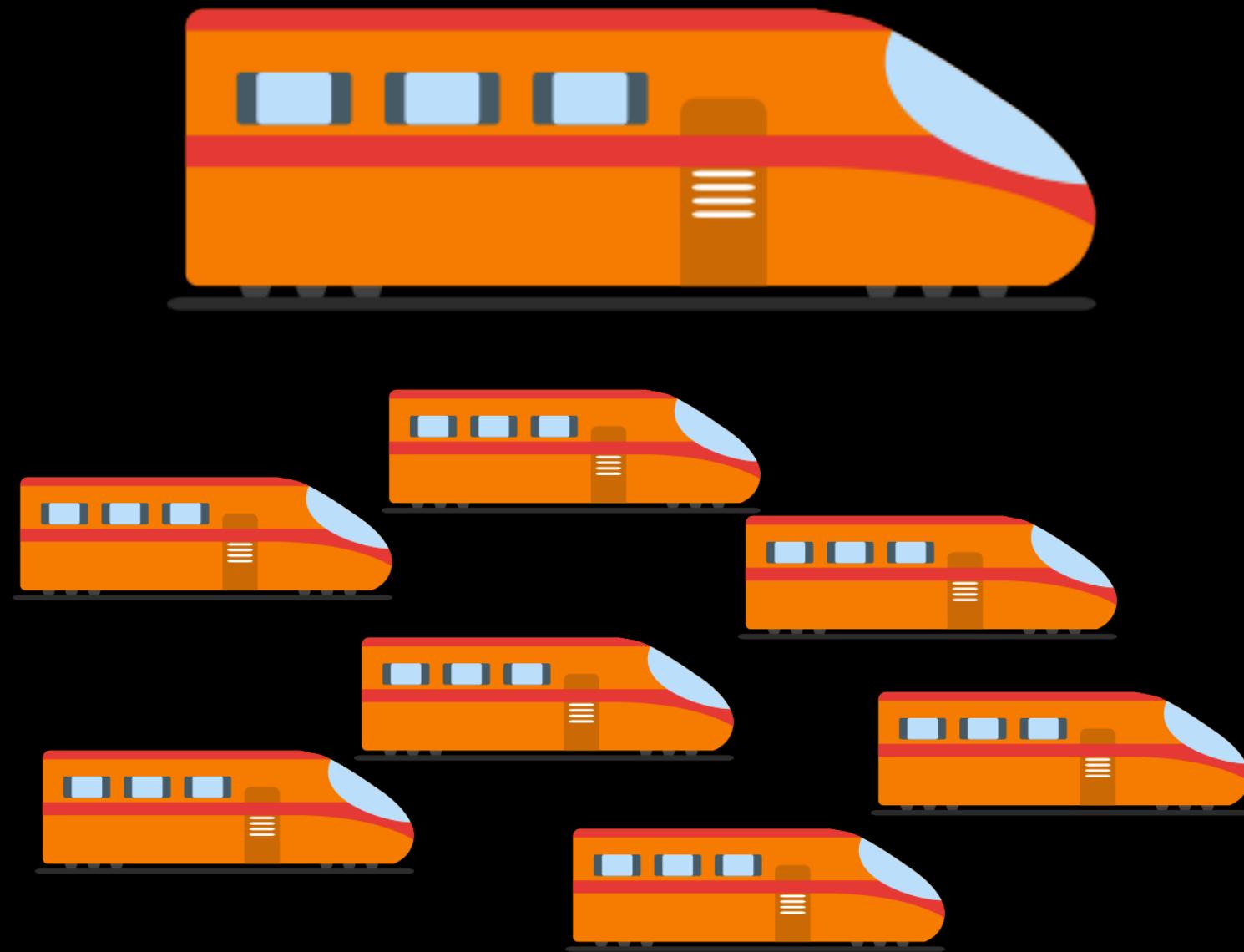
Large periods of uncertainty around big events



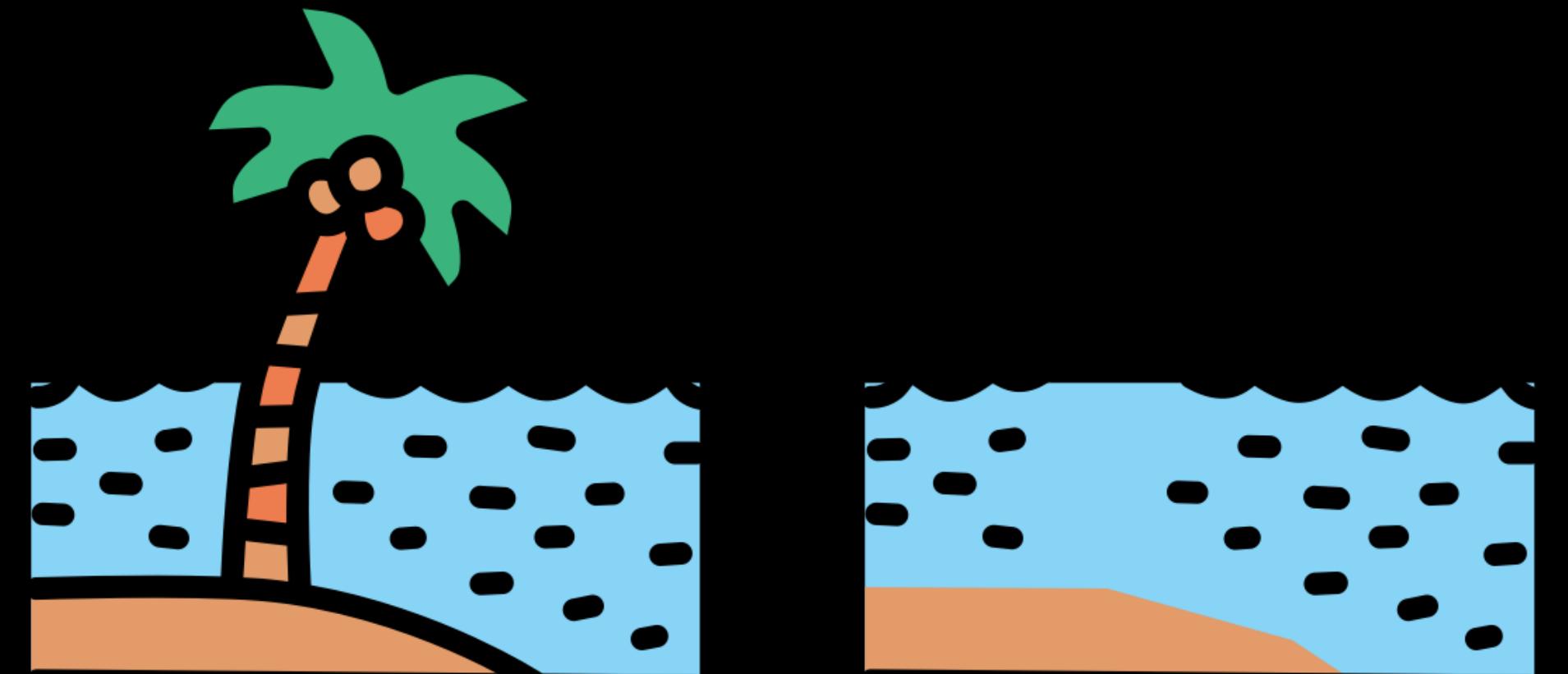
Source: Figure 2 from [Engineering Uncertainty Estimation in Neural Networks for Time Series Prediction at Uber](#)

# Types of Uncertainty

Aleatoric Uncertainty  
(data/subway uncertainty)



Epistemic Uncertainty  
(model/coconut uncertainty)

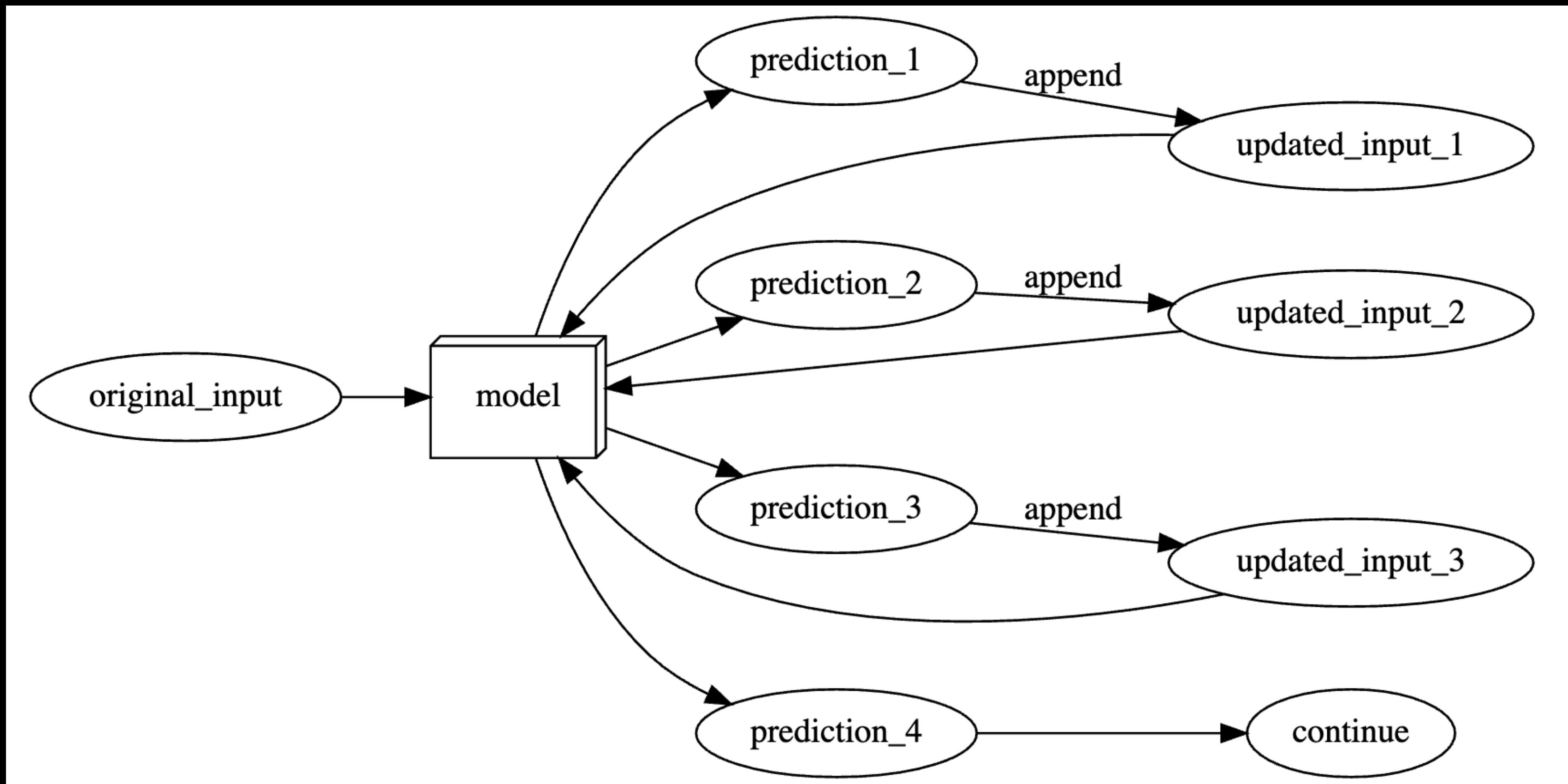


- **Cannot** be reduced with more data

(when measuring prediction intervals,  
we're estimating a form of aleatoric  
uncertainty)

- **Can** be reduced with more data

# Making time series forecasts



For time series forecasts, you have to retrain a model every time you make a prediction.

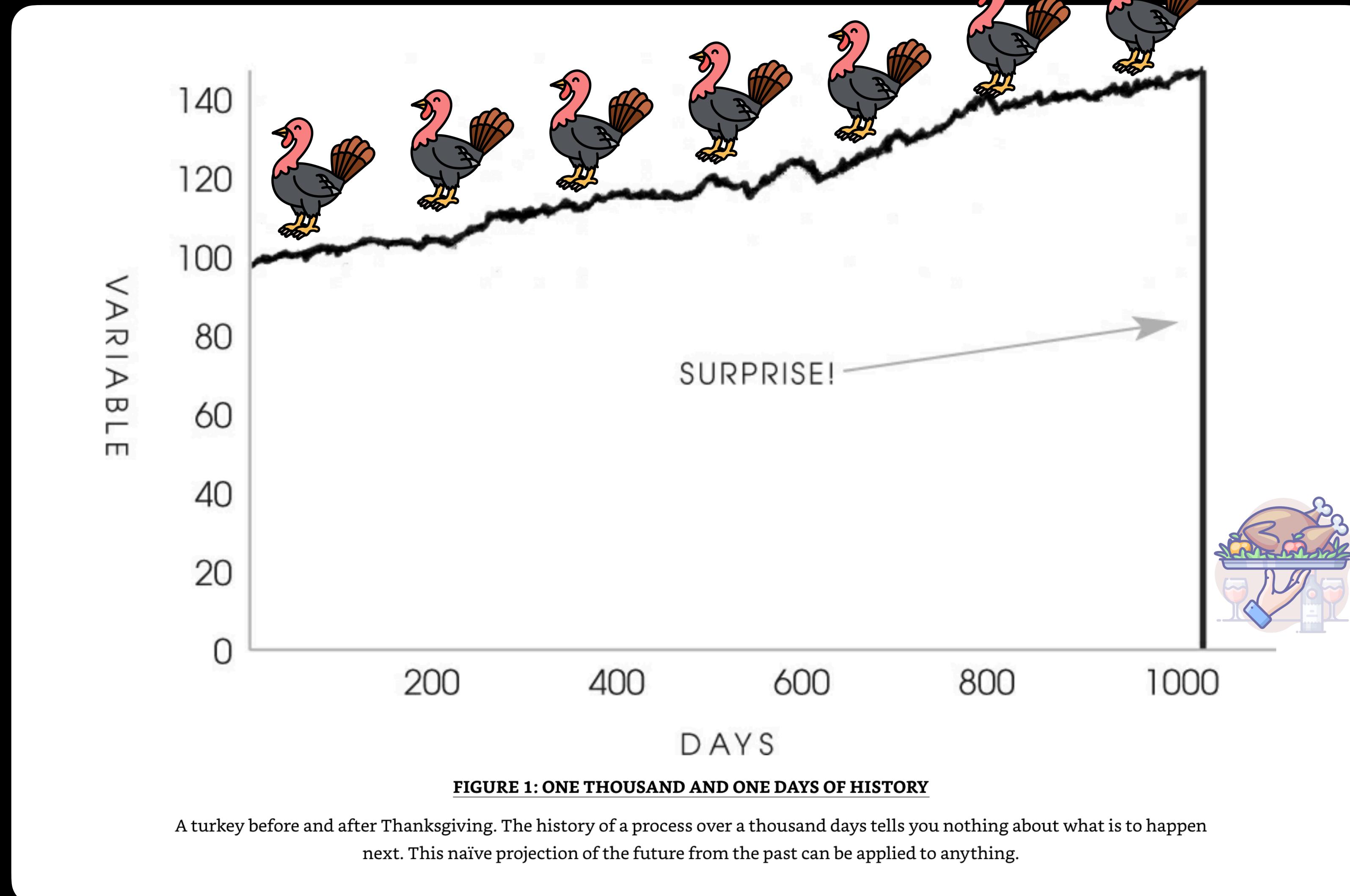
*(as new data comes in, retrain a model  
and make a prediction)*

# The turkey problem



# The turkey problem

(why forecasting is BS)



**Source:** Page 41 of [The Black Swan: The Impact of the Highly Improbable](#) by Nassim Nicholas Taleb

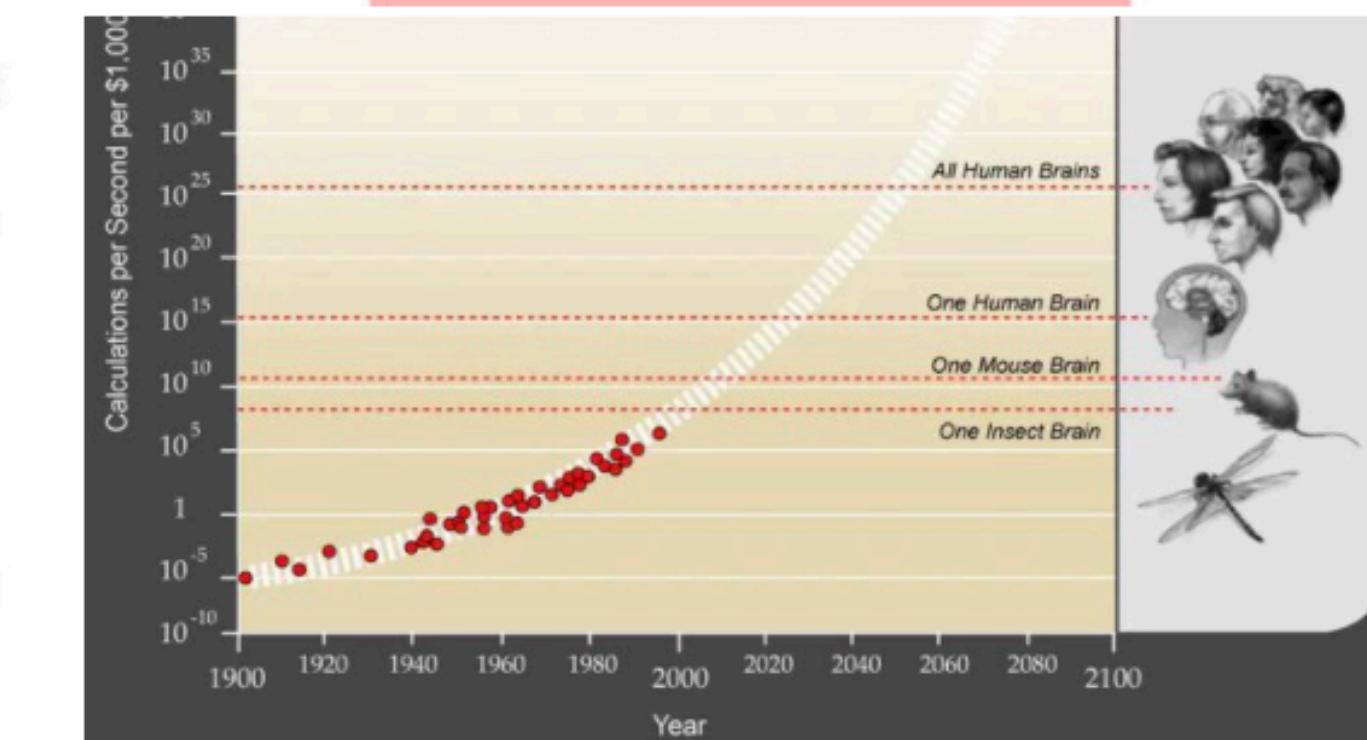
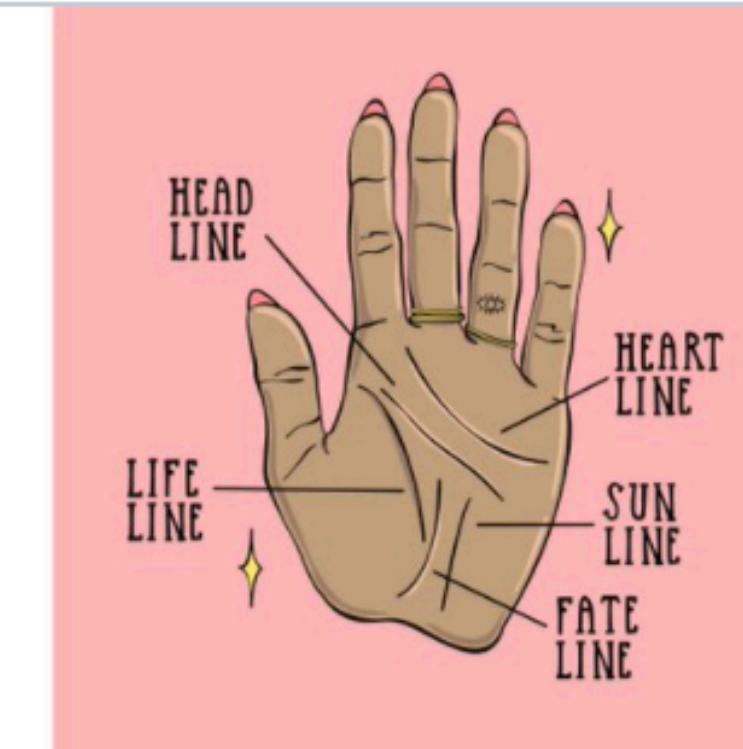
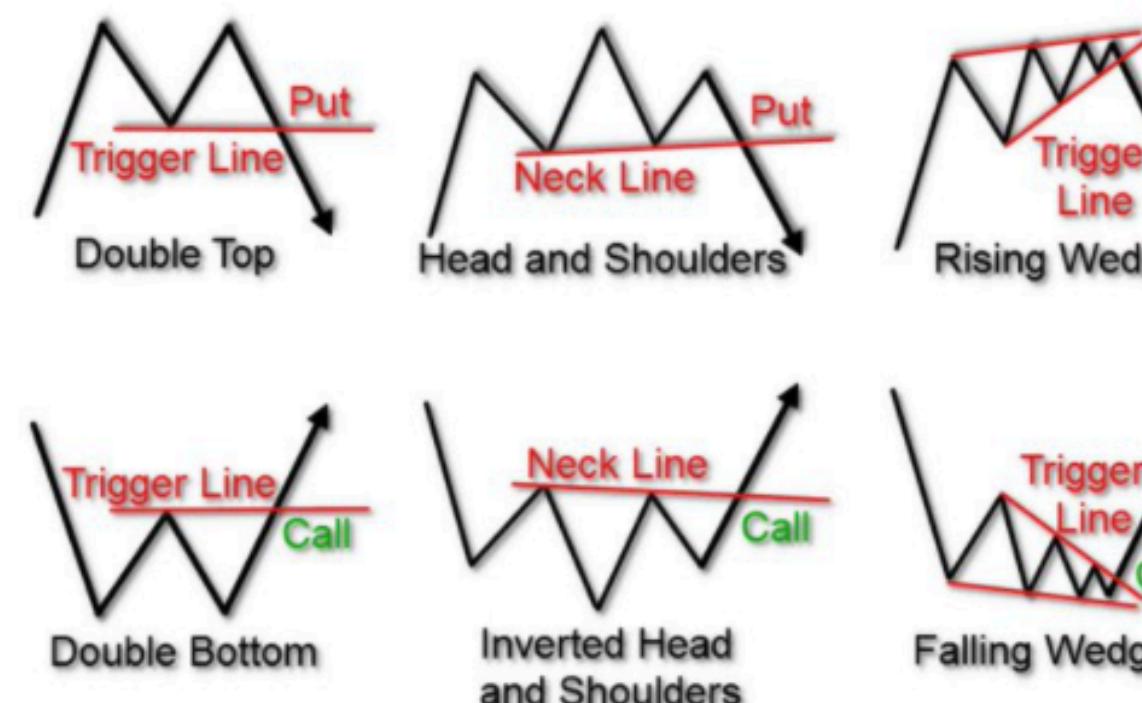


**Mark Saroufim** @marksaroufim · Mar 3  
Same vibe

...

## Time-Series Forecasting: Predicting Stock Prices Using An LSTM Model

In this post I show you how to predict stock prices using a forecasting LSTM model



**Source:** [Mark Saroufim's Twitter](#)

# Improving a model

(from a model's perspective)

```
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(4, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.001),
              metrics=[ "accuracy" ])

# 3. Fit the model
model.fit(x_train_subset, y_train_subset, epochs=5)
```

Smaller model



```
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=[ "accuracy" ])

# 3. Fit the model
model.fit(x_train_full, y_train_full, epochs=100)
```

Larger model

## Common ways to improve a deep model:

- Adding layers
- Increase the number of hidden units
- Change the activation functions
- Change the optimization function
- Change the learning rate (because you can alter each of these, they're hyperparameters)
- Fitting on more data
- Fitting for longer

# Improving a model

(from a data perspective)

## Method to improve a model (reduce overfitting)

### What does it do?

More data

Gives a model more of a chance to learn patterns between samples (e.g. if a model is performing poorly on images of pizza, show it more images of pizza).

Data augmentation (usually for images)

Increase the diversity of your training dataset without collecting more data (e.g. take your photos of pizza and randomly rotate them 30°). Increased diversity forces a model to learn more generalisation patterns.

Better data (**feature engineering**)

Not all data samples are created equally. Removing poor samples from or adding better samples/**engineering new features** in your dataset can improve your model's performance.

Use transfer learning

Take a model's pre-learned patterns from one problem and tweak them to suit your own problem. For example, take a model trained on pictures of cars to recognise pictures of trucks.

# The machine learning explorer's motto

“Visualize, visualize, visualize”



# The machine learning practitioner's motto

“Experiment, experiment, experiment”



*(try lots of things and see what  
tastes good)*