SYBIL: The General-Purpose Forecaster

Architecture Design Report (v1.0)

Authors: Kevin R.C., Francesco B., & Mahdi R.

Date: July 28, 2023

@Temporai, SingularityNET

Contact: kevin@tempor.ai

Outline

Here is the outline summary of the design report:

1) Overview

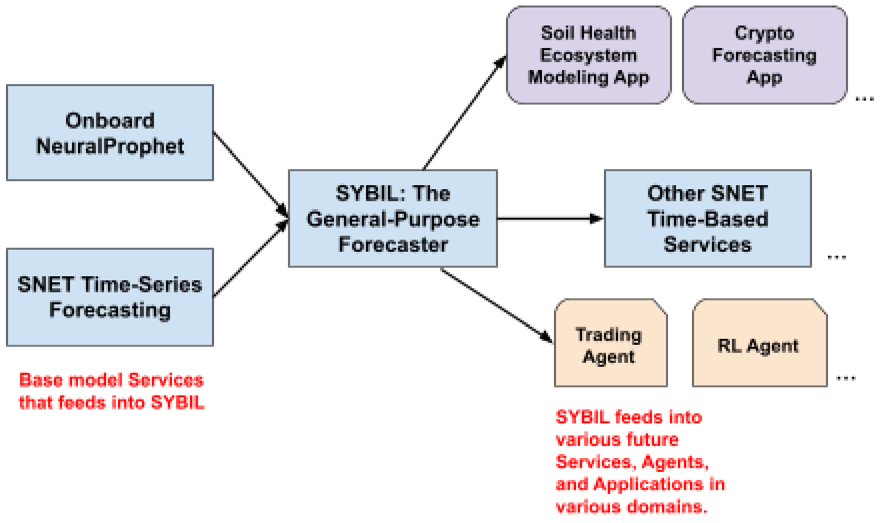
1a) What is SYBIL?

SIBYL is an AutoML service and research tool that produces personalized forecasts on various time-series data, regardless of scientific or industry domain. All users need to do is input the raw data. Applications are boundless, including but not limited to financial/crypto market flows, sales demand/usage, energy/weather predictions, sensors/satellites applications, computer/blockchain networks, and bioinformatics use cases.

See **The problem we are aiming to solve**, **Our Solutions**, and **Milestones** sections in SYBIL's Deep Funding proposal for further details of its problem statement, solution, and milestones respectively [link].

1b) Long-Term Vision

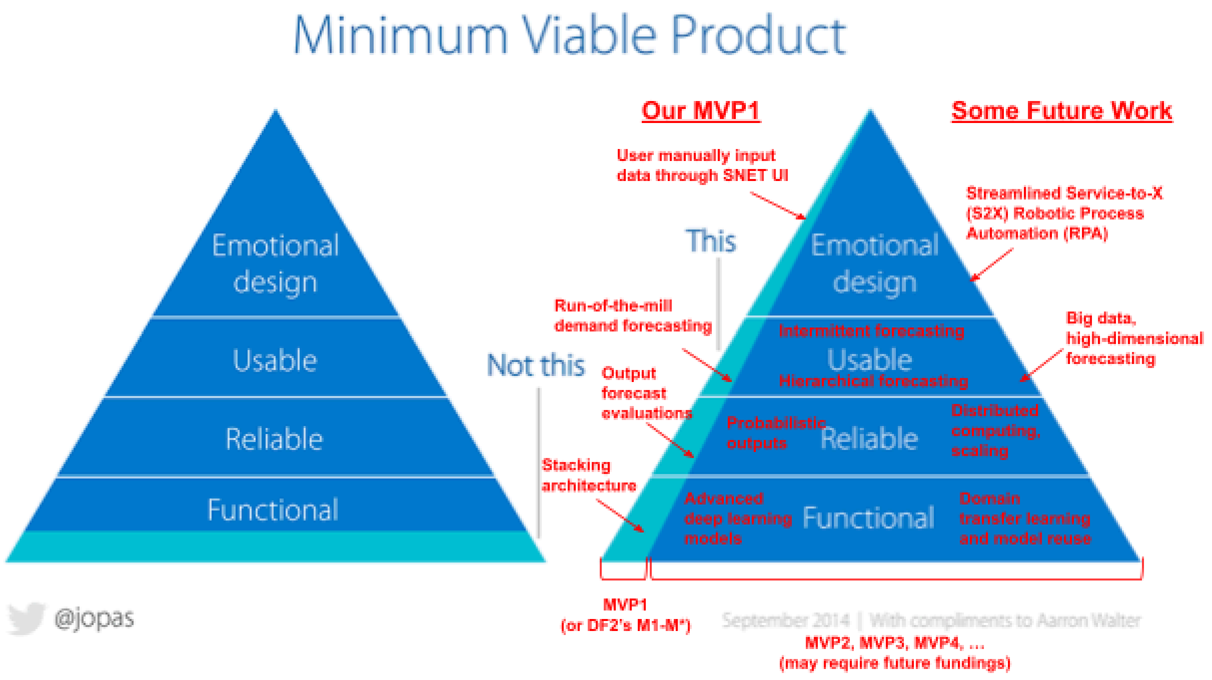
We envision SYBIL as a core infrastructural AI service on the SingularityNET platform. Its general-purpose nature can automatically power any future downstream services, agents, and applications with forecasting needs. In addition to providing reliable forecasting, SYBIL can be used as a research, experimental, and prototyping tool with a suite of evaluation and diagnostic features. This allows end users to analyze and interpret why the forecast has gone in a certain direction and how to improve on its forecasting (e.g., source more data or employ better feature engineering techniques).


Soil Health Ecosystem Modeling App
Crypto Forecasting App
...
Onboard NeuralProphet
SYBIL: The General-Purpose Forecaster
Other SNET Time-Based Services
...
SNET Time-Series Forecasting
Trading Agent
RL Agent
...
Base model Services that feeds into SYBIL
SYBIL feeds into various future Services, Agents, and Applications in various domains.

1c) Goal for DF2

MVP1

As stated in our proposal, our goal for DF2 is to deliver SIBYL v1.0, or **MVP1**, to the SNET marketplace. This includes all of DF2's milestones: from **M1 - M***. It will not contain all of the features to be general-purpose outright, and there will be certain constraints when using MVP1. However, it will be end-to-end and available on the SNET platform. Therefore, we are following the “MVP Pyramid” philosophy, where we develop a bit of everything from core functionalities to the end user experience instead of overloading our SYBIL with features that may or may not see the light of day [link]. This is also why we emphasize having alpha users participate with us throughout our development process.



Minimum Viable Product
Emotional design
Usable
Reliable
Functional
@jopas
Our MVP1
Some Future Work
User manually input data through SNET UI
This
Not this
Streamlined Service-to-X (S2X) Robotic Process Automation (RPA)
Run-of-the-mill demand forecasting
Intermittent forecasting
Hierarchical forecasting
Big data, high-dimensional forecasting
Output forecast evaluations
Probabilistic outputs
Distributed computing, scaling
Stacking architecture
Advanced deep learning models
Domain transfer learning and model reuse
MVP1 (or DF2's M1-M*)
September 2014 | With compliments to Aarron Walter
MVP2, MVP3, MVP4, ... (may require future fundings)

As shown in the rightmost pyramid, we have a laundry list of desired features that will likely not make the cut for MVP1, from additional deep learning models to specific forecasting niches. See the Future Works section for more.

Constraints

Data Requirements

Due to constraints for MVP1, we have set the following dataset requirements:

- Time component with constant frequency (i.e., monthly, daily, hourly)
- Range from 100-10,000 rows (or up to 100,000)
- Strong known trend and seasonality are ideal
- Very near-term forecasting requirements (i.e., nowcasting)

Therefore, MVP1 is not good for:

- Big data, or over 100,000 rows
- High dimensional or high-frequency data
- Timestamped data, without regular frequency
- Streaming or real-time data

Target Use Cases

Environmental Forecasting:

- Climate

- Weather & Temperature
- Air Quality
- Energy Production

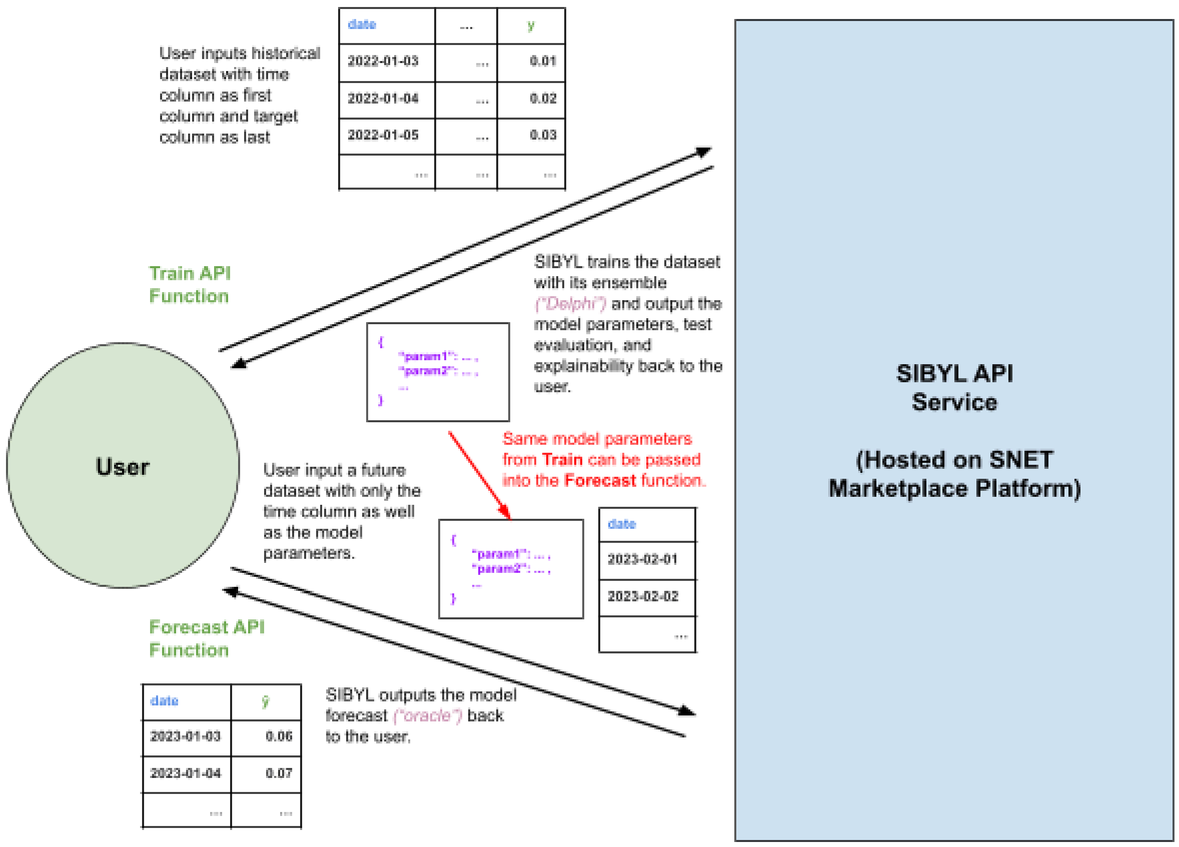
Demand Forecasting (across various domains):
- Energy consumption (overlap with energy production)
- Retail sales forecasting
- Decentralized Finance (DeFi)

Non-target cases:
- High-frequency trading
- User traffic
- Anomaly detection
- Network traffic
- Intermittent or sparse forecasting
- Hierarchical forecasting

In summary, SYBIL's **MVP1** is akin to a training wheel phase. Although the MVP1 will work on various domain environments under certain conditions, it may not necessarily work in all types of "real world" time series.

2) API Service

2a) Schema


User inputs historical dataset with time column as first column and target column as last
date | ... | y
2022-01-03 | ... | 0.01
2022-01-04 | ... | 0.02
2022-01-05 | ... | 0.03
... | ... | ...
Train API Function
SIBYL trains the dataset with its ensemble ("Delphi") and output the model parameters, test evaluation, and explainability back to the user.
{ "param1": ... , "param2": ... , ... }
User
SIBYL API Service
(Hosted on SNET Marketplace Platform)
Same model parameters from Train can be passed into the Forecast function.
User input a future dataset with only the time column as well as the model parameters.
{ "param1": ... , "param2": ... , ... }
date
2023-02-01
2023-02-02
...
Forecast API Function
date | ŷ
2023-01-03 | 0.06
2023-01-04 | 0.07
... | ...
SIBYL outputs the model forecast ("oracle") back to the user.

SYBIL contains a single service with two API functions to call:

1) **Train:** users input historical dataset for SIBYL to train with its stacking architecture and outputs model parameters back to the user in JSON or another compatible file format.
2) **Forecast:** user inputs the future dataset and model parameters to SIBYL. SIBYL then outputs the model forecasts of the future dataset back to the user.

See SYBIL's Deep Funding proposal for further details, plus an illustrative example [link].

Here are the external services that SYBIL may be dependent on:

1) Onboard NeuralProphet
2) Time Series Forecasting (optional)

See SYBIL's Deep Funding proposal for further descriptions about these two services [link].

Train Function

Input

A tabular dataset with a target (or *y*) column, which is used to train the SYBIL stacking architecture. It is passed in as a JSON or other serialized format. Example of input data:

date	...	y
2022-01-03	...	0.01
2022-01-04	...	0.02
2022-01-05	...	0.03
...	...	...

Process

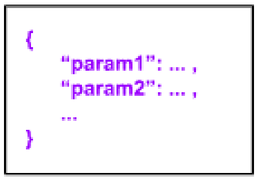
Train function goes through the following steps:

1) Preprocessing: partition, clean, impute, and transform the data so that it is ready to be trained on the base models in the stacked architecture
2) Model: stacked architecture training step by both base and meta-models
3) Evaluation: evaluate stacked architecture performance on the test set of the input data

Further details can be found in sections 3), 4), and 5).

Output

The model parameters for the trained stacked architecture. It may also contain the evaluation performance and metrics on the test set. Like the input, the parameters are also outputted as a JSON or other serialized format, as this is the API response. Example of output parameters:



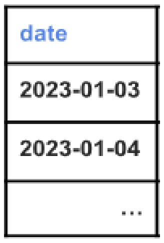
```
{
    "param1": ... ,
    "param2": ... ,
    ...
}
```

It is also possible that the output is the serialized form of the saved model artifact (typical in JBL format).

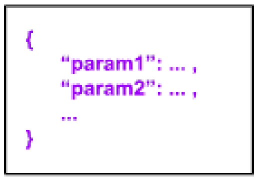
Forecast Function

Input

A tabular dataset *without* a target (or y) column *and* the model parameters from the **Train** function. Example of input data and parameters:



date
2023-01-03
2023-01-04
...



```
{
    "param1": ... ,
    "param2": ... ,
    ...
}
```

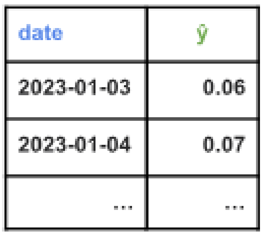
Process

Forecast function mirrors a lot of the Train function, except it omits the steps where hyperparameter tuning is required, as the model parameters are already passed in.

1) Preprocessing: clean, impute, and transform the data so that it is ready to be forecasting base models in the stacked architecture. However, it does not split the data.
2) Model: Stacked architecture loads the existing model parameters and uses that to make model forecasts.

Output

The output is the same tabular dataset as the input but *with* the forecasted target column (or ŷ). It is JSON or other serialized format as this is the API response. Example of output table:



date	ŷ
2023-01-03	0.06
2023-01-04	0.07
...	...

2b) Infrastructure

AWS

For SYBIL milestones M2-M4, we will deploy our current service iteration into the AWS cloud, most likely a single EC2 instance. We may use one of the AWS deployment services, such as CodeDeploy or Elastic Beanstalk. In addition, we will provide an API endpoint link that allows users to execute **Train** and **Forecast** functions through REST API. They are POST requests the user must pass in a JSON body as either the train data or model parameters.

SNET

For SYBIL milestones M5 and onward, we will migrate our service from AWS to SNET platform. Regarding our status for understanding the SNET API Schema, we have read through the Full Guide [link], AI Developers Docs [link], and the DApp UI Component Docs [link] and forwarded any of our questions to the SNET developers. However, we are also aware that SNET is currently ongoing a platform overhaul on an infrastructural level. We await updates on the new deployment process and how to deploy on the test net.

Tech Stack

We will use Python (~v3.9) and some web/Javascript primarily when creating the front end of our service in the SNET platform. Python libraries that we may use include but are not limited to: *numpy, pandas, scikit-learn, statmodels, tsfresh, statforecast, functime, sktime, darts, neuralprophet, pytorch-lightning, and tensorflow*.

3) Train Function - Preprocessing

3a) Fundamental Properties of Time-Series

Here are the fundamental properties of time-series:

1) **Level**: Mean or y-intercept of the time-series
2) **Trend**: long-term growth (or slope) of the time-series
3) **Seasonality**: repeated patterns *with* a fixed period
4) **Cyclicity**: repeated patterns *without* a fixed period
5) **Noise**: residual or randomness in the time-series that cannot be modeled

3b) Splitting a Time-Series into Train, Validation, and Test Sets

In machine learning, the train/validation/test split is a crucial technique used to train a model, tune its hyper-parameters, and assess its performance on unseen data. It is performed by dividing a given dataset into three subsets: the training/validation/test sets. When applying this technique in time series forecasting, special attention is required to handle temporal dependencies and prevent issues like look-ahead bias, as the sequential nature of time series data can significantly impact the model's performance and generalization ability. First, the test set should consist of the most recent samples, constituting roughly 10-20% of the data, kept completely separate from the model development/tuning process, and used only to assess the model's performance on entirely new, unseen data. Among the remaining samples, the last 10-20% should be used for validation and hyper-parameter tuning, and the rest for training the model.

Mapping a time-series forecasting problem into a supervised regression problem allows one to use various regression-based ML methods for forecasting purposes. This mapping generates feature-target (X-y) value pairs from a given time series data. For example, imagine a univariate time-series [1, 2, 3, 4, 5], where the objective is to forecast the next time step using the last two steps. Here, the X-y pairs will be [1, 2] - [3] ; [2, 3] - [4] ; and [3, 4] - [5]. It is important to perform this mapping separately for the train/validation/test splits generated using the above instructions.

Let's explain this further using an example. Imagine a forecasting problem where a 1-year-long time-series of 1-hour granularity is given, and the objective is to train a model that, at any given point in time, can forecast the next 24 hours. We must follow these steps:

1) Initialize train, validation, and test splits. They can be, for example, months [1-8], [9-10], [11, 12], respectively.
2) Separately convert each split into its corresponding (X-y) pairs to get (X_train, y_train), (X_valid, y_valid), (X_test, y_test).
3) Use (X_train, y_train), (X_valid, y_valid) to train and tune the model.
4) Retrain the model on (X_train, y_train) appended with (X_valid, y_valid) using the tuned hyper-parameters.
5) Use the model to forecast the first 24 hours within the test split.
6) Perform the so-called rolling forecast with a sliding window of width 24h. This means the border separating previous train and validation sets and previous validation and test sets are moved forward by 24h, giving new train, validation, and test splits.
7) Go to Step 2) until reaching the last 24 hours of the year.

Note that the so-called time-series cross-validation process discussed in, for example [link], is a special case of the above process where there is no validation set. The width of the sliding window is 1.

3c) Data Cleaning

Data cleaning is another important pre-processing step in time series forecasting, especially in scenarios involving daylight saving time (DST) shifts, leap years, or datasets involving timestamps belonging to different time zones. DST results in having a missing/duplicate hour at the start/end of DST and can be handled by, for example, 1) introducing a three-level categorical variable that takes values of -1/1 when DST starts/ends and 0 elsewhere; 2) removing the duplicate timestamp present in the days that DST ends and interpolating to get a value for the missing hour present in the days that DST starts. Leap years can be handled by introducing a two-level categorical variable. Datasets with timestamps from different timezones can be handled by simply converting all timezones to UTC.

3d) Decomposition

In time-series data, trend, seasonality, and cyclicity patterns are common. Trend refers to a long-term decrease or increase in the series. Seasonality refers to ups and downs with known and fixed frequency (amplitude does not need to be fixed). Cyclicity refers to rises and falls with varying frequency. These components can be separated using decomposition methods such as classical, X11, SEATS, and STL.

In classical decomposition, which can be additive or multiplicative, the first step is to estimate the trend-cycle component by removing seasonality using moving average filtering. These filters reduce seasonality and randomness in the series. More specifically, a moving average filter of order m, denoted by m-MA, eliminates an order m seasonality. When m is an odd number equal to 2*k* + 1, m-MA will be a symmetric average consisting of *k* observations on either side; when m is even, such averaging will not be symmetric anymore, and, to make it symmetric, two moving averages of order m need to be applied one after another. This is typically denoted by 2×*ma*. Once the trend-cycle component is estimated, it should be subtracted from the original series to get the detrended series. Then, the seasonal component for each season is obtained by averaging the de-trended values for that season across the entire series. For instance, when dealing with monthly data, the seasonal component corresponding to January is the average of all detrended January values within the series [link]. The seasonal component *i* is then obtained by concatenating all seasonal values.

Classical decomposition has the following shortcomings:

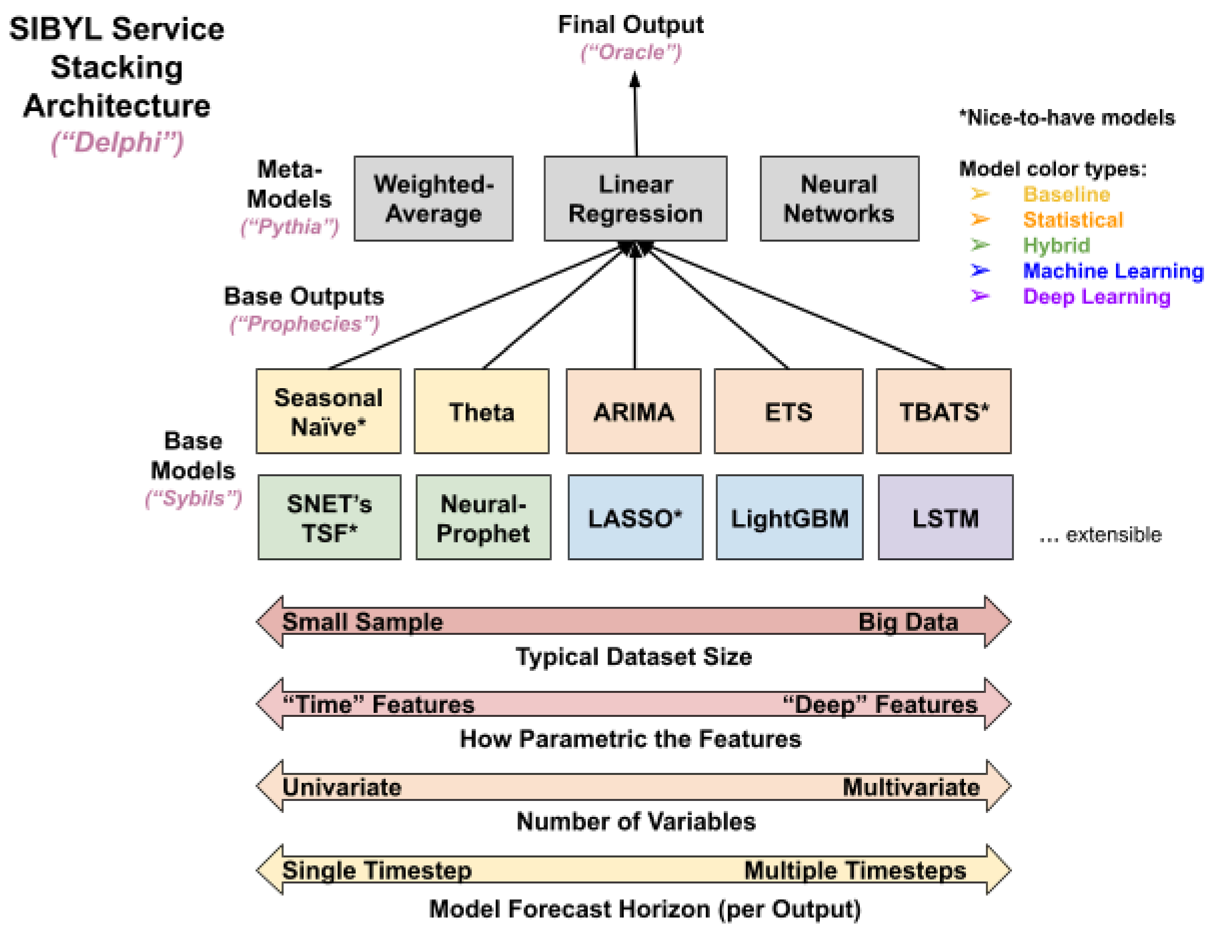
1) The estimate of the trend cycle is unavailable for the first few and last few observations.
2) The trend-cycle estimate tends to over-smooth rapid rises and falls in the data; therefore, such variations end up leaking into the remainder.
3) It assumes that the seasonal component repeats itself once all the seasons have been visited. In other words, it cannot capture changes over time in the seasonal component from year to year (year here refers to a period where we have visited all the seasons).
    a) For example, the seasonality component for March 2010 is the same as for March 2020. This creates a problem for some long series.

More advanced decomposition methods such as X11, SEATS, and STL try to address these shortcomings. We are considering whether to include these advanced methods in SYBIL and will further elaborate on them in this report if we do so.

4) Train Function - Models

4a) Architecture

Here is the stacked generalization architecture, also known as the *stacking ensemble*:



4b) Base Models

These are the base models that we plan to implement and perform experiments on to be incorporated into the final model.

Base Models:

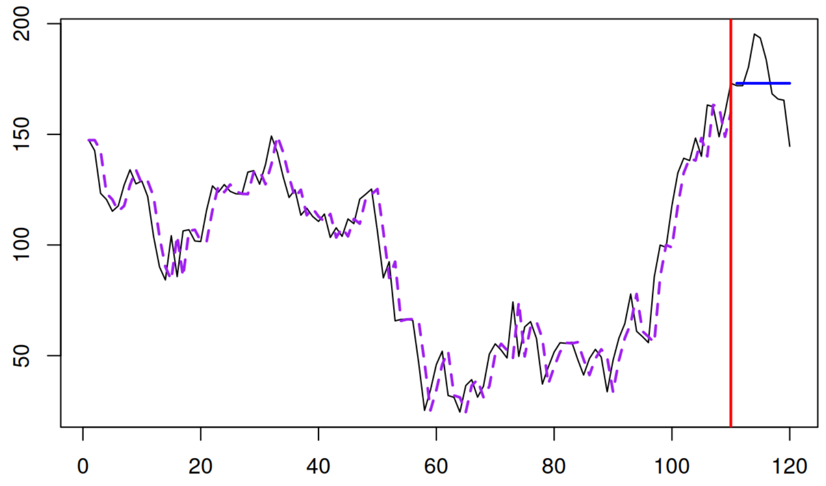
1) Seasonal Naïve* (M2)
2) An ETS Model (M3)
3) Theta (M2)
4) ARIMA (M3)
5) TBATS* (M3)
6) SNET's Time Series Forecasting Service* (M5)
7) NeuralProphet Service (M6)
8) LASSO* (M7)
9) LightGBM (M7)
10) LSTM (M7)

*Note: these models are optional, or nice-to-haves, in the final model.

Seasonal Naïve (Optional)

A *naïve* method uses the previous value y_{t-1} to forecast the next value y_t. Therefore:

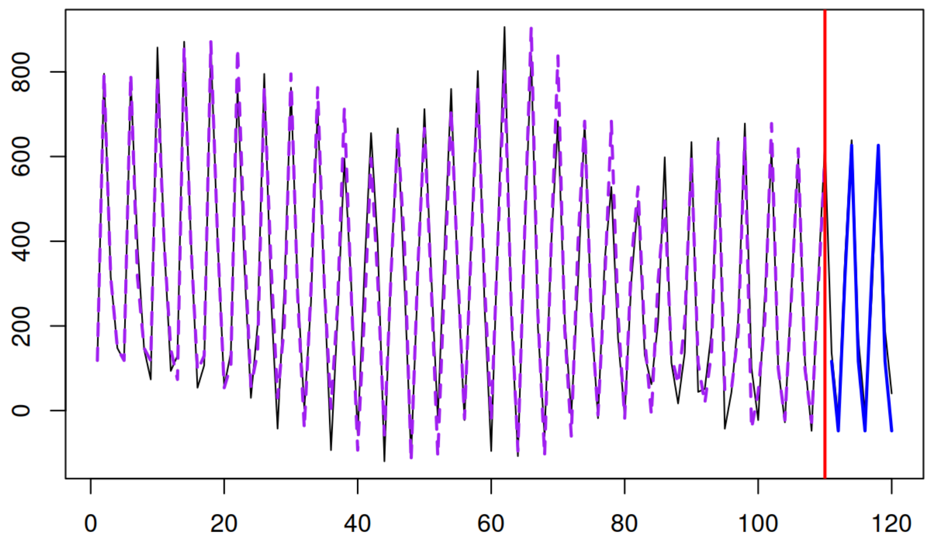
$\hat{y}_t = y_{t-1}$



The graph above shows a *naïve* forecast. The red line is the cutoff between historical (train) and future (predict) data. The blue line is the naïve forecast $\hat{y}_t$ itself, which is horizontal from the last data point y_{t-1}.

A *seasonal naïve* method uses the value from last season $y_{t-m,}$ to forecast the next value y_t. For example, if an hourly time-series has daily seasonality (i.e., electricity consumption) and the last time *t-1* is 5 PM, this method uses yesterday's 6PM to calculate today's 6PM. Therefore:

$\hat{y}_t = y_{t-m}$



The graph above shows a *seasonal naïve* forecast. The seasonal cycle is one interval between the peaks or the valleys. In this case, the forecasted values $\hat{y}_t$ closely follow the actual values y_t

because the dataset maintains constant seasonality mean and magnitude. Although this is not always the case in real-world datasets, *seasonal naïve* is still an ideal baseline model for datasets with a seasonal component. See this article for additional details [link].

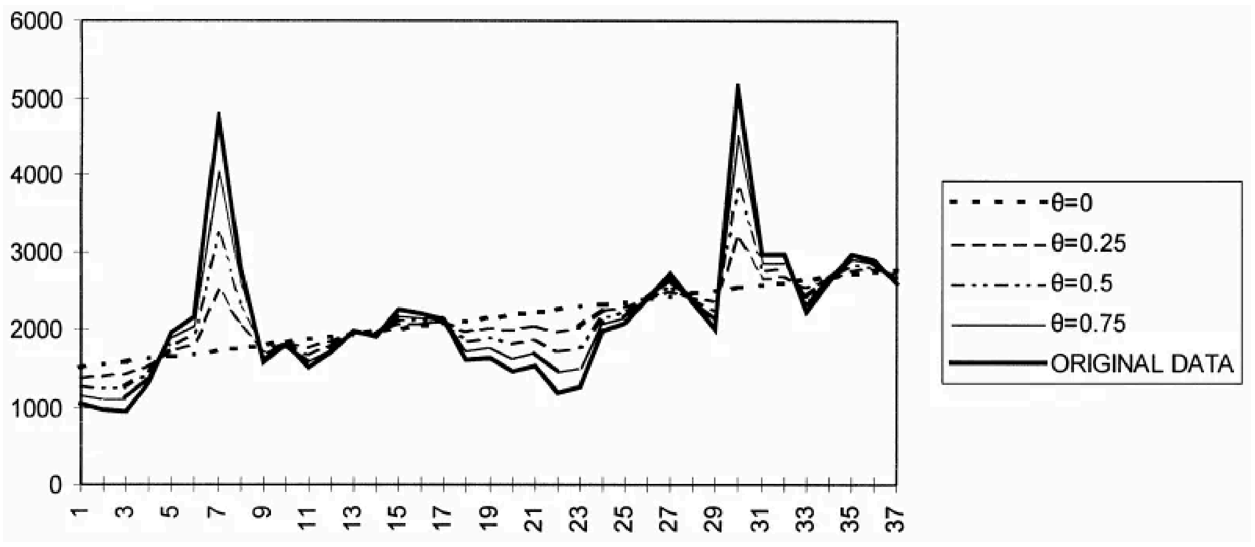
We consider implementing both *naïve* and *seasonal naïve* models during Milestone 2 (M2). Alternatively, they also are used as the scaling coefficient for Mean Absolute Scaled Error (MASE) performance metric (see the Model Evaluation - Evaluation Metrics Description section).

Theta

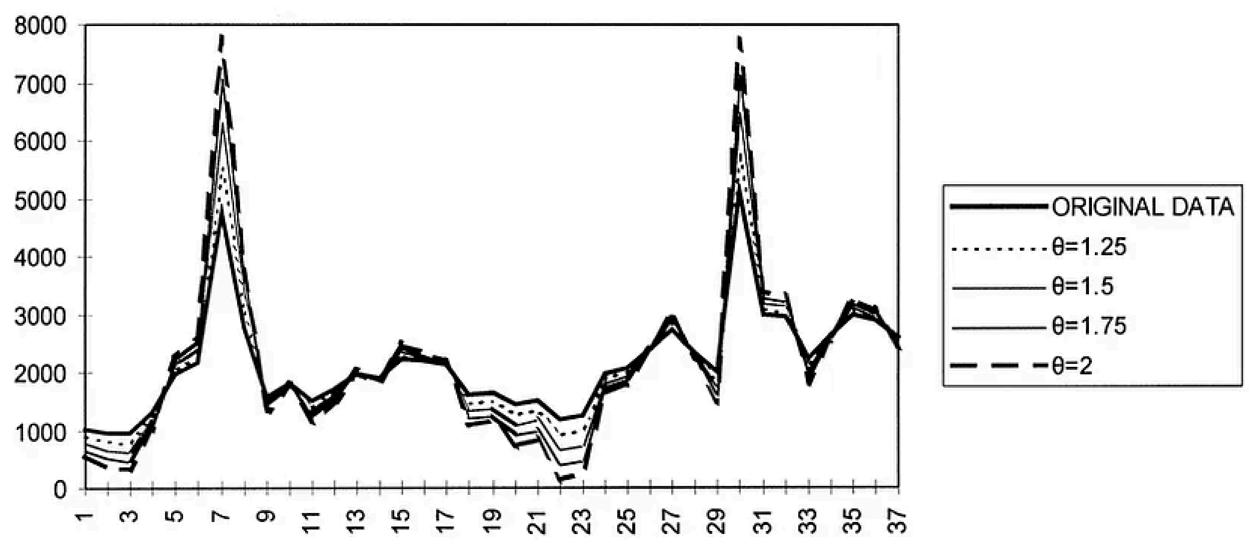
Theta method is an univariate forecasting which decomposes the original data into at least two *Theta lines*. The Theta lines are estimated by modifying the *curvatures* of the original time-series, or the distances between the time-series data points. This modification is managed by a parameter called *theta* (θ). It is applied on the second difference of the time-series (or after being differenced twice). Practically speaking, the parameter θ is a transformation parameter that creates synthetic time-series with the same mean and slope of the original data but different variances [19].

θ means the following:

- When $\theta = 0$, the time-series transforms into a linear regression line, usually a middle line that crosses through the time-series.
- When $0 < \theta < 1$, the time-series short-term fluctuations are smaller, emphasizing more long-term effects. In other words, the peaks and valleys are typically less pronounced than in the original time-series.
- When $\theta > 1$, the time-series short-term fluctuations are magnified, emphasizing the short-term effects. In other words, the peaks and valleys are typically more pronounced than in the original time-series.



When $\theta > 1$, the transformed time-series emphasizes more long-term effects, making it more “deflated” or “smoothed” than the original time-series.



When $\theta < 1$, the transformed time-series emphasizes more on short-term effects, making it more "inflated" or "peaked" than the original time-series.

If seasonality exists in the time-series, it must be first deseasonalized through a decomposition method (i.e., STL) before applying the Theta method. The seasonality will subsequently be added back. See this article for additional information [link].

Implementation in SYBIL

We plan to implement this model during Milestone 2 (M2) as the baseline model due to its simplicity and effectiveness.

ETS

The Error, Trend, Seasonal (ETS) models are a family of time series models with an underlying state space model consisting of the following components [link]:

- level component
- trend component (T)
- seasonal component (S)
- error term (E)

ETS models can be referred to as ETS(error, trend, seasonal), where all of its components can be None (N), Additive (A), or Multiplicative (M). For example, an ETS(A, M, N) model has an additive error, multiplicative trend, and no seasonality.

Exponential Smoothing is an ETS model subtype that combines Error, Trend, and Seasonal components into a smoothing function [link]. See [1] for the taxonomy of exponential smoothing methods.

In short, the concept of exponential smoothing is to have a weighted moving average (WMA) for all historical observations on the future prediction, but exponentially decay the weight the older the observation is or the further back from the current time *t*. Therefore, more recent observations have a disproportionate weight on the next forecast, while older observations have smaller until nearly negligible weights on the forecast.

<u>Single Exponential Smoothing (SES)</u>

Single exponential smoothing (SES) has one smoothing parameter *alpha* (α), ranging from 0 to 1. The smaller the α, the more weight the previous observations have on the final forecast $\hat{y}_t$.

$$\hat{y}_t = \alpha \cdot y_t + (1-\alpha) \cdot \hat{y}_{t-1}$$

<u>Triple Exponential Smoothing (TES) or Holt-Winter's Method</u>

More complicated methods like the triple exponential smoothing (TES), aka Holt-Winter's Method, account for both trend and seasonality in the weighted averaging. In Holt-Winter's method, there are three smoothing parameters: *alpha* (α) for the level, *beta* (β) for the trend, and *gamma* (γ) for the seasonality.

$$\begin{aligned} \ell_x &= \alpha(y_x - s_{x-L}) + (1-\alpha)(\ell_{x-1} + b_{x-1}) \\ b_x &= \beta(\ell_x - \ell_{x-1}) + (1-\beta)b_{x-1} \\ s_x &= \gamma(y_x - \ell_x) + (1-\gamma)s_{x-L} \\ \hat{y}_{x+m} &= \ell_x + mb_x + s_{x-L+1+(m-1)modL} \end{aligned}$$

All of these parameters have values ranging from 0 to 1. The higher the value, the more the parameters decay their respective terms so that the older observations have less impact. The user may need to automatically parameter tune to get their optimal parameter values.

The three smoothing functions are the following:
- L_x: level or y-intercept of the series
- B_x: slope of the series
- S_x: seasonality component of the series

The final forecast $\hat{y}_{x+m}$ is the composite of these three smoothing functions.

<u>Implementation in SYBIL</u>

For SYBIL, we will be experimenting and implementing the following types of exponential smoothing methods:
- Deseasonalize + Double Exponential Smoothing (or ETS(A, A, N))
- Triple Exponential Smoothing, aka Holt-Winter's Method (or ETS(A, A, A))

That way, our ETS approach will cover all of the ETS components available. We will first assume that the components are additive before including the multiplicative feature. We plan to implement this model during Milestone 3 (M3).

<u>ARIMA</u>

ARIMA is a combination of the Autoregressive (AR) model, Moving Average (MA), with differencing (I).

Autoregressive Model (AR)

An AR model uses lagged values, or previous values, with their respective coefficients to forecast the next value y_t. This model can be written as:

$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$ [1]

Where c is the constant (or *y*-intercept), ϕ is the coefficient of the respective lag values, and ε is the error term. This formula has a total of *p* lag values, so this model is also referred to as AR(*p*). The Autoregressive Model differs from the Simple Exponential Smoothing (SES) in terms of the following:

- SES uses all of the historical observations (albeit exponentially decayed) while AR uses a *p* latest historical values (as lagged values)
- AR fits each of the ϕ coefficients for their respective lagged values while SES uses singular α parameter the tune all of the coefficients.

Moving Average Model (MA)

In the ARIMA model, the AR already assigns coefficients in the past values $y_{t\ldots t-p}$. Therefore, the MA model instead models the past errors $\varepsilon_{t\ldots t-q}$:

$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$ [1]

Where the ε_t is the current error, which cannot be modeled, and θ is coefficient of the respective error terms. This formula has a total of *q* lag errors, so this model is also referred to as MA(*q*). *p* and *q* can be different values and are both tunable by the user. AR and MA can be combined to form ARMA(*p*, *q*). It is equivalent to ARIMA(*p*, 0, *q*) without the differencing.

Differencing (I)

As mentioned in the Preprocess section, predicting when the series is stationary is easier due to the constant mean and variance over time. ARIMA has its built-in method to stationarize the time-series called *differencing* [link]. Differencing is simply computing the differences between consecutive time-series observations [1]. The first-order differenced time-series can be written as:

$y'_t = y_t - y_{t-1}$

In the ARIMA model, both the AR and MA components will then model the differenced y'_t instead of the original y_t. The parameter that tunes the amount of differencing is *d*, and here *d*=1. The second-order differenced time-series can then be written as:

$y''_t = y'_t - y'_{t-1}$

Where *d*=2. Typically, you select *d* based on the number of times differencing until there is a reasonable chance that the transformed time-series is no longer non-stationary. If the time-series is already stationary, then no differencing is needed; hence *d*=0. So this is simply an ARMA model.

ARIMA model

To summarize, an ARIMA(*p*, *d*, *q*) has the following three hyperparameters:

- *p*: number of past (or lagged) y-values
- *d*: degree of differencing (to be stationary)
- *q*: number of past (or lagged) error values

Once those three hyperparameters are selected, ARIMA will then fit and find the best AR and MA coefficients (or parameters themselves). Auto-ARIMA models will automatically hyperparameter tune *p*, *d*, and *q*.

What is discussed above is the base version of ARIMA. Here are some important extensions of ARIMA:

- ARMA - ARIMA without differencing (*d*=0)
- SARIMA - with the S that account for seasonality
- SARIMAX - with the X that accounts for exogenous variables
- VARMA - with V for vector autoregression, which allows for multi-step forecasting

Implementation in SYBIL

We will be implementing a fully automated and general-purpose version ARIMA. That is SARIMAX, with ARIMA plus the seasonality and exogenous variables, and its equivalent for VARMA, which is for multi-step forecasting. We plan to implement this model during Milestone 3 (M3).

TBATS (Optional)

TBATS is an end-to-end model that stands for the following [link]:

- **T**rigonometric seasonality
- **B**ox-Cox Transformation
- **A**RMA Error
- **T**rend Components
- **S**easonal Components

This model is best suited for forecasting complex seasonal time-series including:

- multiple seasonal periods
- high-frequency seasonality
- non-integer seasonality
- dual-calendar effects [21]

Implementation in SYBIL

We may plan to implement this model during Milestone 3 (M3), depending on its computational complexity. Hence why this is flagged as optional.

SNET's Time Series Forecasting Service (Optional)

SNET has an existing Time Series Forecasting (TSF) service [link], with the underlying source code called time-series-analysis [link]. TSF contains the following two generic models:

1) CNTK's LSTM
2) Facebook's Prophet

LSTM

LSTM developed from Microsoft Cognitive Toolkit (CNTK) open-sourced library (from Microsoft Azure). See the LSTM section for more details.

Facebook's Prophet

Prophet is a decomposable time-series with three primary model components: *trend*, *seasonality*, and *holidays*.

$$\hat{y}_t = g(t) + s(t) + h(t) + \varepsilon_t$$

- *g(t)*: trend function which models non-period chances in the value
- *s(t)*: periodic or seasonal changes
- *h(t)*: effects of holidays which occur irregular schedules over one or multiple days
- ε_t: error term, or any idiosyncratic changes (i.e., white noise) that cannot be modeled. Prophet makes a parametric assumption that error is normally distributed.

The trend function contains two trend models: a *piecewise linear model* that goes on a constant rate but can directions at set points, and a *nonlinear growth model* that captures more parabolic trends. The seasonal function uses Fourier series to provide a more generalizable model to capture multiple seasonal effects. Lastly, holidays and their countries of origin are included as categorical exogenous variables.

Prophet frames the forecasting problem as purely a *curve-fitting* exercise, unlike the time-series models that explicitly account for the temporal dependence (i.e., autoregression). While Prophet may lose important information from lagged regressors, it offsets it by providing practical forecasting advantages such as *speed*, *flexibility*, and therefore *scalability*. [22]

Implementation in SYBIL

We may implement this model during Milestone 5 (M5). It is optional at the moment, depending on a) the ease of connecting one SNET service (i.e., SNET's TSF -> SYBIL) and b) potential redundancy as SYBIL already contains NeuralProphet, the advanced version of Prophet, and LSTM.

NeuralProphet Service

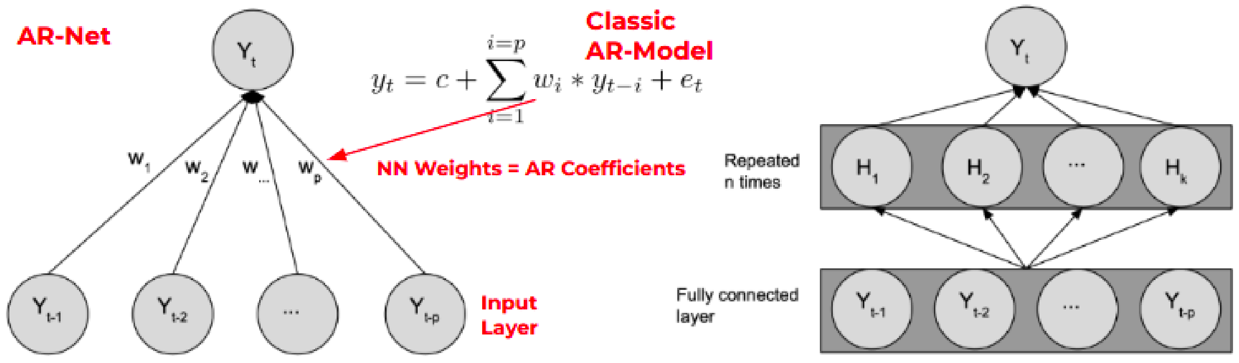
NeuralProphet is an extension of Prophet. Like Prophet, NeuralProphet also contains *trend*, *seasonality*, and *holiday* components. However, NeuralProphet also contains lagged and future regressors, whether of target series (i.e., autoregression) or exogenous variables. Here are the 6 model components [23]:

$$\hat{y}_t = T(t) + S(t) + E(t) + A(t) + L(t) + F(t) + \varepsilon_t$$

- T(t): trend
- S(t): seasonal effect at time *t*
- E(t): event and holiday effects
- A(t): autoregression on past observations
- L(t): regression effect for lagged observations of exogenous variables

- F(t): regression effects for future-known exogenous variables
- ε_t: error term

For trend, NeuralProphet uses a similar method as Prophet but also allows the growth model to change in different points in times, not only the linear model. For seasonality, NeuralProphet uses the same Fourier terms method as Prophet. As for autoregression, lagged, and future regression of exogenous variables, NeuralProphet uses a custom feed-forward neural network (FFN) called AR-NET where the neurons of the input layer represent the regressors (i.e., $Y_{t-1 \dots t-p}$) and weights (i.e., $w_{1 \dots p}$) represents their respective coefficients (just like $\phi_{1 \dots p}$ in an ARIMA model). A potential benefit of AR-Net over classical AR models like ARIMA is its addition of multiple hidden layers, which can be used to capture nonlinearities of the coefficients and the interactions between the regressors themselves [24]. Whereas AR models treat each regressor to be independent of each other.



For more details about Neural Networks (NN), see the LSTM section below.

Implementation in SYBIL

First, we will implement NeuralProphet as a separate SNET Service, as detailed in this *Onboard NeuralProphet* proposal [link]. Once that service is completed, we will integrate it into SYBIL during Milestone 6 (M6). This will become one of the first Service-to-Service (S2S) instances on the SNET platform, where once the end-user triggers SYBIL, if SYBIL needs to use NeuralProphet as a base model, then it will automatically call on NeuralProphet service to execute the model and provide results.

LASSO (Optional)

Linear Regression

Linear Regression (LR) is a widely used statistical technique for predicting an outcome variable based on one or more predictor variables. It assumes a *linear* relationship between the input and output variables and estimates the coefficients that minimize the sum of the squared residuals. In our meta-modeling context, LR is employed to understand and model the linear relationship between the base models' predictions and the true target values. Here is a formula of a typical LR model:

$$\hat{y}_t = c + \beta_1 * X_1 + \beta_2 * X_2 + \dots + \beta_n * X_n + \varepsilon_t \quad = c + \sum_{i=1}^{n} (\beta_i * X_i) + \varepsilon_t$$

Where:

- c is the constant (or y-intercept)
- n is the number of features
- X_i is the ith feature
- β_i Is the coefficient associated with the ith feature
- ε_t is the error that cannot be modeled

The features in LR can be anything that the user inputs, whether they are autoregression $\hat{y}_{t-1}$ or other exogenous variables. Therefore, LR can be considered a more generalized model of other classical time-series models like ETS and ARIMA.

LASSO

Least absolute shrinkage and selection operator (LASSO) regression, also known as L1 regularization, is a modified LR model with an additional penalty term based on the absolute values of the coefficients $|\beta_i|$. This penalty term aims to *shrink* or prevent any coefficient values from getting too big that it dominates the regression model. Here is a formula of a typical LASSO model:

$$\hat{y}_t = c + \sum_{i=1}^{n} (\beta_i * X_i) + \lambda * \sum_{i=1}^{n} |\beta_i| + \varepsilon_t$$

Where lambda (λ) is the regularization parameter that controls the amount of regularization applied. The higher the lambda, the higher the penalty and the more it forces the individual coefficients β_i to shrink [link].

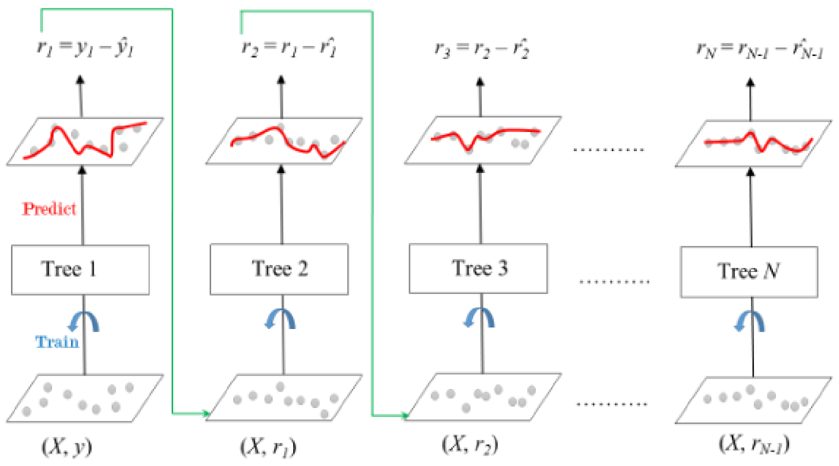
Implementation in SYBIL

We may implement this model during Milestone 7 (M7). It is currently optional due to its simplicity and overgeneralization, so it is a priority. Additionally, the time-series must be tabularized to run through LR.

LightGBM

Gradient Boosted Trees

A boosting algorithm is where a weaker learner (or model) can be sequentially modified to become a stronger, more sophisticated model [link]. Gradient Boosted Trees (GBT) is a boosting algorithm that uses CART (Classification and Regression Decision Trees) as the base learner. The first CART learner fits and predicts ($\hat{y}$) on the data (y), then the subsequent CART fits on the residual (r) of the previous CART. In other words, the succeeding CART models focus on the errors that preceding CART models made until the final Nth CART model, which, combined with the previous CARTs, makes a much stronger GBT model that leaves no stones unturned. Here is a visualization of a GBT model in action [link]:



Although GBT sounds very powerful on paper, it runs into computational complexities in practice due to its sequential nature and difficulty in parallelizing. This complexity compounds the more data features or more trees are present. One example is for every feature, GBT needs to scan all data points to find the best information gain for possible split points. This is a very time-consuming process, especially the larger the dataset.

LightGBM

LightGBM proposes two novel techniques to tackle this complexity problem:
1) Gradient-based One-Side Sampling (GOSS)
2) Exclusive Feature Bundling (EFB)

Based on experiments on multiple public datasets, LightGBM can speed up the training process compared to conventional GBT by 20x while achieving similar accuracy. [25]

LightGBM's other advantages include:
- Requiring lesser memory
- Compatibility with larger and more complex datasets
- Support for both parallel learning and GPU learning [link]

In conclusion, out of the available Gradient Boosting methods, we selected LightGBM to be one of its base models.

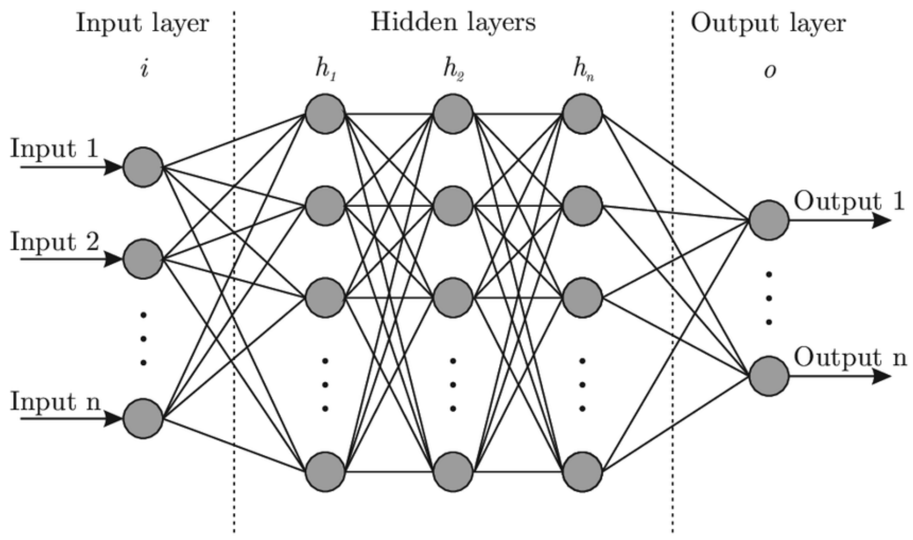
Implementation in SYBIL

We plan to implement this LightGBM during Milestone 7 (M7).

LSTM

Neural Network (NN)

Neural networks (NN) are one-directional layers of neurons (or nodes) that capture "deeper" representations of the input data. A typical NN contains an input layer, an output layer, and a *N* number of hidden layers. They are a type of machine learning algorithm modeled after the human brain, designed to recognize patterns. NN can capture both linear and non-linear relationships, giving them the flexibility to model complex patterns between the base models' predictions and the true target values. The most basic type of neural network is the Feed-Forward Neural Network (FFNN) or Dense Neural Network, where all neurons from one layer connect to all those in the next layer. You can see an illustration of an FFNN below:



The composite output Y from a Shallow Neural Network is calculated as follows:

$$\hat{y} = f(\sum_{i=1}^{n} (W_i * \hat{y}_i) + b)$$

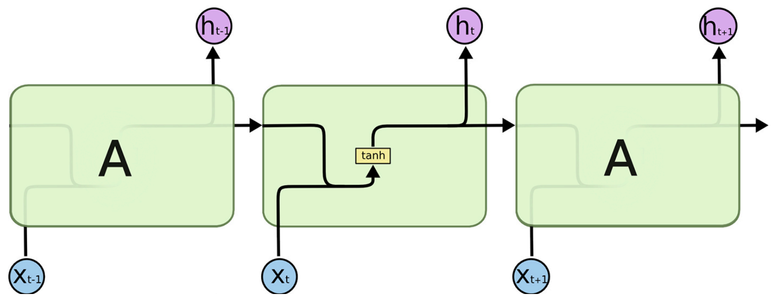
Where:

- $\hat{y}$ is the final prediction
- W_i is the weight for the ith model's prediction
- $\hat{y}_i$ is the prediction of the ith model
- b is the bias term
- f is the activation function

Recurrent Neural Network (RNN)

Recurrent neural network (RNN) is a special type of NN that contains “memory blocks” instead of neurons. They are designed to handle longer sequences of data (whether input, output, or both), thus preserving and capturing temporal continuity. RNN becomes the ideal NN model for time-series data.

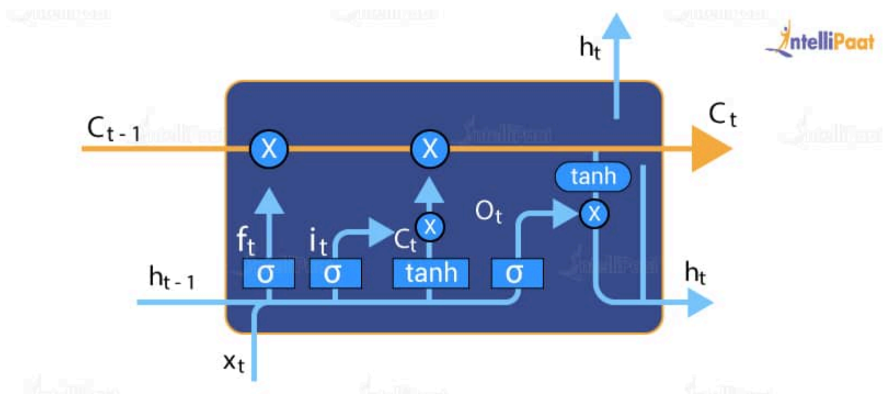
Below is an illustration of RNN, where a feature *X* at different timestep *t-1*, *t*, and *t+1* are inputted into their respective “memory blocks”, and output the values *h-1*, *h*, and *h+1*. This is an example of a sequence-to-sequence (*seq2seq*) model, ideal for *direct* multi-step forecasting that does not require *recursive* forecasting.



One shortcoming of a vanilla RNN is the *vanishing gradient* problem, where the model learnings and updates get telegraphed less and less to the "memory blocks" that are earlier in the sequence. This makes RNN more likely to discard or "forget" older or more historical data. Overcomes the vanishing gradient problem or "forgetting" from RNNs

Long-Short Term Memory (LSTM)

Long Short-Term Memory (LSTMs) is a variant of the RNN model that overcomes the vanishing gradient problem or "forgetting" from RNNs. Unlike RNN, LSTM contains an additional "cell state" *C* (horizontal orange line) that better preserves its state over time. It is guarded by mathematical gates (blue lines) that determine how much of the new information X_t influences *C*. Below is an illustration of an LSTM cell [link]:



Implementation in SYBIL

We plan to implement an LSTM model during Milestone 7 (M7).

4c) Meta-Models

Meta-models are a way to combine the predictions of multiple base models to improve the overall accuracy of the forecast. In this project, we will explore different families of meta-models that will aggregate the underlying base models predictions:

1) Weighted Average (WA, developed in M3)
2) Linear Regression (LR, developed in M4)
3) Shallow Neural Network (NN, developed in M4-6)
4) Random Forest (RF, developed in M4-7)

Weighted Average (WA)

The Weighted Average (WA) is a simple yet effective meta-model that combines the predictions of different base models by assigning them different weights. This approach takes its roots from the theory that multiple experts' forecasts, when combined, can result in an accurate forecast, a phenomenon referred to as the "wisdom of crowds". As documented in the forecasting literature, the simple average tends to be a surprisingly robust forecast. This method is known to be robust according to existing literature [7]. Furthermore, its computational efficiency provides a solid baseline for comparison against more sophisticated methods, such as performance-based weights and stacking methods.

The composite output from the WA model is calculated as the sum of the product of each base model's prediction and its corresponding weight. The Weighted Average (WA) method calculates the composite output as follows:

$$\hat{y} = \sum_{i=1}^{n} (W_i * \hat{y}_i) / \sum_{i=1}^{n} W_i$$

Where:

- $\hat{y}$ is the final prediction
- W_i is the weight of the ith model (calculated as the inverse of each model's sMAPE)
- $\hat{y}_i$ is the prediction of the ith base model

The process of getting the evaluation result from base models using WA is as follows:

1. Train the base models on a subset of the total data (base model training data)
2. Calculate each base model's sMAPE (symmetric Mean Absolute Percentage Error) with the remainder of the data (meta-model training data)
3. Calculate the inverse of each model's sMAPE and use it as the base model's weight, thereby assigning a higher weight to models with lower errors
4. Once the weights are calculated, train the base models on the full dataset so no training data is “lost” because of the training of the meta-model
5. Calculate the final prediction by summing the weighted predictions of all the base models

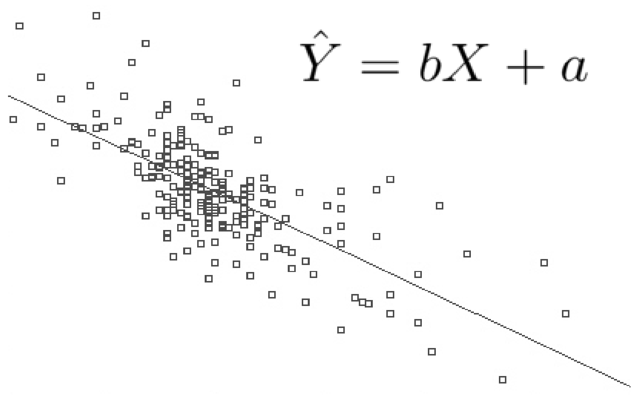
The effectiveness of this meta-model lies in its simplicity and robustness against overfitting. We plan to implement this model during Milestone 3 (M3).

- NOTE: To enhance this naive model, we could experiment with different types of averages (e.g., median) and various CV techniques (like k-fold, time series). These improvements could enhance the model's performance by better incorporating expert forecasts. Moreover, Global Learning - training across multiple datasets, which requires globally shared features, could provide further improvements by leveraging cross-series information.

Linear Regression (LR)

For a brief description of Linear Regression, see the LASSO section above.

While similar to WA in assigning weights to each model's predictions, the weights in LR are determined through regression analysis rather than being inversely proportional to their error measures.



In the context of our meta-modeling, the composite output Y from the Linear Regression model is calculated as follows:

$\hat{y} = \alpha + \beta_1\hat{y}_1 + \beta_2\hat{y}_2 + \ldots + \beta_n{*}\hat{y}_n + \varepsilon$

where:

- $\hat{y}$ is the final prediction from the meta-model
- α is the intercept of the regression line
- β_i is the coefficient for the *i*th base model's prediction
- $\hat{y}_i$ is the prediction of the *i*th base model
- ε is the error term

The process of getting the evaluation result from base models using LR involves:

1. Train the base models on a subset of the total data (base model training data)
2. Calculate each base model's sMAPE (symmetric Mean Absolute Percentage Error) with the remainder of the data (meta-model training data)
3. Fit a linear regression model with the base models' predictions as input features and the true target values as the output.
4. Once the LR model is fitted, train the base models on the full dataset so no training data is "lost" because of the training of the meta-model
5. Calculate the final prediction by inputting the base model predictions in the LR model

We plan to implement this meta-model during Milestone 4 (M4).

Neural Network (NN)

For a brief description of Neural Network, see the LSTM section above. In short, Neural Networks (NN), even shallow ones, have the ability to learn from the underlying data structure. This allows them to capture complex patterns that simpler models like WA or LR might miss.

The process of getting the evaluation result from base models using NN involves:

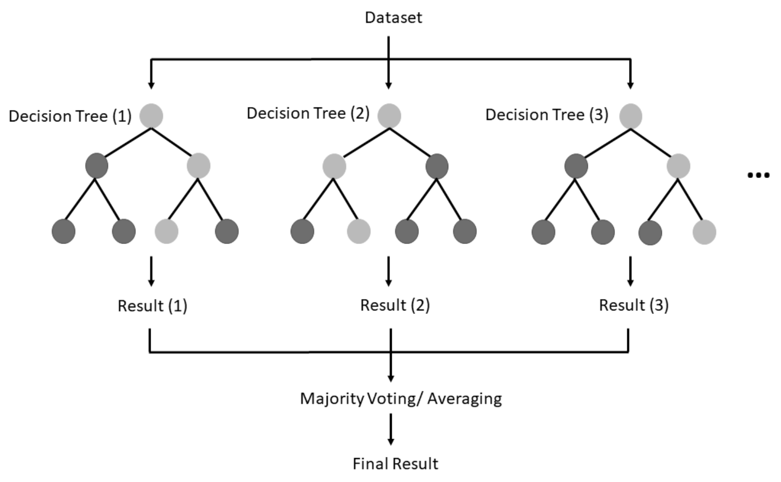
1. Train the base models on a subset of the total data (base model training data).
2. Calculate each base model's sMAPE (symmetric Mean Absolute Percentage Error) with the remainder of the data (meta-model training data).
3. Fit a shallow neural network with the base models' predictions as input features and the true target values as the output.
4. Once the NN model is fitted, train the base models on the full dataset so no training data is "lost" because of the training of the meta-model.

5. Calculate the final prediction by inputting the base models' predictions into the NN model.

Given its increased complexity compared to other models, we plan to implement this meta-model during Milestones 4-6 (M4-6), if necessary.

Random Forest (RF)

Random Forest (RF) is an ensemble machine learning algorithm that leverages the power of multiple decision trees for making predictions. It constructs a multitude of decision trees at training time and outputs the mode (for classification) or mean (for regression) of individual tree predictions. RF models are particularly well-suited to handle large datasets with high dimensionality, balancing prediction accuracy and model interpretability. In the context of our meta-modeling, RF is used to model the intricate relationships between the base models' predictions and the true target values, with each decision tree considering a subset of data and features to mitigate the risk of overfitting.



Random Forest

The final prediction Y from a Random Forest model is calculated as follows:

$$\hat{y} = \frac{1}{n}\sum_{i=1}^{n} \hat{y}_i$$

where:

- $\hat{y}$ is the final prediction
- n is the number of trees in the forest
- $\hat{y}_i$ is the prediction of the ith tree

The process of getting the evaluation result from base models using RF involves:

1. Train the base models on a subset of the total data (base model training data).
2. Calculate each base model's sMAPE (symmetric Mean Absolute Percentage Error) with the remainder of the data (meta-model training data).

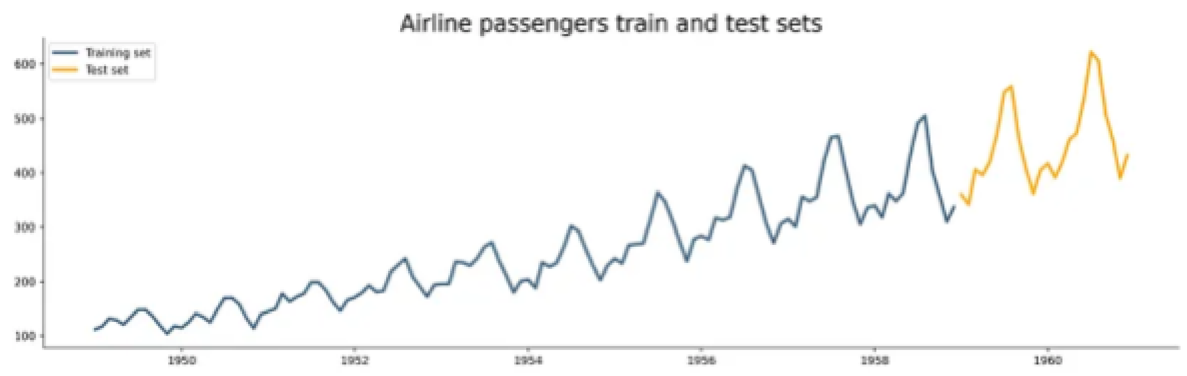
3. Fit a Random Forest model with the base models' predictions as input features and the true target values as the output.
4. Once the RF model is fitted, train the base models on the full dataset so no training data is “lost” because of the training of the meta-model.
5. Calculate the final prediction by inputting the base models' predictions into the RF model.

Given its robustness and ability to capture complex patterns, we plan to implement this meta-model sometimes during Milestones 4-7 (M4-7), if necessary.

5) Train Function - Evaluation

5a) Procedure

As described in the Preprocessing section, the input data must be properly split into at least train and test sets in sequential order. Then, the base models will fit the train set and be evaluated on the test set. See an example of a dataset split into train and test sets [16]:



5b) Metric Descriptions

Before detailing the evaluation metrics selected for SYBIL, here is an overview of the following four commonly used metrics:

1. Mean Absolute Error (MAE)
2. Mean Absolute Scaled Error (MASE)
3. Mean Absolute Percentage Error (MAPE)
4. Symmetric Mean Absolute Percentage Error (sMAPE)

For more details about these evaluation metrics and other available evaluations, see [3, 4].

Mean Absolute Error (MAE)

Absolute Error (AE) is the absolute difference (or residual) between the actual value y and the forecasted value $\hat{y}$, for a given future time t. Therefore, AE is always non-negative regardless of whether the forecast is underestimated or overestimated.

$$AE = |y - \hat{y}|$$

Mean Absolute Error (MAE) is the sum of the AEs for all forecasted times divided by the total number of forecasted times *n*.

$$MAE = \frac{1}{n}\sum_{t=1}^{n} |y_t - \hat{y}_t|$$

Mean Absolute Scaled Error (MASE)

This error is *scaled errors*-based. MASE is AE scaled by the in-sample MAE of a benchmark method, such as naïve or seasonal naïve method [5]. A naïve method uses the previous value y_{t-1} to forecast the current value y_t, and gets the error. Here is the scaling formula using the naïve method, which aggregates the errors of all of the naïve forecasts in its own MAE fashion:

$$s_t = \frac{1}{T-1}\sum_{t=2}^{T} |y_t - y_{t-1}|$$

Here is MASE, which is AE scaled by the above scaling formula at each given time *t*:

$$MASE = \frac{1}{n}\sum_{t=1}^{n} \frac{|yt - \hat{y}t|}{st}$$

Mean Absolute Percentage Error (MAPE)

The next two metrics are *percentage errors*-based, where the base error is scaled by the actual value *y* at the given time *t*. They will then be multiplied by 100 to be in percentage (%) unit.

MAPE first takes the sum of all absolute error (AE) divided by actual value *y* at the given time *t*, then divide by the number of time-series *N*, and finally multiply by 100.

$$MAPE = \frac{1}{N}\sum_{t=1}^{n} \frac{|yt - \hat{y}t|}{yt} \times 100\ (\%)$$

There are a few shortcomings regarding using MAPE:

1) MAPE is *not symmetric*, meaning exchanging the values of *y* with *ŷ* values changes the value of the error measure [3]. For example, if *y* = 150 and *ŷ* = 100, then the percentage error is 50/150 = 0.33 or 33%. However, if *y* = 100 and *ŷ* = 150, then the percentage error is 50/100 = 0.5 or 50%. Therefore, MAPE *"puts a heavier penalty on forecasts* [150] *that exceed the actual* [100] *than those* [100] *that are less than the actual* [150]*"* [6].
2) MAPE cannot work when an actual value *y* of 0 exists due to *y* being in the denominator. Therefore, MAPE cannot work with sparse datasets like during intermittent forecasting.
3) MAPE weighs the error measure at the given *t* unevenly. The error measure has more weight where the actual value y_t is smaller because that smaller value y_t scales it. Therefore, data points with low actual values y_t will disproportionately affect the final MAPE score, especially the closer to 0 [17].

4) Point forecasts that are optimized for MAPE target a non-conventional median from the underlying distribution: the median of an auxiliary variable whose density is proportional to the density of the original target variable divided by that variable itself [19]. This adds another difficulty to justifying using MAPE instead of metrics such as MSE or MAE. The latter metrics target the mean and median points from the underlying distribution. Deciding to target those points can be supported by the forecaster's desire to have unbiased forecasts, or be less sensitive to outliers. However, deciding to target the non-conventional median, as the MAPE does, has no clear reasons behind it.

Symmetric Mean Absolute Percentage Error (sMAPE)

sMAPE (also known as adjusted MAPE) is developed to resolve the *symmetry* shortcoming from MAPE, depending on how *symmetry* is defined. Rather than scaling using the actual value, sMAPE scales using the midpoint between the actual and the forecast, or (y_t + $\hat{y}_t$)/2. To work with negative numbers, the actual and forecasted numbers both become absolute numbers or (|y_t| + |$\hat{y}_t$)|/2. In the original sMAPE equation, the scaling coefficient is in the denominator, so the 2 is moved to the numerator:

$$sMAPE = \frac{1}{N}\sum_{i=1}^{n}\frac{2 \times |yt - \hat{y}t|}{|yt| + |\hat{y}t|} \; x\ 100\ (\%)$$

Compared to MAPE, sMAPE provides the following advantages:

1) sMAPE resolves the *symmetry* shortcoming if it means exchanging the values of y with ŷ values and preserving the same error measure.
2) sMAPE is more robust towards outlier data points and caps their contribution to the total error measure [18].

However, sMAPE is not perfect as it now penalizes the underestimates $\hat{y}_t$ (due to a smaller denominator) more than the overestimates $\hat{y}_t$ for the same value of y_t [3]. Therefore, there are multiple variations of sMAPE being proposed, including one that modifies the denominator to be *max*(|y_t| + |$\hat{y}_t$| + ϵ, 0.5 + ϵ), where the smoothing parameter ϵ is set to default of 0.1 [11]. This provides the dual benefits of preventing dividing by zero when y_t and $\hat{y}_t$ are 0s and stabilizing when y_t is near 0.

5c) Uses in SYBIL

We will start using the following metrics:

- **sMAPE**: the original method, although we may also modify the denominator as *max*(|y_t| + |$\hat{y}_t$| + ϵ, 0.5 + ϵ) [11] and [link].
- **MASE:** using the naïve method as the benchmark method for its scaling coefficient, although we may also use the seasonal naïve method.

We may further explore alternative metrics for future work. For example, Root Mean Square Scaled Error (RMSSE) for hierarchical forecasting.

6) Applications

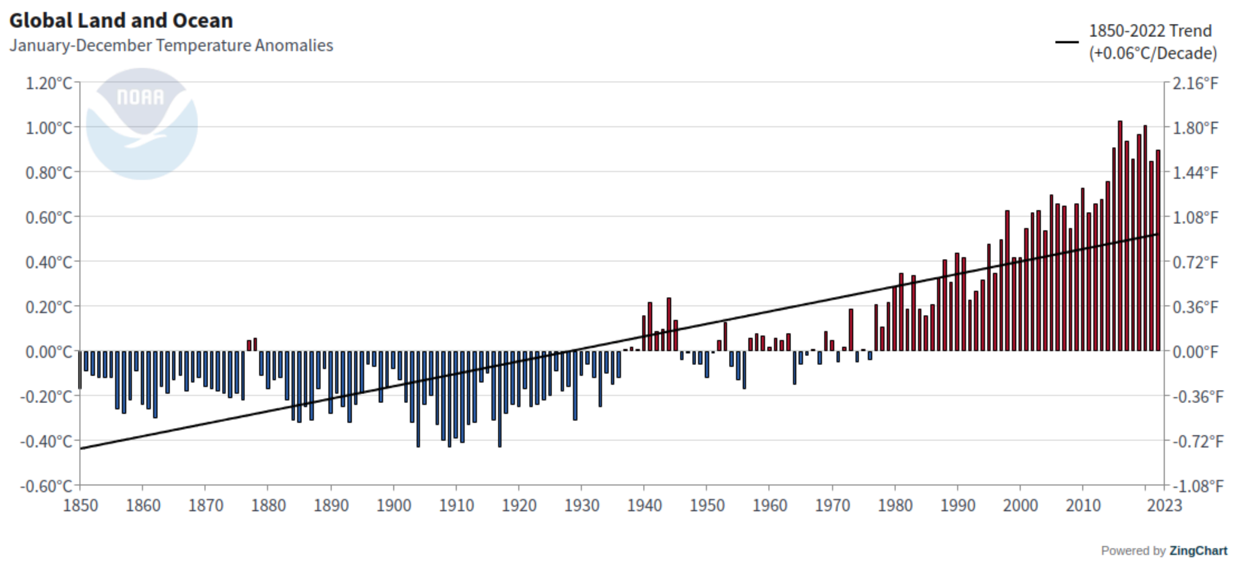
6a) Initial Use Cases

Since SYBIL's goal is to become a general-purpose forecaster, our starting use cases include a wide array of domains, which includes (but are not limited) the following:

1) **Climate:** long-term global land and ocean surface temperature forecasting
2) **Air quality:** AQ forecasting of GHGs (e.g., CO_2, PM2.5) or other noxious pollutants, measured by household IoT sensors or governmental environmental monitoring centers
3) **Energy:** electricity consumption forecasting from individual IoT appliances to buildings and entire grids
4) **Retail:** ranging from demand forecasting using Point-of-Sale transaction data to supply chain and inventory turnover forecasting
5) **DeFi:** decentralized finance (DeFi) applications, such as crypto price forecasting, market flows, yield farming, and blockchain network flows etc

Climate

We will be forecasting the global surface *temperature anomaly* data, which is the departure of temperature from a reference value or long-term average [link]. In other words, the higher the anomaly, the more the current temperature deviates from historical trends and averages. In recent years, the temperature anomalies have been positive, indicating the *positive acceleration* of global surface temperature for both land and ocean [link]:



The global temperature anomaly data come from US NOAA's Global Surface Temperature Analysis (NOAAGlobalTemp) and is available online, with downloaded data [link].

The second dataset pertains to *climate forcings*, which are global factors that affect Earth's land and ocean temperatures. There are eight forcings identified that can be classified as either *natural* or *anthropogenic* (human-caused):

Natural forcings:

- Solar Irradiance (*Solar*): amount of heat Earth receives from sun
- Stratospheric Aerosols (*StratAer*): particles that exist in Earth's stratosphere, usually from mixture of sulfuric acid (e.g., volcanic eruption) and water vapor
- Orbital Forcings (*Orbital*): effect from tilt of Earth's axis and orbit shape

Anthropogenic forcings:

- Well-Mixed Greenhouse Gases (*WMGHG*): effect from greenhouse gas emissions (i.e., CO_2, CH_4, and N_2O)
- Changes in Land Use (*Land_Use*) rate of deforestation for agricultural or industrial purposes
- Direct Tropospheric Aerosols (*TropAerDir*): absorbing sulfates, nitrates, sea salts, black carbon, etc. through solar and terrestrial radiation
- Indirect Tropospheric Aerosols (*TropAerInd*): absorbing sulfates, nitrates, sea salts, black carbon, etc. through cloud cloud formation
- Ozone (O_3): ozone layer depletion due to combination of sunlight and anthropogenic emissions [26]

Details of the climate forcings can be found on NASA GISS website [link] with downloadable data [link] of GISS CMIP5 simulations from years 1850-2012.

We plan to combine these two datasets, with the NOAA's *temperature anomaly* as the target time-series and GISS' *climate forcings* as the exogenous variables from 1850-2012 annually. That way, depending on the interpretability of SYBIL's base models, we can not only evaluate the *anomaly* forecast but also determine which *forcings* variables (or their lags) play the most important contributions to the final forecast. This combined dataset will not only be a good test of our model architecture but may also be a valuable global climate study.

This climate use case is advised by Matt I. from SingularityNET.

Air Quality

We will explore the following air quality datasets for development and testing purposes:

- UCI's Italian Device Air Quality: responses of a gas multisensor device deployed on an Italian field, including CO, NOx, and NO2. Data is recorded hourly from March 2004 to February 2005 [link].
- UCI's Beijing Multi-Site Air Quality: air pollutants data from 12 monitoring sites controlled by the Beijing Municipal Environmental Monitoring Center. The time period is from March 2013 - February 2017. AQ data includes PM2.5, PM10, SO2, NO2, CO, and O3 [link].

Energy

We will explore the following energy datasets for development and testing purposes:

- OpenEI's SF Hospital Load: recorded electricity consumption of a hospital in San Francisco in 2015 by the hour. It is available as a NeuralPropeht dataset [link].
- UCI's Appliances energy prediction: appliances energy use in a low-energy building in Belgium. Dataset includes house temperature and humidity conditions from a ZigBee wireless sensor network and weather data from a nearby airport weather station. The energy data was logged every 10 minutes for 4.5 months [link].

- UCI’s Individual household electric power consumption: Measurements of electric power consumption in one household near Paris, France. Measurements include electrical quantities and sub-metering values. It has a minute sampling rate between December 2006 - November 2010 (47 months) [link].

Retail

We will explore the following retail dataset for development and testing purposes:

- UCI’s Online Retail II: contains transactions of a UK-based online retailer between January 2009 - September 2011. The company sells giftware, and many customers are wholesalers [link].

In addition, we will be collaborating with a local farm-to-table Italian restaurant. The restaurant will provide us with point-of-sale data to forecast sales of certain items and dishes, to better manage its ingredients and prevent food waste.

DeFi

We seek to collaborate with projects in the DeFi and Web3 space to forecast use cases. Possible examples include forecasting certain cryptos' demand or transaction volume, their inflow and outflow from certain decentralized exchanges (DEXes), forecasting underlying blockchain transaction flows, and so forth.

As for data sources for development and testing purposes, we plan to explore Yahoo! Finance, CryptoQuant, Bitquery, and crypto exchange APIs.

6b) Best Practices

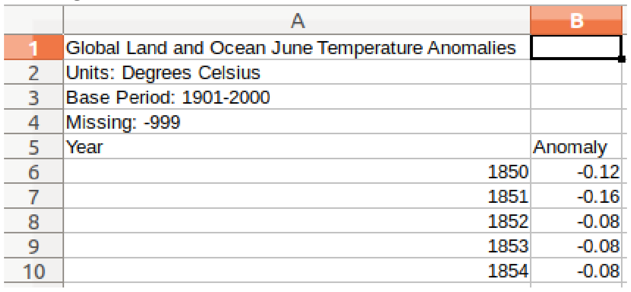
Save as CSV

Data files may come in various file types, including Excel, TXT, Parquet, and zipped files. This iteration of SYBIL will only accept CSV files with comma (,) delimitation.

Remove Non-Tabular Information

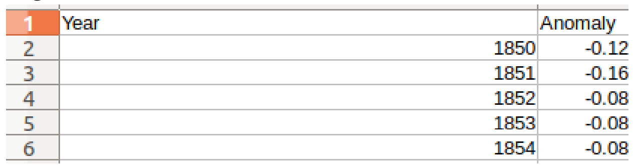
Remove information that is not associated with the data table. For example, in the NOAAGlobalTemp data (see Climate section), lines 1-4 contain data descriptions. Those need to be removed so the data starts with the header. Do not remove the header, as SYBIL will assume the first line is the header.

Wrong:



	A	B
1	Global Land and Ocean June Temperature Anomalies	
2	Units: Degrees Celsius	
3	Base Period: 1901-2000	
4	Missing: -999	
5	Year	Anomaly
6	1850	-0.12
7	1851	-0.16
8	1852	-0.08
9	1853	-0.08
10	1854	-0.08

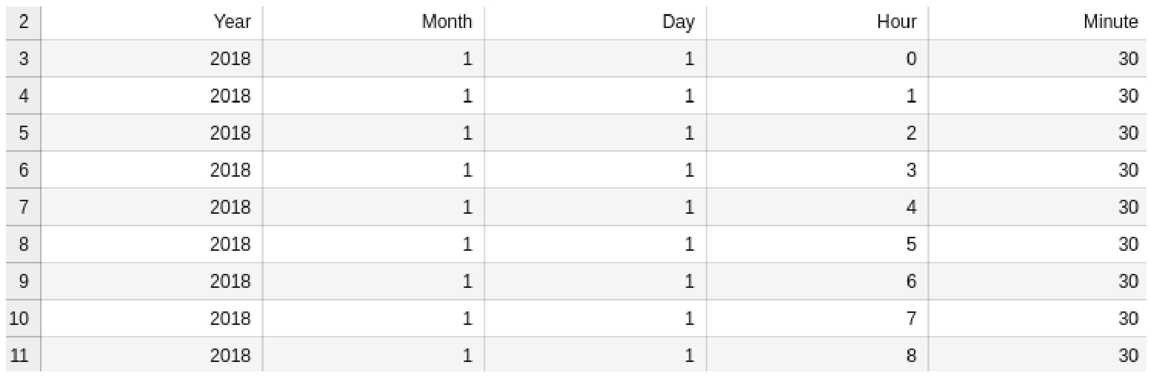
Right:



1	Year	Anomaly
2	1850	-0.12
3	1851	-0.16
4	1852	-0.08
5	1853	-0.08
6	1854	-0.08

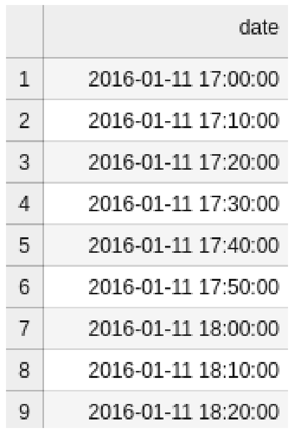
Ensure Singular Time Column

Some datasets may have their datetime spread across multiple columns. For example, Year, Month, Day, Hour, and Minute as shown below:



2	Year	Month	Day	Hour	Minute
3	2018	1	1	0	30
4	2018	1	1	1	30
5	2018	1	1	2	30
6	2018	1	1	3	30
7	2018	1	1	4	30
8	2018	1	1	5	30
9	2018	1	1	6	30
10	2018	1	1	7	30
11	2018	1	1	8	30

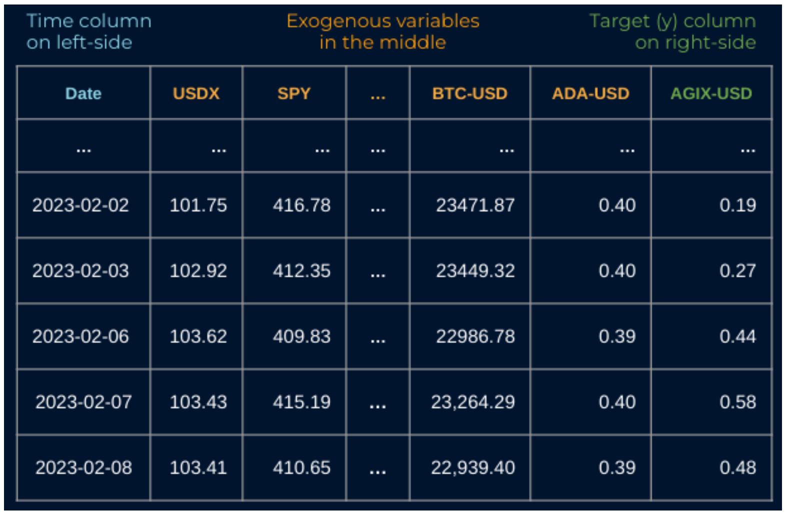
If that is the case, then combine these columns into one singular time column, in this case containing from year to minute:



	date
1	2016-01-11 17:00:00
2	2016-01-11 17:10:00
3	2016-01-11 17:20:00
4	2016-01-11 17:30:00
5	2016-01-11 17:40:00
6	2016-01-11 17:50:00
7	2016-01-11 18:00:00
8	2016-01-11 18:10:00
9	2016-01-11 18:20:00

Time on Left and Target on Right

Once there is a singular time column, then ensure that the time column is on the left most side the dataset, while the target column (*y*) is on the rightmost side. SIBYL will assume the data will be in this order, regardless of the column names. If there are any other time-series in the dataset, then leave them in the middle. SIBYL will treat these as exogenous variables and preprocess and model them accordingly. See the illustration below:


Time column on left-side | Exogenous variables in the middle | Target (y) column on right-side

Date	USDX	SPY	...	BTC-USD	ADA-USD	AGIX-USD
...	...	...	...	...	...	...
2023-02-02	101.75	416.78	...	23471.87	0.40	0.19
2023-02-03	102.92	412.35	...	23449.32	0.40	0.27
2023-02-06	103.62	409.83	...	22986.78	0.39	0.44
2023-02-07	103.43	415.19	...	23,264.29	0.40	0.58
2023-02-08	103.41	410.65	...	22,939.40	0.39	0.48

Once these four conditions are met, SYBIL should train on your dataset without a problem. There is no guarantee that its output model artifacts will produce good forecasting results that depend on the input dataset's values.

7) Future Works

7a) Features

Here are some of the possible features to include in SYBIL after MVP1:

- Specialized forecasting
  - Time-Series Classification
  - Anomaly Detection
  - Intermittent forecasting
  - Hierarchical forecasting
  - Long-term forecasting
  - Mixed-frequency forecasting
  - Global modeling
  - Cross-learning
- New functionalities
  - Probabilistic Outputs
    - Forecast distributions
    - Prediction intervals
- New metrics
  - Root Mean Squared Scaled Error (RMSSE)

7b) Models

Here some possible additional deep learning models to include in the base models:

- N-BEATs
- Temporal Convolutional Networks (TCN)

- Deep Autoregressive Models (e.g., DeepAR)
- Transformer-based
    - Informer
    - Temporal Fusion Transformer (TFT)
    - N-HiTS

7c) Product

Future version of SYBIL may be deployed on a standalone website. Additionally, it can be plug-and-playable with external client software or apps through a specified endpoint.

8) References

1) Hyndman, R., & Athanasopoulos, J. (2021). *Forecasting: Principles and Practice (3rd ed).* https://otexts.com/fpp3

2) Petropoulos, F., et al. (2022). *Forecasting: theory and practice.* https://www.sciencedirect.com/science/article/pii/S0169207021001758

3) Hewamalage, H., Ackermann, K., & Bergmeir, C. (2022). *Forecast Evaluation for Data Scientists: Common Pitfalls and Best Practices.* https://link.springer.com/article/10.1007/s10618-022-00894-5

4) Jadon, A., Patil, A., & Jadon, S. (2022). *A Comprehensive Survey of Regression Based Loss Functions for Time Series Forecasting.* https://arxiv.org/abs/2211.02989

5) Hyndman, R., & Koehler, A. (2014). *Another look at measures of forecast accuracy*. https://www.sciencedirect.com/science/article/pii/S0169207006000239

6) Hyndman, R. (2014). *Errors on percentage errors*. https://robjhyndman.com/hyndsight/smape

7) Wang, X., Hyndman, R., Li, & F., Kang, Y. (2022). *Forecast combinations: an over 50-year review.* https://arxiv.org/abs/2205.04216

8) Bojer, C. (2021). *Understanding machine learning-based forecasting methods: A decomposition framework and research opportunities*. https://www.sciencedirect.com/science/article/pii/S0169207021001771

9) Gastinger, J., Nicolas, S., Stepić, D., Schmidt, M., & Schülke, A. (2021). *A study on Ensemble Learning for Time Series Forecasting and the need for Meta-Learning*. https://arxiv.org/abs/2104.11475

10) Montero-Manso, P., Athanasopoulos, G., Hyndman, R., & Thiyanga, T. (2020). *FFORMA: Feature-based forecast model averaging*. https://robjhyndman.com/papers/fforma.pdf

11) Godahewa, R., Bergmeir, C., Webb, G., & Montero-Manso, P. (2023). *An accurate and fully-automated ensemble model for weekly time series forecasting.* https://www.sciencedirect.com/science/article/abs/pii/S0169207022000085

12) Rahman, M., Santu, S., Islam, M., & Murase, K. (2014). *Forecasting Time Series - A Layered Ensemble Architecture*.

https://www.researchgate.net/publication/281971102_Forecasting_Time_Series_-A_Layered_Ensemble_Architecture

13) Rahman, M., Islam, M., Murase, K., & Yao, X. (2015). *Layered Ensemble Architecture for Time Series Forecasting*. https://www.researchgate.net/publication/273787018_Layered_Ensemble_Architecture_for_Time_Series_Forecasting

14) Gabrys, B. (2010). *Meta-learning for time series forecasting and forecast combination*. https://www.academia.edu/2213498/Meta_learning_for_time_series_forecasting_and_forecast_combination?email_work_card=title

15) Hansen, J. (2002). *Data mining of time series using stacked generalizers*. https://www.researchgate.net/publication/222651510_Data_mining_of_time_series_using_stacked_generalizers

16) Radečić, D. (2021). *Time Series From Scratch — Train/Test Splits and Evaluation Metrics*. https://towardsdatascience.com/time-series-from-scratch-train-test-splits-and-evaluation-metrics-4fd654de1b37

17) MLBoost (2023). *(1) Model Evaluation - MAPE*. https://www.youtube.com/watch?v=eHYjr1Zhb_4&ab_channel=MLBoost

18) MLBoost (2023). *(2) Model Evaluation - adjustedMAPE*. https://www.youtube.com/watch?v=_uaJWg5eNpE&ab_channel=MLBoost

19) MLBoost (2023). *(7) Under Absolute Percentage Error loss, a Non-conventional Median is Optimal!*. https://www.youtube.com/watch?v=4APK8Dj3_24&ab_channel=MLBoost

20) Spiliotis, E., Assimakopoulos, V., & Makridakis, S. (2020). *Generalizing the Theta method for automatic forecasting*. https://www.researchgate.net/publication/338507620_Generalizing_the_Theta_method_for_automatic_forecasting

21) De Livera, A., Hyndman, R., & Snyder, R. (2010). *Forecasting time series with complex seasonal patterns using exponential smoothing*. https://robjhyndman.com/papers/ComplexSeasonality.pdf

22) Taylor, S., & Letham, B. (2017). *Forecasting at Scale*. https://peerj.com/preprints/3190.pdf

23) Triebe, O., Hewamalage, H., Pilyugina, P., Laptev, N., Bergmeir, C., & Rajagopal, R. (2021). *NeuralProphet: Explainable Forecasting at Scale*. https://arxiv.org/abs/2111.15397

24) Triebe, O., Laptev, N., Bergmeir, C., & Rajagopal, R. (2019). *AR-Net: A simple Auto-Regressive Neural Network for time-series*. https://arxiv.org/abs/1911.12436

25) Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, & Liu, T. (2017). *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*. https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

26) Goertzel, B., Znidar, N., Bayetta, M., Iklé, M., & Goertzel, S. (2021). *Causal and Predictive Analysis of Climate Change Using Granger Causality*. https://eartharxiv.org/repository/view/2372