**Abstract**

Hyperparameters are parameters that are specified prior to running machine learning algorithms that have a large effect on the predictive power of statistical models parameter of a prior distribution, the term which used to distinguish them from parameters of the model for the underlying system under analysis. Knowledge of the relative importance of a hyperparameter to an algorithm and its range of values is crucial to hyperparameter tuning and creating effective models.

The hyperparameter database allows users to visualize and understand how to choose hyperparameters that maximize the predictive power of their models. The hyperparameter database is created by running millions of hyperparameter values, calculating the individual conditional expectation of every hyperparameter on the quality of a model. The data science part need to generating models using H2O to find best hyperparameters.

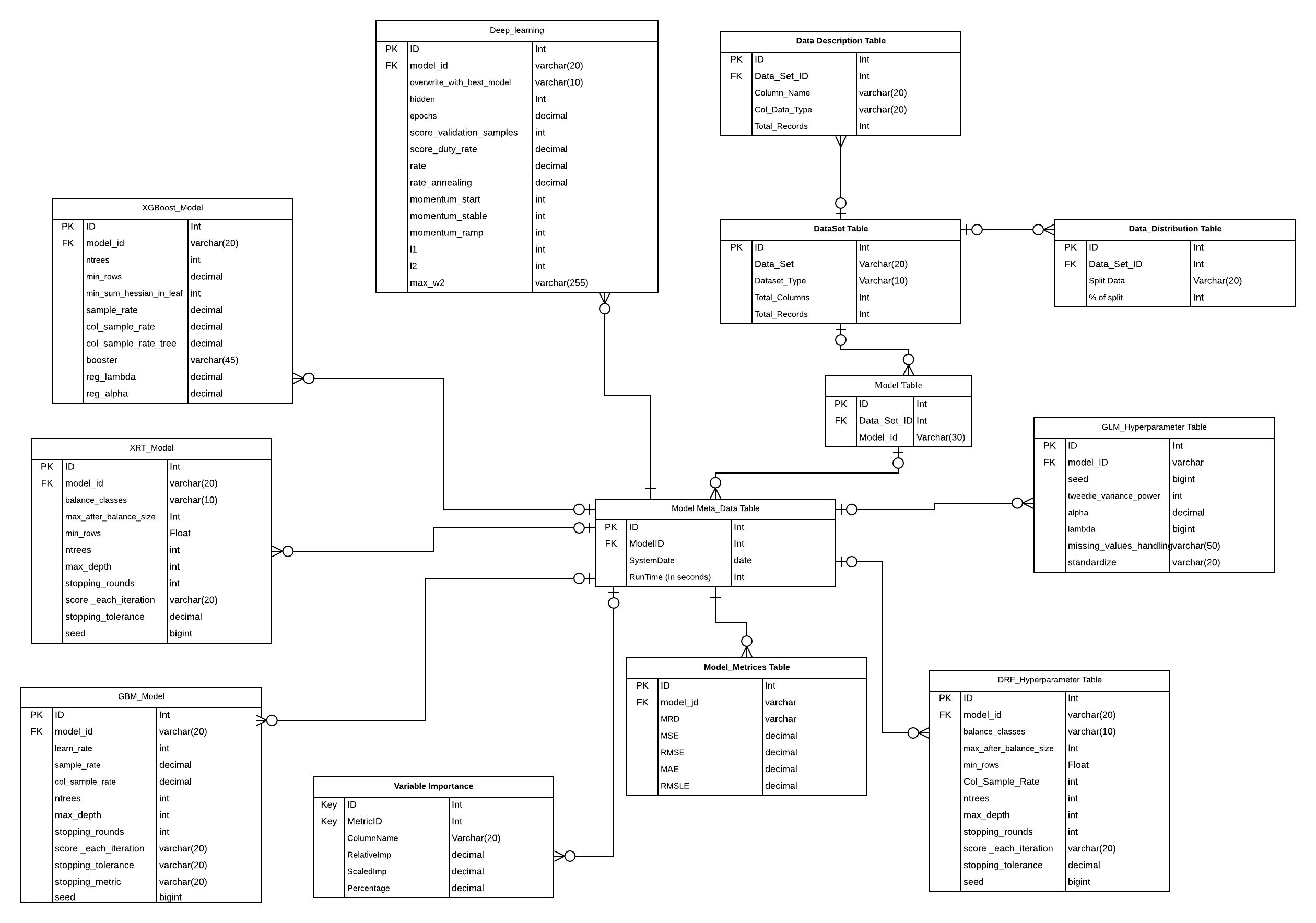
**Background**

The data we collected and stored concerns predicting housing transaction price which contains values of cities, floors, unit area households counts and parking capacity, rooms, heat fuel, heat type and front door structure. We separated and grouped data into different entities and attributes and build the one-to-many connections between them, which presented the data in more structured and organized way and allows us to query data, sort data, and manipulate data in various ways for the future performance.

**Dataset**

The dataset is from the website <https://www.kaggle.com/econdata/predciting-price-transaction#trainPrice.csv> . Housing price always been a popular item that people expect to predict. Since it is critical for us to find out the factors that affecting transaction price. This data set covers different aspects of factors which influence the housing price, which requires the scientific and specific method to calculate the best result.

**ERD**

****

**Normalization**

1NF

For all of our tables, We check them one by one and eliminate all the redundant data to ensure there are no repeating groups. We divided Alpha and lambda attributes in GLM Hyperparameter table into atomic as alpha one to seven and lambda one to five. And divided hiddens into hidden one to three in Deep Learning model. We make sure there are no same values in each table.

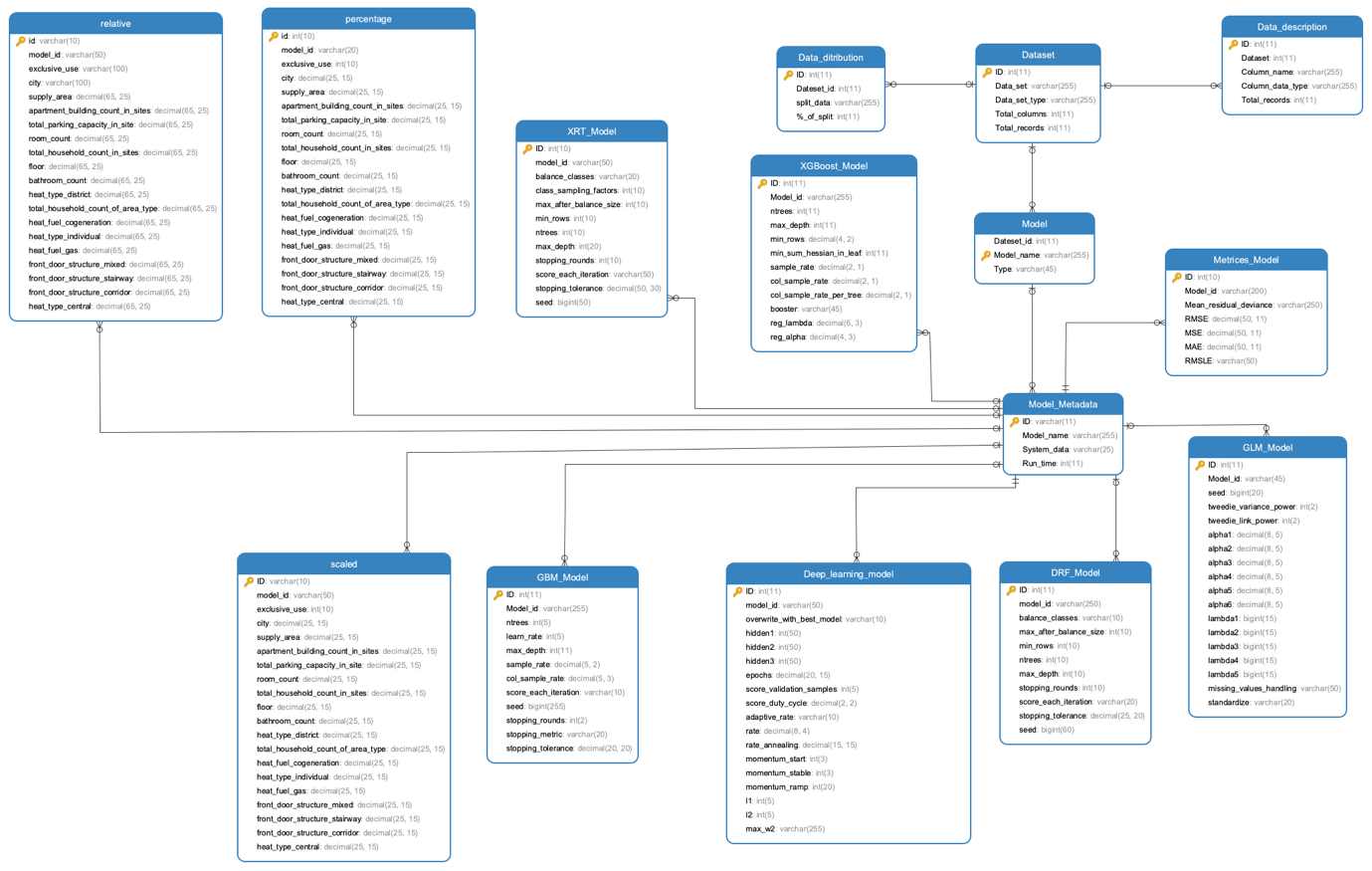
2NF

We check all the tables that whether there are any functional dependencies on part of any candidate key and make sure there are no partial dependencies.

3NF

We check all our tables and make sure there are no non-prime attribute is transitively dependent of any key. All the fields are directly depend on the primary key.

**Physical Model**

****

**Use Case**

1. Select the best model

SELECT me.Model\_id, me.RMSE, mm.Model\_name, mm.Run\_time

FROM Metrices\_Model me inner join Model\_Metadata mm

ON me.Model\_id = <mm.ID>

order by RMSE desc

LIMIT 1;

1. Select the hyperparameter with the same model but different runtime.

SELECT <mm.ID>, mm.Model\_name, mm.Run\_time,

me.RMSE,

dl. hidden1, hidden2,hidden3,epochs

FROM Model\_Metadata mm left join Deep\_learning\_model dl

ON <mm.ID> = dl.model\_id

JOIN Metrices\_Model me

ON dl.model\_id = me.Model\_id

WHERE <mm.ID> LIKE "DL%"

ORDER BY mm.Model\_name, me.RMSE, mm.Run\_time desc；

1. Select the average rmse with the same type model

SELECT avg(mm.RMSE) as GLM\_AVERAGE\_rmse

FROM Metrices\_Model mm

WHERE mm.Model\_id LIKE "G%"

1. Select ID, name, runtime and rmse which is higher than the average rmse with the same “XG” ID

SELECT <metr.ID>, metr.Model\_id,mm.Model\_name, mm.Run\_time, metr.RMSE

FROM Metrices\_Model metr INNER JOIN Model\_Metadata mm

ON metr.Model\_id = <mm.ID>

WHERE metr.Model\_id LIKE "XG%"

HAVING metr.RMSE > (

SELECT avg(metr.RMSE)

FROM Metrices\_Model metr

WHERE metr.Model\_id LIKE "XG%"

)

ORDER BY metr.RMSE DESC;

1. Select the counts of models which runtime is 2000

SELECT COUNT(\*)

FROM Model\_Metadata mm

WHERE mm.Run\_time = 2000

order BY mm.Run\_time

;

1. Select the top 10 rmse in XRT model

SELECT <metr.ID>, metr.Model\_id,mm.Model\_name, mm.Run\_time, metr.RMSE

FROM Metrices\_Model metr INNER JOIN Model\_Metadata mm

ON metr.Model\_id = <mm.ID>

WHERE metr.Model\_id LIKE "XRT%"

ORDER BY metr.RMSE

LIMIT 10;

1. Select the range of the learning rate of all the model

SELECT

MIN(gm.ntrees) AS min\_ntrees,

MAX(gm.ntrees) AS max\_ntrees,

MIN(gm.max\_depth) AS min\_max\_depth,

MAX(gm.max\_depth) AS max\_max\_depth

FROM GBM\_Model gm;

1. Select all cities which variable is higher than 0.08

SELECT \*

FROM percentage pp

having pp.city > 0.08

order by pp.city desc;

1. Select all the runtime from the dataset

SELECT distinct mm.Run\_time

from Model\_Metadata mm

order by mm.Run\_time;

1. Select the type, counts of the type and the best RMSE by order from model data tables

SELECT mo.TYPE AS MODEL\_TYPE, COUNT(\*) AS AMOUNT, MAX(me.RMSE) AS BEST\_RMSE

FROM Model\_Metadata mm, Model mo, Metrices\_Model me

WHERE mm.Model\_name = mo.Model\_name AND <mm.ID> = me.Model\_id

GROUP BY mo.type

ORDER BY max(me.RMSE) desc ;

**Views**

1. **Select all the counts of deep learning models in model**

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `finall`.`case1` AS

SELECT

COUNT(0) AS `COUNT(\*)`

FROM

`finall`.`model` `m`

WHERE

(`m`.`Type` = 'DL')

1. **Select GLM ID, model name and standardize from metadata by runtime** order

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `finall`.`case2` AS

SELECT

`mm`.`ID` AS `ID`,

`mm`.`Model\_name` AS `Model\_name`,

`dl`.`standardize` AS `standardize`

FROM

(`finall`.`model\_metadata` `mm`

JOIN `finall`.`glm\_model` `dl` ON ((`mm`.`ID` = `dl`.`Model\_id`)))

ORDER BY `mm`.`Model\_name` , `mm`.`Run\_time` DESC

1. **Select top 20 DL MSE and runtime in metrices model**

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `finall`.`case3` AS

SELECT

`metr`.`ID` AS `ID`,

`metr`.`Model\_id` AS `Model\_id`,

`mm`.`Model\_name` AS `Model\_name`,

`mm`.`Run\_time` AS `Run\_time`,

`metr`.`MSE` AS `MSE`

FROM

(`finall`.`metrices\_model` `metr`

JOIN `finall`.`model\_metadata` `mm` ON ((`metr`.`Model\_id` = `mm`.`ID`)))

WHERE

(`metr`.`Model\_id` LIKE 'DL%')

ORDER BY `metr`.`MSE`

LIMIT 20

1. **Select average RMSE in DRF model**

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `finall`.`case4` AS

SELECT

AVG(`mm`.`RMSE`) AS `GLM\_AVERAGE\_rmse`

FROM

`finall`.`metrices\_model` `mm`

WHERE

(`mm`.`Model\_id` LIKE 'DRF%')

**Functions**

1. **Get\_Average\_Stopping\_tolerance\_in\_XRT\_Model`**

CREATE DEFINER=`root`@`localhost` FUNCTION `get\_Average\_Stopping\_tolerance\_in\_XRT\_Model`(id int) RETURNS varchar(500) CHARSET utf8

BEGIN

DECLARE a varchar(500);

-- DECLARE b BIGINT;

SELECT AVG(stopping\_tolerance) INTO a FROM XRT\_Model

WHERE ID = id;

RETURN (a);

END

1. **get\_Deep\_learning\_model\_Epochs**

CREATE DEFINER=`root`@`localhost` FUNCTION `get\_Deep\_learning\_model\_Epochs`(id int) RETURNS varchar(500) CHARSET utf8

BEGIN

DECLARE a varchar(500);

-- DECLARE b BIGINT;

SELECT epochs INTO a FROM Deep\_learning\_model

WHERE ID = id limit 1;

RETURN (a);

END

1. **Get Max\_depth\_Bigger\_Than\_Enter\_In\_XGBoost\_Model**

CREATE DEFINER=`root`@`localhost` FUNCTION `Max\_depth\_Bigger\_Than\_Enter\_In\_XGBoost\_Model`(EnteredNum int) RETURNS varchar(50) CHARSET utf8

BEGIN

DECLARE b BIGINT;

SELECT count(max\_depth) INTO b FROM XGBoost\_Model

WHERE max\_depth > EnteredNum;

RETURN (b);

END

1. **Type\_Max\_Get\_Max\_MAE\_In\_Metrices\_Model**

CREATE DEFINER=`root`@`localhost` FUNCTION `Type\_Max\_Get\_Max\_MAE\_In\_Metrices\_Model`(Enter VARCHAR(50)) RETURNS varchar(500) CHARSET utf8

BEGIN

DECLARE namemodel VARCHAR(500);

SELECT max(MAE) INTO namemodel FROM Metrices\_Model

where maxx = "max";

RETURN namemodel;

END

**Analytics ＆ Conclusions**

By storing Hyperparameters data set in the database enables us obtain the structural and organized data to call the different functions for analyzing and select the best model for prediction, which make it more visualized to check and use the data and achieve different utilization. By creating the use cases, functions and views, we can select single or combined date set, get the best model, calculate the average or the max data for improving the different performance.

# Citations

<https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/>

<https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning>)

<https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a>

<https://towardsdatascience.com/hyperparameters-in-deep-learning-927f7b2084dd>

<https://www.w3schools.com/sql/sql_create_index.asp>

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-2017>

<https://www.w3schools.com/sql/sql_view.asp>

# License

MIT License

Copyright (c) 2019 Hsiang-Hua Chen, Yuan Chai

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.