

# HYPERPARAMETER DATABASE PROJECT

## GROUP DB03

### 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. 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 is a public resource with algorithms, tools, and data that 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, over thousands of public datasets and calculating the individual conditional expectation of every hyperparameter on the quality of a model.

Currently, the hyperparameter database analyzes the effect of hyperparameters on the following algorithms: Distributed Random Forest (DRF), Generalized Linear Model (GLM), Gradient Boosting Machine (GBM). Naïve Bayes Classifier, Stacked Ensembles, Xgboost and Deep Learning Models (Neural Networks).

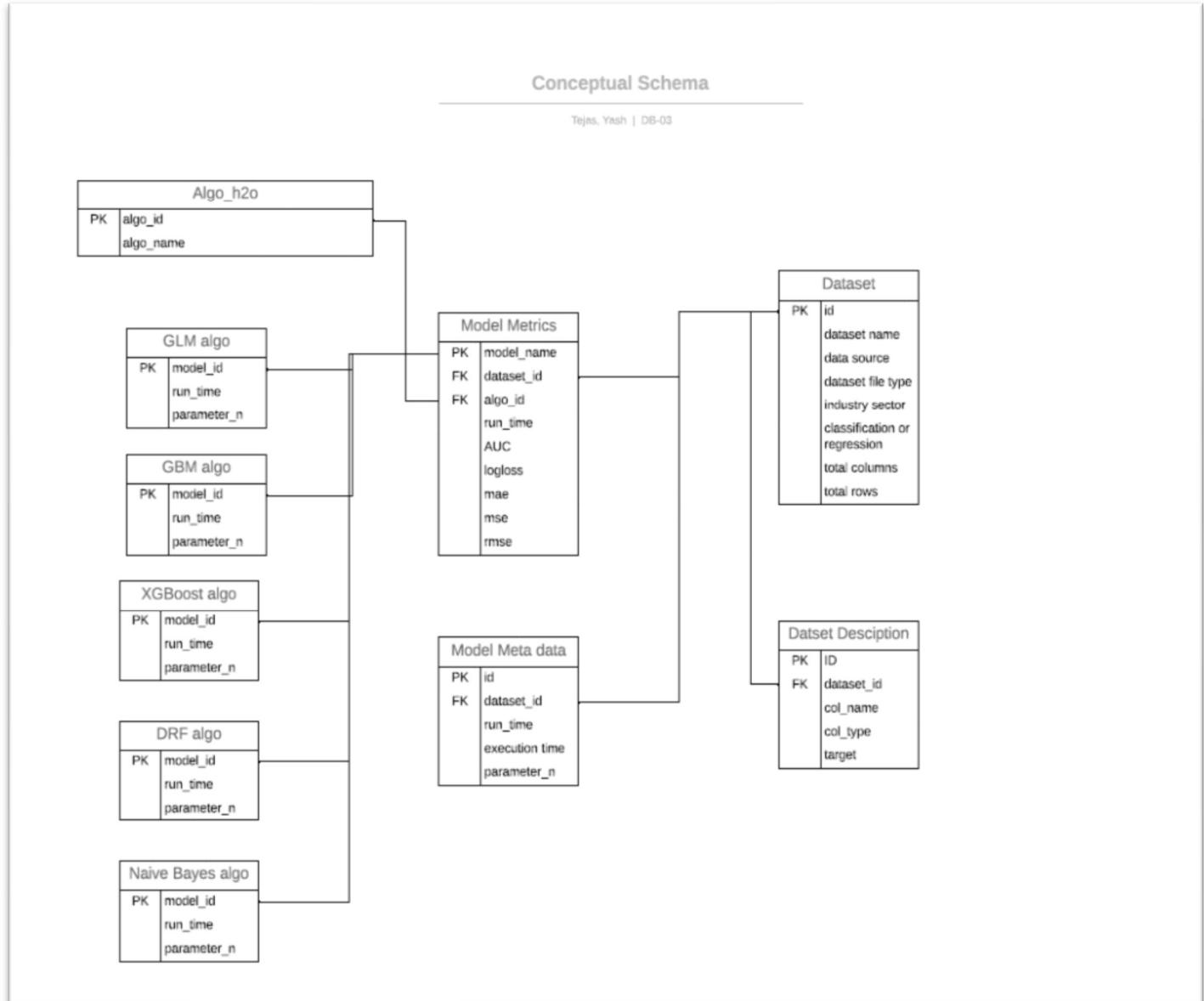
The hyperparameter database also uses these data to build models that can predict hyperparameters without search and for visualizing and teaching statistical concepts such as power and bias/variance tradeoff.

As a part of the DMDD team we created a conceptual model and stored all the data into a physical database by going through the following procedures

- Scraped JSON files and stores the data in dataframes further saving them in CSV Format.
- Created a conceptual model of the database.
- Perform database normalization – 1NF, 2NF, 3NF.
- Created tables as per the requirement.
- Inserted data into the respective tables from the file which is generated and stored by the Data Science member.
- Queried database using SQL, created Views (virtual tables) and Functions for various use cases of the database.
- Performed analytics on the database using queries e.g. getting the best value for the hyperparameter from the database.

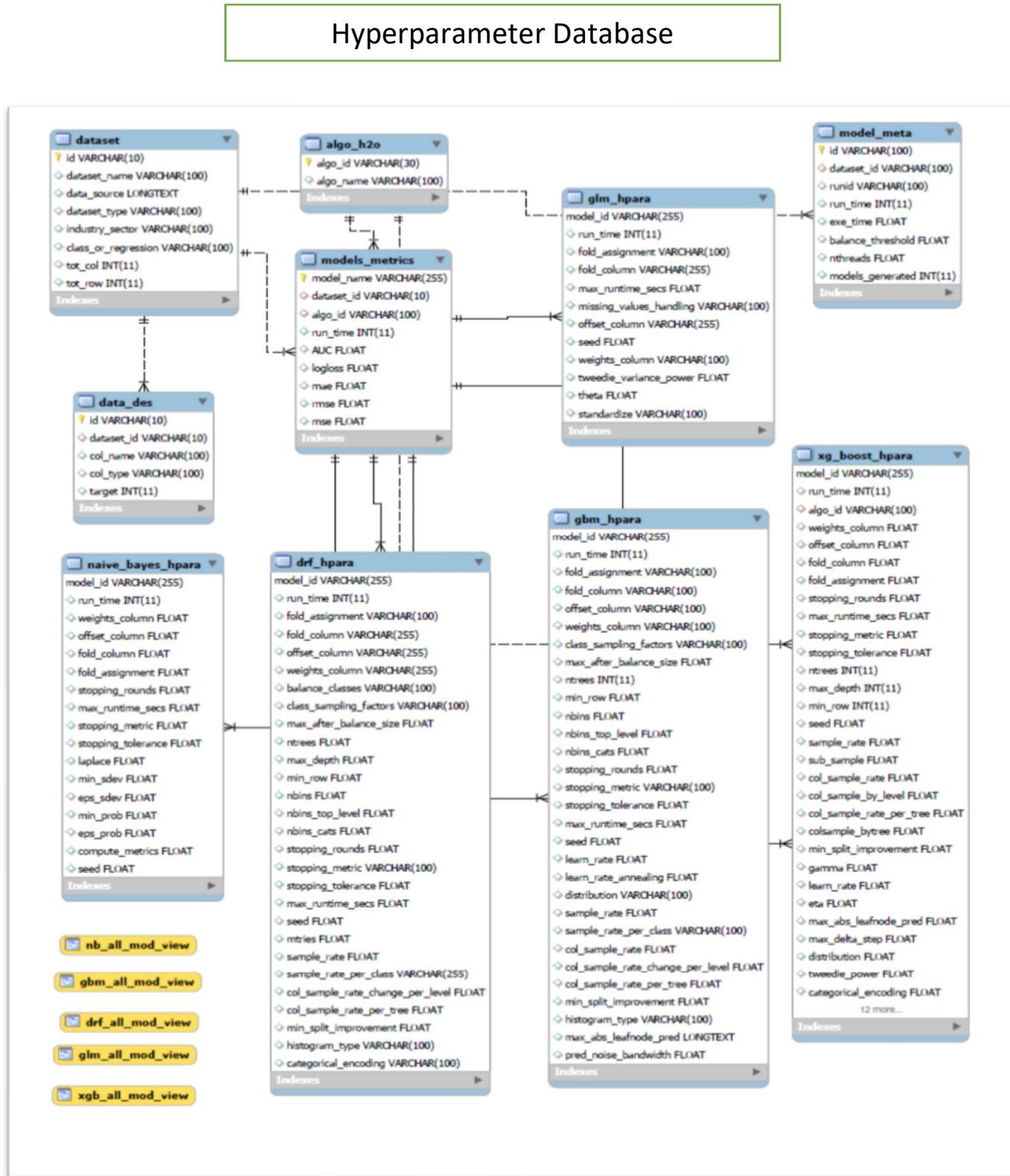
## CONCEPTUAL DIAGRAM :

A conceptual data model captures the key entities (a person, place, concept, event or thing about which the organization wants to collect data), and the relationships between these entities. Attributes are often not added to conceptual data models as these are higher level models.



## ENTITY RELATIONSHIP DIAGRAM:

An **entity relationship** model, also called an **entity-relationship (ER) diagram**, is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within **databases** or information systems



# JSON TO CSV

- Scraped the JSON files to create a dataframe of all the Hyperparameters for respective algorithm.  
Imported JSON files of all the models.

The screenshot shows a Jupyter Notebook interface with the title "jupyter Final Everything" and the status "Last Checkpoint: Yesterday at 7:28 PM (autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Python 3 logo. Below the toolbar is a toolbar with icons for file operations like Open, Save, Run, and Kernel. The main area has a section titled "IMPORTING LIBRARIES" containing the code:

```
In [6]: import pandas as pd  
import os  
import glob
```

Below this is a section titled "ASSIGNING PATHS" containing the code:

```
In [7]: path_300 = r"C:\Users\Owner\Desktop\Hyperparameter CSV\Final Json\hyperparameter-db-project-ds03-master\nRidYg95h 300*.json"  
path_500 = r"C:\Users\Owner\Desktop\Hyperparameter CSV\Final Json\hyperparameter-db-project-ds03-master\xBB2YBVDT 500*.json"  
path_800 = r"C:\Users\Owner\Desktop\Hyperparameter CSV\Final Json\hyperparameter-db-project-ds03-master\douhJzEiy 800*.json"  
path_1000 = r"C:\Users\Owner\Desktop\Hyperparameter CSV\Final Json\hyperparameter-db-project-ds03-master\uh5qmPqcn 1000*.json"  
path_1200 = r"C:\Users\Owner\Desktop\Hyperparameter CSV\Final Json\hyperparameter-db-project-ds03-master\9Zo6W4yCU 1200*.json"  
  
In [8]: files_300 = glob.glob(path_300)  
files_500 = glob.glob(path_500)  
files_800 = glob.glob(path_800)  
files_1000 = glob.glob(path_1000)  
files_1200 = glob.glob(path_1200)
```

Created Dictionaries for each algorithms consisting of only the Hyperparameters defined in the documentation of the H2o AutoML

The screenshot shows a Jupyter Notebook interface with the title "Creating Dictionary". The code cell contains the following dictionary definition:

```
In [9]: dict=[{"DRF": [ "fold_assignment", "fold_column", "offset_column", "weights_column",  
"balance_classes", "class_sampling_factors", "max_after_balance_size", "ntrees",  
"max_depth", "min_rows", "nbins", "nbins_top_level", "nbins_cats", "stopping_rounds",  
"stopping_metric", "stopping_tolerance", "max_runtime_secs", "seed", "mtries",  
"sample_rate", "sample_rate_per_class", "col_sample_rate_change_per_level", "col_sample_rate_per_tree",  
"min_split_improvement", "histogram_type", "categorical_encoding", "model_id"],  
"GBM": [ "fold_assignment", "fold_column", "offset_column", "weights_column", "class_sampling_factors",  
"max_after_balance_size", "ntrees", "min_rows", "nbins", "nbins_top_level", "nbins_cats", "stopping_rounds",  
"stopping_metric", "stopping_tolerance", "max_runtime_secs", "seed", "learn_rate", "learn_rate_annealing",  
"distribution", "sample_rate", "sample_rate_per_class", "col_sample_rate", "col_sample_rate_change_per_level",  
"col_sample_rate_per_tree", "min_split_improvement", "histogram_type", "max_abs_leafnode_pred", "pred_noise_bandwid",  
"GLM": [ "seed", "fold_assignment", "fold_column", "offset_column", "weights_column", "tweedie_variance_power",  
"theta", "alpha", "lambda", "standardize", "missing_values_handling", "max_runtime_secs"],  
"XGBoost": [ 'weights_column', 'offset_column', 'fold_column', 'fold_assignment',  
'stopping_rounds', 'max_runtime_secs', 'stopping_metric', 'stopping_tolerance',  
'ntrees', 'max_depth', 'min_row', 'seed', 'sample_rate', 'sub_sample',  
'col_sample_rate', 'col_sample_by_level', 'col_sample_rate_per_tree',  
'colsample_bytree', 'min_split_improvement', 'gamma', 'learn_rate', 'eta',  
'max_abs_leafnode_pred', 'max_delta_step', 'distribution', 'tweedie_power', 'categorical_encoding',  
'tree_method', 'num_leaves', 'min_sum_hessian_in_leaf', 'min_data_in_leaf', 'grow_policy', 'booster',  
'reg_lambda', 'sample_type', 'normalize_type', 'rate_drop', 'one_drop', 'skip_drop' ]}]
```

## Scraping all the JSON files for each model of Runtime 500

```
300
hp_XGBoosts=pd.DataFrame(columns=dict["XGBoost"])
hp_GLMs=pd.DataFrame(columns=dict["GLM"])
hp_GBMs=pd.DataFrame(columns=dict["GBM"])
hp_DRFs=pd.DataFrame(columns=dict["DRF"])
temp_name_xgb=[]
temp_name_glm=[]
temp_name_gbm=[]
temp_name_drf=[]

for f in files_500:
    if "XGBoost" in f:
        data1=pd.read_json(f)
        aa=data1[["model_id"]][['actual']]['name']
        temp_name_xgb.append(aa)
        data1=data1.transpose()
        data1[["model_id"]]=data1.actual["model_id"]["name"]
        data1[["training_frame"]]=data1.actual["training_frame"]["name"]
        data2=data1[dict["XGBoost"]].copy()
        data3=data2.drop("default")
        hp_XGBoosts=pd.concat([hp_XGBoosts,data3],join="inner")
        hp_XGBoosts["runtime"]=500
        hp_XGBoosts["name"]=temp_name_xgb
    elif "GLM" in f:
        data1=pd.read_json(f)
        aa=data1[["model_id"]][['actual']]['name']
        temp_name_glm.append(aa)
        data1=data1.transpose()
        data1[["model_id"]]=data1.actual["model_id"]["name"]
        data1[["training_frame"]]=data1.actual["training_frame"]["name"]
        data2=data1[dict["GLM"]].copy()
        data3=data2.drop("default")
        hp_GLMs=pd.concat([hp_GLMs,data3],join="inner")
        hp_GLMs["runtime"]=500
        hp_GLMs["name"]=temp_name_glm
    elif "GBM" in f:
        data1=pd.read_json(f)
        aa=data1[["model_id"]][['actual']]['name']
        temp_name_gbm.append(aa)
        data1=data1.transpose()
        data1[["model_id"]]=data1.actual["model_id"]["name"]
        data1[["training_frame"]]=data1.actual["training_frame"]["name"]
        data2=data1[dict["GBM"]].copy()
        data3=data2.drop("default")
        hp_GBMs=pd.concat([hp_GBMs,data3],join="inner")
        hp_GBMs["runtime"]=500
        hp_GBMs["name"]=temp_name_gbm
    elif "DRF" in f:
        data1=pd.read_json(f)
        aa=data1[["model_id"]][['actual']]['name']
        temp_name_drf.append(aa)
        data1=data1.transpose()
        data1[["model_id"]]=data1.actual["model_id"]["name"]
        data1[["training_frame"]]=data1.actual["training_frame"]["name"]
        data2=data1[dict["DRF"]].copy()
        data3=data2.drop("default")
        hp_DRFs=pd.concat([hp_DRFs,data3],join="inner")
        hp_DRFs["runtime"]=500
        hp_DRFs["name"]=temp_name_drf
```

- Repeated the same process and scraped the data from JSON of different runtimes.
- Further appending them into a single dataframe for each algorithm.

## **NORMALIZATION**

### **check that tables are in First normal form (1NF)**

- Each table has a primary key: minimal set of attributes which can uniquely identify a record
  - Generated Primary keys for the tables which didn't have one.
- The values in each column of a table are atomic (No multi-value attributes allowed)
  - Cleaned the data so there were no atomic values in every table
- There are no repeating groups: two columns do not store similar information in the same table
  - Drop the tables which from the dataset which showed redundancy

### **A check that tables are in Second normal form (2NF)**

- All requirements for 1st NF must be met.
  - Satisfied
- No partial dependencies.
  - While tidying the datasets cleaned the partial dependencies
- No calculated data
  - The variables depended upon other variables were removed

### **A check that tables are in Third normal form (3NF) and the final SQL**

- All requirements for 2nd NF must be met.
  - Satisfied
- Eliminate fields that do not directly depend on the primary key; that is no transitive dependencies.
  - Satisfied.

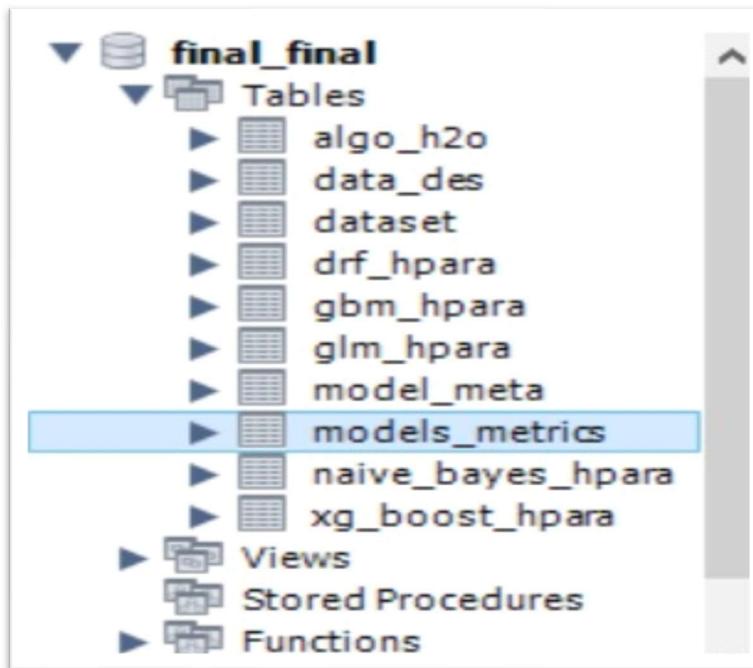
## CREATING TABLES IN SQL

### Creating Tables

```
FINAL FINAL Creating Database x
CREATE DATABASE IF NOT EXISTS FINAL_FINAL;
USE FINAL_FINAL;
CREATE TABLE dataset
(
    id VARCHAR(10),
    dataset_name VARCHAR(100),
    data_source LONGTEXT,
    dataset_type VARCHAR(100),
    industry_sector VARCHAR(100),
    class_or_regression VARCHAR(100),
    tot_col INT,
    tot_row INT,
    PRIMARY KEY(id)
);
```

As per our final ERD we created the tables and assigned the primary keys and foreign keys respective to each table.

### Final Tables



## VIEWS:

**View** can be described as virtual table which derived its data from one or more than one table columns. It is stored in the database. **View** can be created using tables of same database or different database. It is **used to** implement the security mechanism in the **SQL Server**

### VIEW 1

GBM VIEW

```
final_final_VIEWS x
1 • USE final_final;
2
3 -- View all GBM models
4 • CREATE VIEW gbm_all_mod_view AS
5   SELECT gbm_hpara.* , models_metrics.rmse FROM
6     models_metrics JOIN gbm_hpara
7   ON
8     models_metrics.model_name = gbm_hpara.model_id
9   WHERE models_metrics.algo_id = 'A2'
10  ORDER BY gbm_hpara.run_time,models_metrics.rmse ;
11
```

GBM VIEW OUTPUT

```
final_final_VIEWS x | gbm_all_mod_view
1 • USE final_final;
2
3 -- View all GBM models
4 • CREATE VIEW gbm_all_mod_view AS
5   SELECT gbm_hpara.* , models_metrics.rmse FROM
6     models_metrics JOIN gbm_hpara
7   ON
8     models_metrics.model_name = gbm_hpara.model_id
9   WHERE models_metrics.algo_id = 'A2'
10  ORDER BY gbm_hpara.run_time,models_metrics.rmse ;
11
12 -- View all Xgboost models
13 • CREATE VIEW xgb_all_mod_view AS
14   SELECT xg_boost_hpara.* , models_metrics.rmse FROM
15     models_metrics JOIN xg_boost_hpara
16   ON
17     models_metrics.model_name = xg_boost_hpara.model_id
18   WHERE models_metrics.algo_id = 'A1'
19   ORDER BY xg_boost_hpara.run_time,models_metrics.rmse ;
20
21 -- View all Naive Bayes models
22 • CREATE VIEW nb_all_mod_view AS
23   SELECT naive_bayes_hpara.* , models_metrics.rmse FROM
24     models_metrics JOIN naive_bayes_hpara
25   ON
```

Output

#	Time	Action	Message	Duration / Fetch
25	15:38:18	SELECT DISTINCT (col_sample_rate), COUNT(model_id) FROM gbm_hpara GROUP BY col_sample_rate	4 row(s) returned	0.000 sec / 0.000 sec
26	15:38:52	SELECT dataset.dataset_name, dataset.id, COUNT(model_meta.run_time) FROM dataset JOIN model_meta ON dataset.id = model_meta.dataset_id WHERE dataset.dataset_name = 'A2' AND model_meta.algo_id = 'A2' GROUP BY dataset.dataset_name, dataset.id	1 row(s) returned	0.000 sec / 0.000 sec
27	15:39:35	SELECT * FROM final_final.dfr_all_mod_view LIMIT 0, 1000	8 row(s) returned	0.000 sec / 0.000 sec
28	15:40:21	SELECT * FROM final_final.gbm_all_mod_view LIMIT 0, 1000	134 row(s) returned	0.000 sec / 0.000 sec

## VIEW 2:

### XGBOOST VIEW

```
-- View all Xgboost models
CREATE VIEW xgb_all_mod_view AS
SELECT xg_boost_hpara.* , models_metrics.rmse FROM
models_metrics JOIN xg_boost_hpara
ON
models_metrics.model_name = xg_boost_hpara.model_id
WHERE models_metrics.algo_id = 'A1'
ORDER BY xg_boost_hpara.run_time,models_metrics.rmse ;
```

### XGBOOST VIEW OUTPUT

The screenshot shows a database interface with two main sections. The top section is titled "XGBOOST VIEW" and contains the SQL code for creating a view named "xgb\_all\_mod\_view". The bottom section is titled "XGBOOST VIEW OUTPUT" and shows the results of executing the view. The output includes a table with columns: model\_id, run\_time, algo\_id, weights\_column, offset\_column, fold\_column, fold\_assignment, stopping\_rounds, max\_runtime\_secs, stopping\_metric, stopping\_tolerance, ntrees. Below the table, there is an "Action Output" section showing the history of actions taken, including the execution of the view query.

#	Time	Action	Message	Duration / Fetch
28	15:40:21	SELECT * FROM final_final.gbm_all_mod_view LIMIT 0, 1000	134 row(s) returned	0.000 sec / 0.000 sec
29	15:40:40	SELECT * FROM final_final.glm_all_mod_view LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec
30	15:41:29	SELECT * FROM final_final.nb_all_mod_view LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
31	15:41:59	SELECT * FROM final_final.xgb_all_mod_view LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

## VIEW 3

### GLM VIEW INPUT

```
-- View all GLM models
CREATE VIEW glm_all_mod_view AS
SELECT glm_hpara.* , models_metrics.rmse FROM
models_metrics JOIN glm_hpara
ON
models_metrics.model_name = glm_hpara.model_id
WHERE models_metrics.algo_id = 'A1'
ORDER BY glm_hpara.run_time,models_metrics.rmse ;
```

### GLM VIEW OUTPUT

The screenshot shows a database interface with the following details:

- Toolbar:** Includes icons for file operations, search, and various database functions.
- Query Bar:** Shows the query `SELECT * FROM final_final(glm_all_mod_view);`.
- Result Grid:** Displays the output of the query, showing four rows of data:

model_id	run_time	fold_assignment	fold_column	max_runtime_secs	missing_values_handling	offset_column	seed	weights_column
GLM_grid_1_AutoML_20190420_203958_model_1	300	Modulo	0	0	MeanImputation	-4.60175e18		
GLM_grid_1_AutoML_20190420_204019_model_1	300	Modulo	0	0	MeanImputation	3.10247e17		
GLM_grid_1_AutoML_20190420_204025_model_1	300	Modulo	0	0	MeanImputation	2.38493e18		
GLM_grid_1_AutoML_20190421_181829_model_1	800	Modulo	0	0	MeanImputation	-3.64012e17		
- Action Output:** Shows the history of actions and their durations.

#	Time	Action	Message	Duration / Fetch
26	15:38:52	SELECT dataset.dataset_name, dataset.id, COUNT(model_meta.run_time) FROM ...	1 row(s) returned	0.000 sec / 0.000 sec
27	15:39:35	SELECT * FROM final_final.gbm_all_mod_view LIMIT 0, 1000	8 row(s) returned	0.000 sec / 0.000 sec
28	15:40:21	SELECT * FROM final_final.glm_all_mod_view LIMIT 0, 1000	134 row(s) returned	0.000 sec / 0.000 sec
29	15:40:40	SELECT * FROM final_final.glm_all_mod_view LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec

## VIEW 4

### NAÏVE\_BAYES VIEW

```
-- View all Naive Bayes models
CREATE VIEW nb_all_mod_view AS
SELECT naive_bayes_hpara.* , models_metrics.rmse FROM
models_metrics JOIN naive_bayes_hpara
ON
models_metrics.model_name = naive_bayes_hpara.model_id
WHERE models_metrics.algo_id = 'A1'
ORDER BY naive_bayes_hpara.run_time,models_metrics.rmse ;
```

### NAÏVE\_BAYES VIEW OUTPUT

The screenshot shows a database interface with several panes:

- Top Bar:** Shows tabs for "final\_final\_VIEWS", "glm\_all\_mod\_view", and "nb\_all\_mod\_view".
- Query Editor:** Displays the SQL command: `1 • SELECT * FROM final_final.nb_all_mod_view;`
- Result Grid:** A large grid area where results are displayed. The columns are labeled: model\_id, run\_time, weights\_column, offset\_column, fold\_column, fold\_assignment, stopping\_rounds, max\_runtime\_secs, stopping\_metric, stopping\_tolerance, laplace, and min\_sdev.
- Action Output:** A log of recent actions at the bottom left. It includes columns for #, Time, Action, Message, and Duration / Fetch.

#	Time	Action	Message	Duration / Fetch
27	15:39:35	SELECT * FROM final_final.drf_all_mod_view LIMIT 0, 1000	8 row(s) returned	0.000 sec / 0.000 sec
28	15:40:21	SELECT * FROM final_final.gbm_all_mod_view LIMIT 0, 1000	134 row(s) returned	0.000 sec / 0.000 sec
29	15:40:40	SELECT * FROM final_final(glm_all_mod_view LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec
30	15:41:29	SELECT * FROM final_final(nb_all_mod_view LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

- Output:** A section below the action log with a "Read On" button.

## VIEW 5

### DRF VIEW INPUT

```
-- View all DRF models
CREATE VIEW drf_all_mod_view AS
SELECT drf_hpara.* , models_metrics.rmse FROM
models_metrics JOIN drf_hpara
ON
models_metrics.model_name = drf_hpara.model_id
WHERE models_metrics.algo_id = 'A3'
ORDER BY drf_hpara.run_time,models_metrics.rmse ;
```

### DRF VIEW OUTPUT

The screenshot shows a database interface with a query window and a results grid.

Query Window:

```
final_final_VIEWS on_all_mod_view
1 • | SELECT * FROM final_final.drf_all_mod_view;
```

Result Grid:

model_id	run_time	fold_assignment	fold_column	offset_column	weights_column	balance_classes	class_sampling_factors	max_after_balance_size	n
DRF_1_AutoML_20190420_204025	300	Modulo	NULL	NULL	NULL	FALSE	NULL	5	16
DRF_1_AutoML_20190420_203958	300	Modulo	NULL	NULL	NULL	TRUE	NULL	5	15
DRF_1_AutoML_20190420_204019	300	Modulo	NULL	NULL	NULL	TRUE	NULL	5	15
DRF_1_AutoML_20190419_152459	500	Modulo	NULL	NULL	NULL	FALSE	NULL	5	15

Action Output:

#	Time	Action	Message	Duration / Fetch
24	15:37:33	SELECT gbm_hpara.model_id , gbm_hpara.learn_rate , models_metrics.run_time.m...	134 row(s) returned	0.000 sec / 0.000 sec
25	15:38:18	SELECT DISTINCT (col_sample_rate), COUNT(model_id) FROM gbm_hpara GRO...	4 row(s) returned	0.000 sec / 0.000 sec
26	15:38:52	SELECT dataset.dataset_name, dataset.id, COUNT(model_meta.run_time) FROM ...	1 row(s) returned	0.000 sec / 0.000 sec
27	15:39:35	SELECT * FROM final_final.drf_all_mod_view LIMIT 0, 1000	8 row(s) returned	0.000 sec / 0.000 sec

## RUN TIME FOR ALL VIEWS.

36	12:50:23	CREATE VIEW gbm_algorithm AS SELECT models.model_name, mod_metrics.rmse, gbm_hpara.* FROM models INNER JOIN mod_metrics... 0 row(s) affected	0.254 sec
37	12:50:28	CREATE VIEW glm_algorithm AS SELECT models.model_name, mod_metrics.rmse, glm_hpara.* FROM models INNER JOIN mod_metrics... 0 row(s) affected	0.141 sec
38	12:50:32	CREATE VIEW drf_algorithm AS SELECT models.model_name, mod_metrics.rmse, drf_hpara.* FROM models INNER JOIN mod_metrics... 0 row(s) affected	0.234 sec
39	12:50:35	CREATE VIEW naive_bayes_algorithm AS SELECT models.model_name, mod_metrics.rmse, naive_bayes_hpara.* FROM models INNER JOIN mod_metrics... 0 row(s) affected	0.171 sec
40	12:50:39	CREATE VIEW xg_boost_algorithm AS SELECT models.model_name, mod_metrics.rmse, xg_boost_hpara.* FROM models INNER JOIN mod_metrics... 0 row(s) affected	0.203 sec

### ALL VIEWS

- ▼ **final\_final**
  - ▶ Tables
  - ▼ Views
    - ▶ drf\_all\_mod\_view
    - ▶ gbm\_all\_mod\_view
    - ▶ glm\_all\_mod\_view
    - ▶ nb\_all\_mod\_view
    - ▶ xgb\_all\_mod\_view
  - ▶ Stored Procedures
  - ▶ Functions
- ▶ md
- ▶ test db chandani

### ALL FUNCTIONS

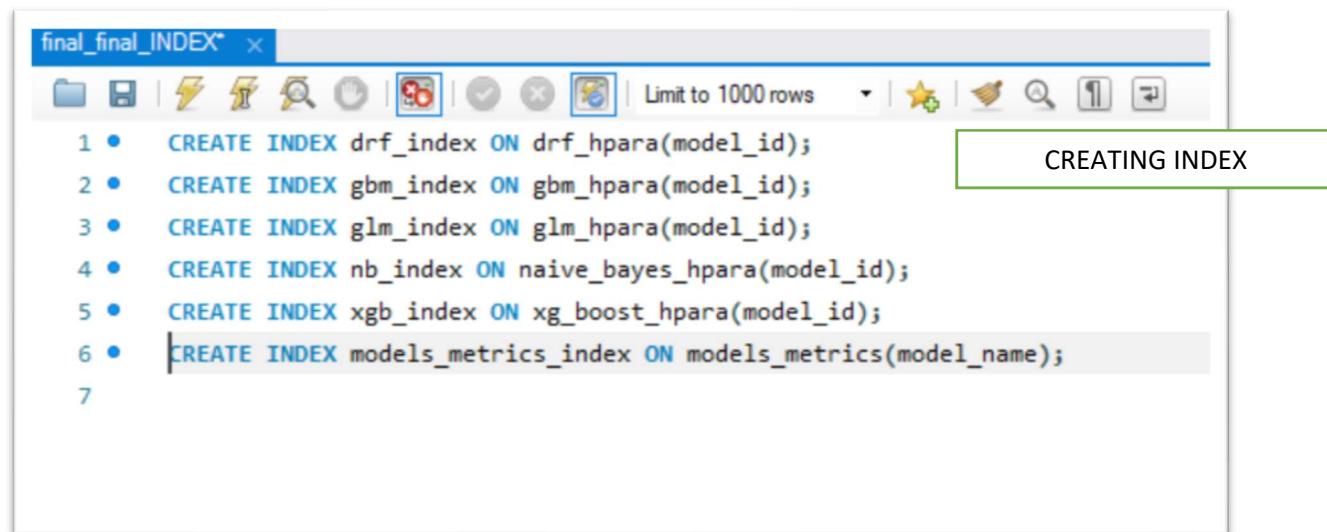
#### SCHEMAS

Filter objects

- ▶ Tables
- ▶ Views
- ▶ Stored Procedures
- ▼ Functions
  - f() avg\_rmse
  - f() class\_or\_reg
  - f() count\_models
  - f() diff\_best\_worst

## INDEXING

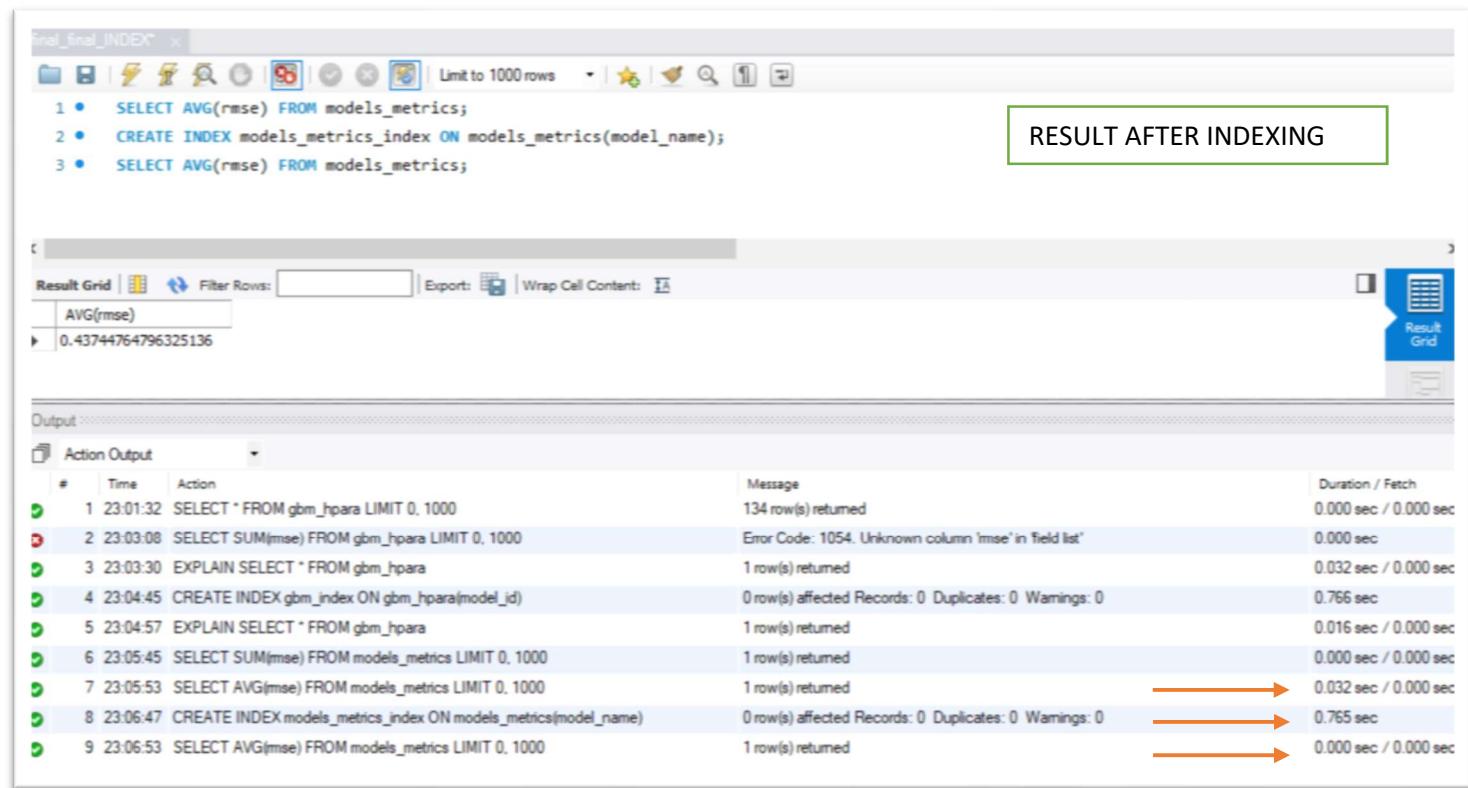
Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book. By creating an index we improved the efficiency of the search type by restricting the search to limited dataset.



```
final_final_INDEX* ×
CREATE INDEX drf_index ON drf_hpara(model_id);
CREATE INDEX gbm_index ON gbm_hpara(model_id);
CREATE INDEX glm_index ON glm_hpara(model_id);
CREATE INDEX nb_index ON naive_bayes_hpara(model_id);
CREATE INDEX xgb_index ON xg_boost_hpara(model_id);
CREATE INDEX models_metrics_index ON models_metrics(model_name);
```

CREATING INDEX

Showing the example of how indexing improved the performance



```
final_final_INDEX* ×
SELECT AVG(rmse) FROM models_metrics;
CREATE INDEX models_metrics_index ON models_metrics(model_name);
SELECT AVG(rmse) FROM models_metrics;
```

RESULT AFTER INDEXING

Result Grid
AVG(rmse) 0.43744764796325136

Action Output

#	Time	Action	Message	Duration / Fetch
1	23:01:32	SELECT * FROM gbm_hpara LIMIT 0, 1000	134 row(s) returned	0.000 sec / 0.000 sec
2	23:03:08	SELECT SUM(mse) FROM gbm_hpara LIMIT 0, 1000	Error Code: 1054. Unknown column 'mse' in 'field list'	0.000 sec
3	23:03:30	EXPLAIN SELECT * FROM gbm_hpara	1 row(s) returned	0.032 sec / 0.000 sec
4	23:04:45	CREATE INDEX gbm_index ON gbm_hpara(model_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.766 sec
5	23:04:57	EXPLAIN SELECT * FROM gbm_hpara	1 row(s) returned	0.016 sec / 0.000 sec
6	23:05:45	SELECT SUM(mse) FROM models_metrics LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
7	23:05:53	SELECT AVG(mse) FROM models_metrics LIMIT 0, 1000	1 row(s) returned	0.032 sec / 0.000 sec
8	23:06:47	CREATE INDEX models_metrics_index ON models_metrics(model_name)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.765 sec
9	23:06:53	SELECT AVG(mse) FROM models_metrics LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

After giving the index to that table we can see the improved performance.

## FUNCTIONS:

### Functions and their practical use

Once a **function** is defined, it can be used over and over again, and we can invoke the same **function** many times in our database. which saves us the work. We have defined a set of function to classify the Hyperparameters based on the use.

#### AVERAGE Function:

The Function here gives the value of average RMSE for each algorithm

```
USE final_final;
-- Function to calculate the average of all RMSE models for a given algorithm
DELIMITER $$

CREATE FUNCTION avg_rmse(algo_id_ip VARCHAR(10)) RETURNS DOUBLE
DETERMINISTIC
BEGIN
    DECLARE avg_rmse DOUBLE;
    SET @total_models = (SELECT COUNT(models_metrics.model_name) FROM models_metrics WHERE models_metrics.algo_id = algo_id_ip);
    SET @total_rmse = (SELECT SUM(rmse) FROM models_metrics WHERE models_metrics.algo_id = algo_id_ip);
    SET avg_rmse = (@total_rmse/@total_models);
RETURN avg_rmse;
END $$

DELIMITER ;
```

FUNCTION 1 INPUT

The screenshot shows a MySQL Workbench interface. The top bar has tabs for 'final\_final\_Functions' and 'avg\_rmse'. The main area contains a query editor with the following content:

```
1 • select final_final.avg_rmse('A2');
2
```

Below the query editor is a results grid. The first row shows the function call:

final_final.avg_rmse('A2')
----------------------------

The second row shows the result of the function call:

0.4403867173782537
--------------------

FUNCTION 1 OUTPUT

## CHANGE IN RMSE:

The function here shows by how the Values of RMSE has improved for a particular algorithm amongst each of its Model by Calculating the difference of RMSE between the Best and worst Models amongst the generated models.

```
-- Difference between best and worst of an algorithm  
DELIMITER $$  
CREATE FUNCTION diff_best_worst(algorithm_id VARCHAR(100)) RETURNS FLOAT  
DETERMINISTIC  
BEGIN  
    DECLARE minn FLOAT;  
    DECLARE maxx FLOAT;  
    DECLARE diff FLOAT;  
    SET minn = (SELECT MIN(rmse) FROM models_metrics WHERE models_metrics.algo_id = algorithm_id);  
    SET maxx = (SELECT MAX(rmse) FROM models_metrics WHERE models_metrics.algo_id = algorithm_id);  
    SET diff = maxx - minn;  
    RETURN diff;  
END $$  
DELIMITER ;
```

FUNCTION 2 INPUT

final\_final\_Functions diff\_best\_worst x

FUNCTION 2 OUTPUT

The screenshot shows a MySQL Workbench interface. The top tab bar has 'final\_final\_Functions' and 'diff\_best\_worst x'. The main area contains the function definition and a query window. The query window has two lines: '1 • select final\_final.diff\_best\_worst('A3');' and '2'. Below the query window is a result grid. The first row of the grid has a left column icon and a right column containing 'final\_final.diff\_best\_worst('A3')'. The second row has a left column icon and a right column containing '0.015119999647140503'. At the bottom of the result grid are buttons for 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'.

## MODELS OF EACH ALGORITHM:

The function here gives the count of all generated including of all runtime for each algorithm

```
-- To display the count of models for a given algorithm id  
DELIMITER $$  
CREATE FUNCTION count_models(algorithm_id VARCHAR(100)) RETURNS DOUBLE  
DETERMINISTIC  
BEGIN  
    DECLARE count_algo INT;  
    SET count_algo = (SELECT COUNT(model_name) FROM models_metrics WHERE algorithm_id = models_metrics.algo_id);  
    RETURN count_algo;  
END $$  
DELIMITER ;
```

FUNCTION 3 INPUT

final\_final\_Functions count\_models

1 • select final\_final.count\_models('A2');  
2

Result Grid | Filter Rows: Export: Wrap Cell Content:

final_final.count_models('A2')
142

**PROBLEM TYPE :**

The function here gives the problem type of the datatype whether it is a classification or Regression type

```
-- To find if the dataset is used for regression or classification type of work
DELIMITER $$

CREATE FUNCTION class_or_reg(file_id_2 VARCHAR(100)) RETURNS VARCHAR(100)
DETERMINISTIC
BEGIN
    DECLARE filetype2 VARCHAR(100);
    SET filetype2 = (SELECT class_or_regression FROM dataset WHERE dataset.id = file_id_2);
RETURN filetype2;
END $$

DELIMITER ;
```

FUNCTION 4 INPUT

FUNCTION 4 OUTPUT

The screenshot shows the MySQL Workbench interface with a query editor window titled 'final\_final\_Functions' containing the following SQL code:

```
1 • | select final_final.class_or_reg('D1');|
2
```

The result grid below shows the output of the query:

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
final_final.class_or_reg(D1)			
Classification			

## USE CASES:

The queries run to get the desired outputs from the database which can be further used further for practical uses is termed as a use case for a database.

The Following are the queries which we run in MySQL for the Hyperparameter Database

- USE CASE 1 – To find the best GBM algorithm for a run-time.

USE CASE 1 INPUT

-- Use Case 1 : To find the best GBM algorithm along for a run-time

```
1 SELECT
2   gbm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, models_metrics.algo_id, models_metrics.run_time, models_metrics.AUC,
3   models_metrics.logloss, models_metrics.mae, models_metrics.rmse, models_metrics.mse
4   FROM
5   gbm_hpara JOIN models_metrics ON
6   gbm_hpara.model_id = models_metrics.model_name
7   JOIN algo_h2o ON
8   algo_h2o.algo_id = models_metrics.algo_id
9   WHERE models_metrics.run_time = 300
10  ORDER BY models_metrics.rmse;
```

USE CASE 1 OUTPUT

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

model_id	algo_name	dataset_id	algo_id	run_time	AUC	logloss	mae	rmse	mse
GBM_grid_1_AutoML_20190420_203958...	Gradient Boosting Machine GBM	D1	A2	300	0.756161	0.508591	0.299041	0.413809	0.171238
GBM_5_AutoML_20190420_203958	Gradient Boosting Machine GBM	D1	A2	300	0.746315	0.514319	0.292398	0.416938	0.173838
GBM_grid_1_AutoML_20190420_203958...	Gradient Boosting Machine GBM	D1	A2	300	0.744142	0.516116	0.291218	0.417226	0.174078
GBM_5_AutoML_20190420_204025	Gradient Boosting Machine GBM	D1	A2	300	0.745365	0.515644	0.301186	0.417315	0.174152
GBM_grid_1_AutoML_20190420_203958...	Gradient Boosting Machine GBM	D1	A2	300	0.747323	0.512397	0.300243	0.417659	0.174439
GBM_1_AutoML_20190420_204025	Gradient Boosting Machine GBM	D1	A2	300	0.756751	0.517751	0.311608	0.417823	0.174576
GBM_grid_1_AutoML_20190420_204025...	Gradient Boosting Machine GBM	D1	A2	300	0.742594	0.51332	0.29961	0.417956	0.174687
GBM_grid_1_AutoML_20190420_203958...	Gradient Boosting Machine GBM	D1	A2	300	0.74638	0.519715	0.295414	0.418382	0.175044
GBM_grid_1_AutoML_20190420_204025...	Gradient Boosting Machine GBM	D1	A2	300	0.770397	0.533106	0.303151	0.418952	0.175521

Result 1 ×

Output

Action Output

#	Time	Action	Message	Duration / F
14	15:16:30	select final_final.class_or_reg(D1) LIMIT 0, 1000	1 row(s) returned	0.000 sec /
15	15:18:59	select final_final.count_models(A2) LIMIT 0, 1000	1 row(s) returned	0.000 sec /
16	15:19:56	select final_final.diff_best_worst(A3) LIMIT 0, 1000	1 row(s) returned	0.000 sec /
17	15:29:48	SELECT obm.hoara.model_id, algo_h2o.algo_name, models.metrics.dataset_id, ...	36 row(s) returned	0.000 sec /

- USE CASE 2 - To find the best GLM algorithm along for a run-time

### USE CASE 2 INPUT

```
-- Use Case 2 : To find the best GLM algorithm along for a run-time
SELECT
glm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, models_metrics.algo_id, models_metrics.run_time, models_metrics.AUC,
models_metrics.logloss, models_metrics.mae, models_metrics.rmse, models_metrics.mse
FROM
glm_hpara JOIN models_metrics ON
glm_hpara.model_id = models_metrics.model_name
JOIN algo_h2o ON
algo_h2o.algo_id = models_metrics.algo_id
WHERE models_metrics.run_time = 300
ORDER BY models_metrics.rmse;
```

### USE CASE 2 OUTPUT

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

model_id	algo_name	dataset_id	algo_id	run_time	AUC	logloss	mae	rmse	mse
GLM_grid_1_AutoML_20190420_204025_model_1	Generalized Linear Model GLM	D1	A1	300	0.751699	0.503103	0.2955	0.413658	0.171113
GLM_grid_1_AutoML_20190420_203958_model_1	Generalized Linear Model GLM	D1	A1	300	0.751699	0.503103	0.2955	0.413658	0.171113
GLM_grid_1_AutoML_20190420_204019_model_1	Generalized Linear Model GLM	D1	A1	300	0.751699	0.503103	0.2955	0.413658	0.171113

Result 2 ×

Output

Action Output

#	Time	Action	Message	Duration / Fetch
15	15:18:59	select final_final.count_models(A2) LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.0
16	15:19:56	select final_final.diff_best_worst(A3) LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.0
17	15:29:48	SELECT gbm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, ...	36 row(s) returned	0.000 sec / 0.0
18	15:31:13	SELECT glm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, m...	3 row(s) returned	0.188 sec / 0.0

- USE CASE 3 – To find the best DRF algorithm along for a run-time

### USE CASE 3 INPUT

```
-- Use Case 3 : To find the best DRF algorithm along for a run-time
SELECT
drf_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, models_metrics.algo_id, models_metrics.run_time, models_metrics.AUC,
models_metrics.logloss, models_metrics.mae, models_metrics.rmse, models_metrics.mse
FROM
drf_hpara JOIN models_metrics ON
drf_hpara.model_id = models_metrics.model_name
JOIN algo_h2o ON
algo_h2o.algo_id = models_metrics.algo_id
WHERE models_metrics.run_time = 300
ORDER BY models_metrics.rmse;
```

### USE CASE 3 OUTPUT

model_id	algo_name	dataset_id	algo_id	run_time	AUC	logloss	mae	rmse	mse
DRF_1_AutoML_20190420_204025	Distributed Random Forest DRF	D1	A3	300	0.735275	0.74172	0.315364	0.426168	0.181619
DRF_1_AutoML_20190420_203958	Distributed Random Forest DRF	D1	A3	300	0.722464	0.642488	0.319193	0.430645	0.185455
DRF_1_AutoML_20190420_204019	Distributed Random Forest DRF	D1	A3	300	0.704701	0.54628	0.338532	0.434363	0.188671

Result 3 ×

Output

#	Time	Action	Message	Duration / Fetch
16	15:19:56	select final_final.dff_best_worst(A3) LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 s
17	15:29:48	SELECT gbm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, ...	36 row(s) returned	0.000 sec / 0.000 s
18	15:31:13	SELECT glm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, m...	3 row(s) returned	0.188 sec / 0.000 s
19	15:31:47	SELECT df_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, m...	3 row(s) returned	0.016 sec / 0.000 s

## USE CASE 4- To find the best XGBoost algorithm along for a run-time

### USE CASE 4 INPUT

```
-- Use Case 4 : To find the best XGBoost algorithm along for a run-time
SELECT
xg_boost_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, models_metrics.algo_id, models_metrics.run_time, models_metrics.AUC
models_metrics.logloss, models_metrics.mae, models_metrics.rmse, models_metrics.mse
FROM
xg_boost_hpara JOIN models_metrics ON
xg_boost_hpara.model_id = models_metrics.model_name
JOIN algo_h2o ON
algo_h2o.algo_id = models_metrics.algo_id
WHERE models_metrics.run_time = 300
ORDER BY models metrics.rmse;
```

### USE CASE 4 OUTPUT

Result Grid										Result Grid
model_id	algo_name	dataset_id	algo_id	run_time	AUC	logloss	mae	rmse	mse	Read Only
Result 4										
Output										
Action Output										
#	Time	Action	Message	Duration / Fetch						
17	15:29:48	SELECT gbm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, ...	36 row(s) returned	0.000 sec / 0.000 sec						
18	15:31:13	SELECT glm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, m...	3 row(s) returned	0.188 sec / 0.000 sec						
19	15:31:47	SELECT df_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, m...	3 row(s) returned	0.016 sec / 0.000 sec						
20	15:32:09	SELECT xg_boost_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset...	0 row(s) returned	0.047 sec / 0.000 sec						

## USE CASE 5: To find the best Naive Bayes algorithm along for a run-time

USE CASE 5 INPUT

```
-- Use Case 5 : To find the best Naive Bayes algorithm along for a run-time
SELECT
naive_bayes_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, models_metrics.algo_id, models_metrics.run_time, models_metrics.
models_metrics.logloss, models_metrics.mae, models_metrics.rmse, models_metrics.mse
FROM
naive_bayes_hpara JOIN models_metrics ON
naive_bayes_hpara.model_id = models_metrics.model_name
JOIN algo_h2o ON
algo_h2o.algo_id = models_metrics.algo_id
WHERE models_metrics.run_time = 300
ORDER BY models_metrics.rmse;
```

USE CASE 5 OUTPUT

Result Grid										Result Grid
Action Output										Read C
#	Time	Action	Message							Duration / Fetch
#	Time	Action	Message	Duration / Fetch						
18	15:31:13	SELECT glm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, m...	3 row(s) returned	0.188 sec / 0.000 sec						
19	15:31:47	SELECT df_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, m...	3 row(s) returned	0.016 sec / 0.000 sec						
20	15:32:09	SELECT xg_boost_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset...	0 row(s) returned	0.047 sec / 0.000 sec						
21	15:33:01	SELECT naive_bayes_hpara.model_id, algo_h2o.algo_name, models_metrics.dat...	0 row(s) returned	0.000 sec / 0.000 sec						

## USE CASE 6: Count the number of algorithms generated for each runtime

USE CASE 6 INPUT

```
-- Use Case 6 : Count the number of algorgirms generated for each runtime
SELECT algo_h2o.algo_name,models_metrics.run_time, COUNT(algo_h2o.algo_name) AS 'No of models'
FROM models_metrics JOIN algo_h2o
ON models_metrics.algo_id = algo_h2o.algo_id
GROUP BY algo_h2o.algo_name, models_metrics.run_time;
```

USE CASE 6 OUTPUT

The screenshot shows a database interface with the following components:

- Result Grid:** A table titled "Result Grid" with columns "algo\_name", "run\_time", and "No of models". The data is as follows:

algo_name	run_time	No of models
Generalized Linear Model GLM	500	1
Generalized Linear Model GLM	300	3
Generalized Linear Model GLM	800	1
Generalized Linear Model GLM	1000	1
Generalized Linear Model GLM	1200	2
Gradient Boosting Machine GBM	500	40
Gradient Boosting Machine GBM	300	36
Gradient Boosting Machine GBM	800	17

- Action Output:** A log of SQL actions with columns "#", "Time", "Action", "Message", and "Duration / Fetch". The log entries are:

#	Time	Action	Message	Duration / Fetch
19	15:31:47	SELECT df_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, m...	3 row(s) returned	0.016 sec / 0.000 sec
20	15:32:09	SELECT xg_boost_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset...	0 row(s) returned	0.047 sec / 0.000 sec
21	15:33:01	SELECT naive_bayes_hpara.model_id, algo_h2o.algo_name, models_metrics.dat...	0 row(s) returned	0.000 sec / 0.000 sec
22	15:35:12	SELECT algo_h2o.algo_name,models_metrics.run_time, COUNT(algo_h2o.algo_na...	21 row(s) returned	0.000 sec / 0.000 sec

- Sidebar:** Includes tabs for "Result Grid", "Form Editor", and "Read Only".

## USE CASE 7: How many trees should I include in my DRF?

### USE CASE 7 INPUT

```
-- Use Case 7: How many trees should I include in my DRF?  
SELECT drf_hpara.model_id, drf_hpara.ntrees, models_metrics.rmse  
FROM drf_hpara JOIN models_metrics  
ON  
drf_hpara.model_id = models_metrics.model_name  
ORDER BY rmse;
```

### USE CASE 7 OUTPUT

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

model_id	ntrees	rmse
DRF_1_AutoML_20190421_184305	15	0.419243
DRF_1_AutoML_20190420_204025	16	0.426168
DRF_1_AutoML_20190419_152459	15	0.427609
DRF_1_AutoML_20190421_184315	14	0.428252
DRF_1_AutoML_20190421_183436	27	0.429922
DRF_1_AutoML_20190420_203958	15	0.430645
DRF_1_AutoML_20190421_181829	12	0.433861
DRF_1_AutoML_20190420_204019	15	0.434363

Result 7 x | Read Only

Output

Action Output	#	Time	Action	Message	Duration / Fetch
20 15:32:09 SELECT xg_boost_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset...	20	15:32:09	SELECT xg_boost_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset...	0 row(s) returned	0.047 sec / 0.000 sec
21 15:33:01 SELECT naive_bayes_hpara.model_id, algo_h2o.algo_name, models_metrics.dat...	21	15:33:01	SELECT naive_bayes_hpara.model_id, algo_h2o.algo_name, models_metrics.dat...	0 row(s) returned	0.000 sec / 0.000 sec
22 15:35:12 SELECT algo_h2o.algo_name,models_metrics.run_time, COUNT(algo_h2o.algo_na...	22	15:35:12	SELECT algo_h2o.algo_name,models_metrics.run_time, COUNT(algo_h2o.algo_na...	21 row(s) returned	0.000 sec / 0.000 sec
23 15:36:26 SELECT drf_hpara.model_id, drf_hpara.ntrees, models_metrics.rmse FROM drf_hpa...	23	15:36:26	SELECT drf_hpara.model_id, drf_hpara.ntrees, models_metrics.rmse FROM drf_hpa...	8 row(s) returned	0.000 sec / 0.000 sec

## USE CASE 8: What should I set my learning rate to for GBM?

USE CASE 8 INPUT

```
-- Use Case 8: What should I set my learning rate to for GBM?
SELECT gbm_hpara.model_id , gbm_hpara.learn_rate , models_metrics.run_time,models_metrics.rmse
FROM gbm_hpara JOIN models_metrics
ON
gbm_hpara.model_id = models_metrics.model_name
ORDER BY models_metrics.run_time,models_metrics.rmse;
```

USE CASE 8 OUTPUT

The screenshot shows a database interface with two main sections: 'Result Grid' and 'Action Output'.

**Result Grid:**

model_id	learn_rate	run_time	rmse
GBM_grid_1_AutoML_20190420_203958...	0.05	300	0.413809
GBM_5_AutoML_20190420_203958...	0.1	300	0.416938
GBM_grid_1_AutoML_20190420_203958...	0.1	300	0.417226
GBM_5_AutoML_20190420_204025	0.1	300	0.417315
GBM_grid_1_AutoML_20190420_203958...	0.1	300	0.417659
GBM_1_AutoML_20190420_204025	0.1	300	0.417823
GBM_grid_1_AutoML_20190420_204025...	0.1	300	0.417956
GBM_grid_1_AutoML_20190420_203958...	0.05	300	0.418382
GBM_grid_1_AutoML_20190420_204025...	0.1	300	0.418952

**Action Output:**

#	Time	Action	Message	Duration / Fetch
21	15:33:01	SELECT naive_bayes_hpara.model_id, algo_h2o.algo_name, models_metrics.dat...	0 row(s) returned	0.000 sec / 0.000 sec
22	15:35:12	SELECT algo_h2o.algo_name,models_metrics.run_time, COUNT(algo_h2o.algo_na...	21 row(s) returned	0.000 sec / 0.000 sec
23	15:36:26	SELECT df_hpara.model_id, df_hpara.ntrees, models_metrics.rmse FROM df_hpa...	8 row(s) returned	0.000 sec / 0.000 sec
24	15:37:33	SELECT gbm_hpara.model_id , gbm_hpara.learn_rate , models_metrics.run_time,m...	134 row(s) returned	0.000 sec / 0.000 sec

## USE CASE 9: To display all distinct col\_sample\_rate values from GBM

### USE CASE 9 INPUT

```
-- Use Case 9: To display all distinct col_sample_rate value from GBM
SELECT DISTINCT (col_sample_rate), COUNT(model_id)
FROM gbm_hpara
GROUP BY col_sample_rate;
```

### USE CASE 9 OUTPUT

col_sample_rate	COUNT(model_id)	Result Grid
0.8	34	
0.7	44	
0.4	25	
1	31	

Result 9 × Read Only

Output:

Action Output	#	Time	Action	Message	Duration / Fetch
22	15:35:12		SELECT algo_h2o.algo_name,models_metrics.run_time,COUNT(algo_h2o.algo_na...	21 row(s) returned	0.000 sec / 0.000 sec
23	15:36:26		SELECT df_hpara.model_id,df_hpara.ntrees,models_metrics.mse FROM df_hpa...	8 row(s) returned	0.000 sec / 0.000 sec
24	15:37:33		SELECT gbm_hpara.model_id , gbm_hpara.learn_rate , models_metrics.run_time.m...	134 row(s) returned	0.000 sec / 0.000 sec
25	15:38:18		SELECT DISTINCT (col_sample_rate), COUNT(model_id) FROM gbm_hpara GRO...	4 row(s) returned	0.000 sec / 0.000 sec

## USE CASE 10: Number of run times for a particular Dataset

## USE CASE 10 INPUT

```
-- Use Case 10: Number of run times for a particular Dataset
SELECT
dataset.dataset_name, dataset.id, COUNT(model_meta.run_time)
FROM
dataset JOIN model_meta
ON
dataset.id = model_meta.dataset_id
GROUP BY dataset.dataset_name;
```

## USE CASE 10 OUTPUT

The screenshot shows a database interface with a result grid and an action output log.

**Result Grid:**

dataset_name	id	COUNT(model_meta.run_time)
Indian Patient liver records	D1	5

**Action Output:**

#	Time	Action	Message
23	15:36:26	SELECT df_hpara.model_id, df_hpara.ntrees, models_metrics.mse FROM df_hpa...	8 row(s) returned
24	15:37:33	SELECT gbm_hpara.model_id , gbm_hpara.learn_rate , models_metrics.run_time,m...	134 row(s) returned
25	15:38:18	SELECT DISTINCT (col_sample_rate), COUNT(model_id) FROM gbm_hpara GRO...	4 row(s) returned
26	15:38:52	SELECT dataset.dataset_name, dataset.id, COUNT(model_meta.run_time) FROM ...	1 row(s) returned

## **Conclusion:**

The hyperparameter database also uses these data to build models that can predict hyperparameters without search and for visualizing and teaching statistical concepts such as power and bias/variance tradeoff

## **References**

Nik Brown Github. [https://github.com/nikbearbrown/INFO\\_6210](https://github.com/nikbearbrown/INFO_6210)

Normalization - 1NF, 2NF, 3NF and 4NF <https://www.youtube.com/watch?v=UrYLYV7WSHM&t=71s>

W3schools <https://www.w3schools.com/sql>

<http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html>

<http://docs.h2o.ai/h2o/latest-stable/h2o-docs/grid-search.html?highlight=hyperparameters#supported-grid-search-hyperparameters>

## **Contributor Statement**

- We are thankful to Prof. Nik Brown and all the TAs who helped us to complete this project.
- Project was completed by Yash Jain and Tejas Patil

## **Copyright 2019 Yash Jain, Tejas Patil**

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.

The text in this document by Smit Doshi is licensed under the Creative Commons Attribution 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/3.0/us/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.