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).

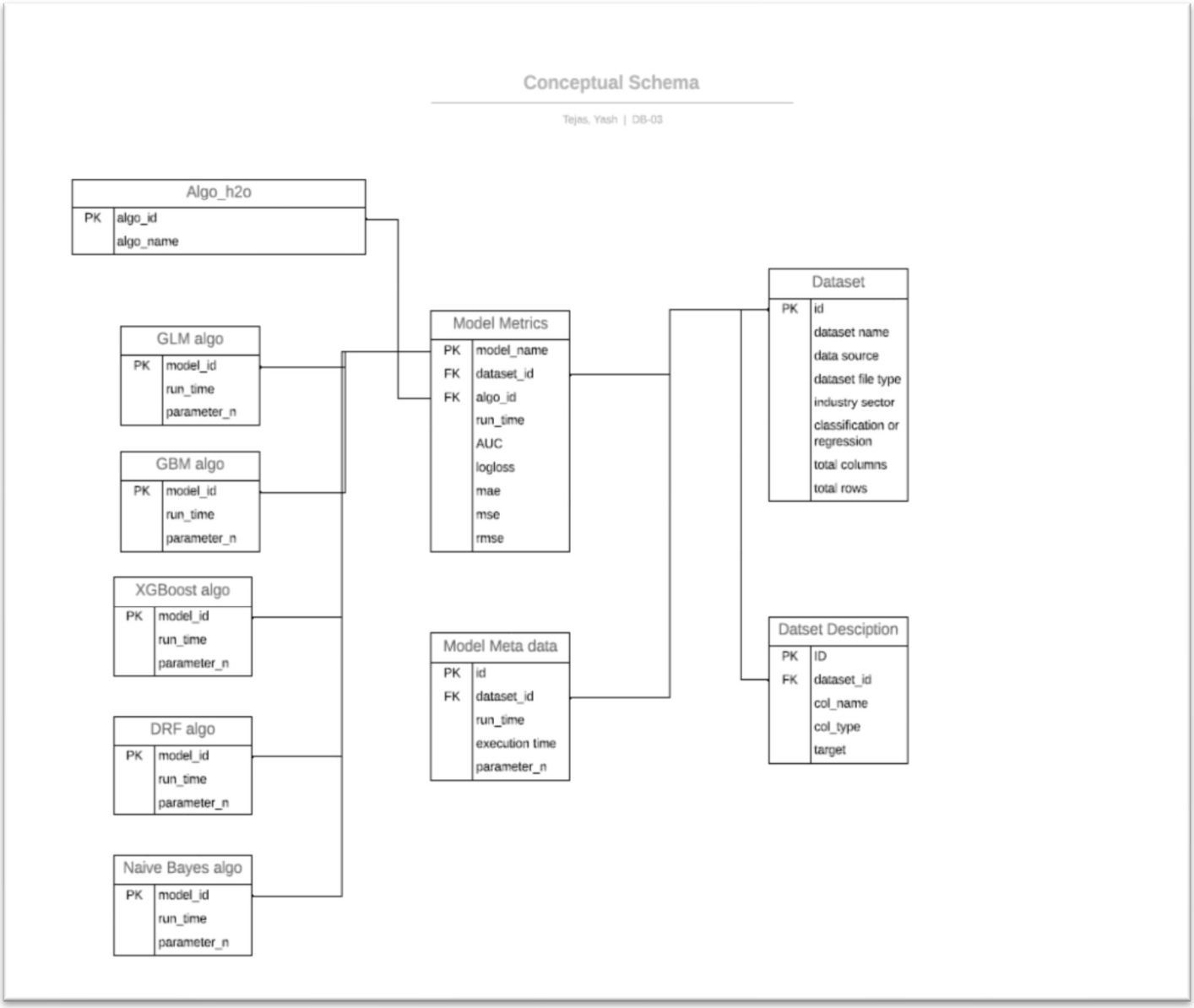
The hyperparameter database also uses these data to build models that can predict hyperparameters without search and for visualizing and teaching statistical concepts such as power and bias/variance tradeoff.

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 :

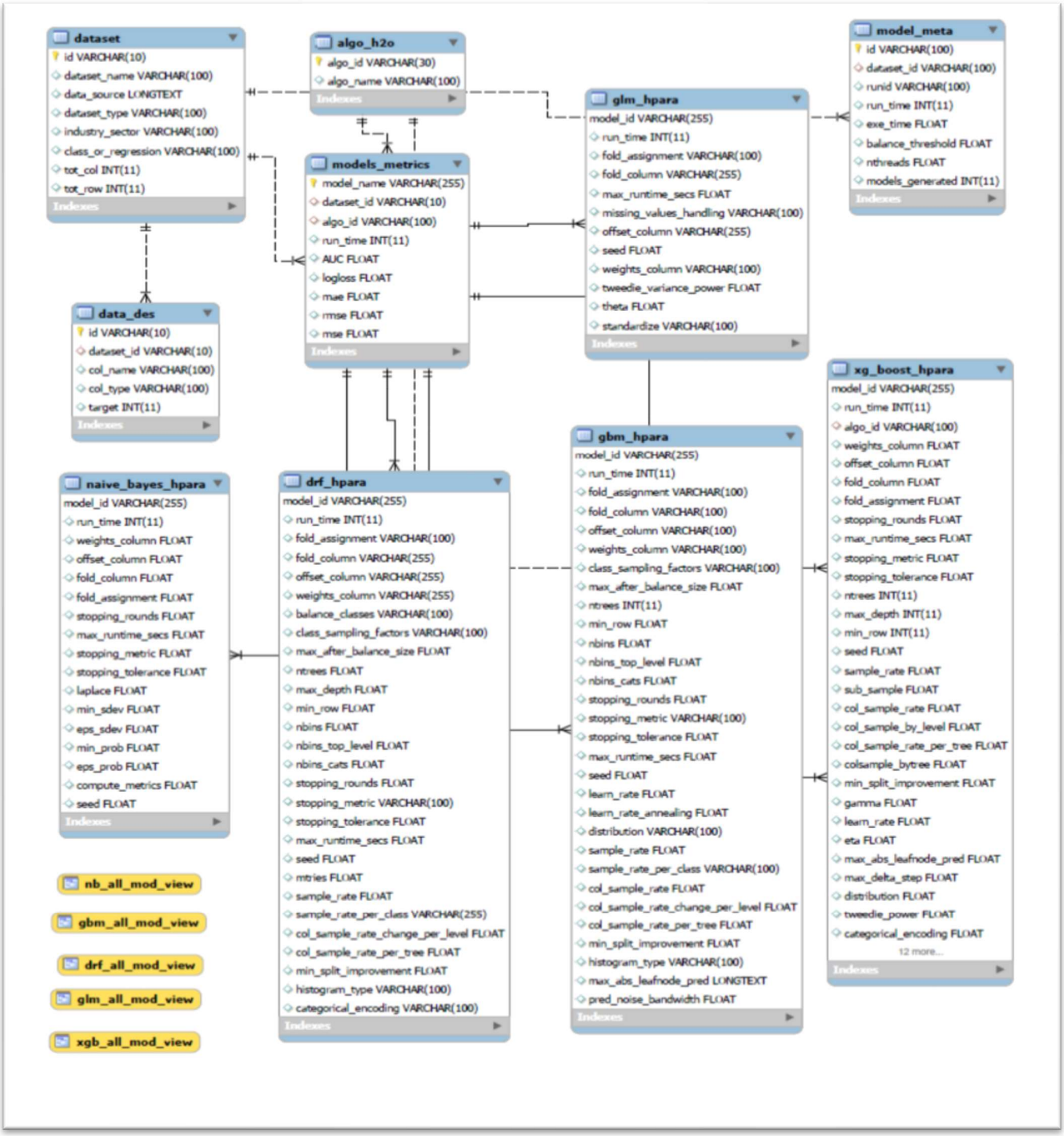
A conceptual data model captures the key entities (a person, place, concept, event or thing about which the organization wants to collect data), and the relationships between these entities. Attributes are often not added to conceptual data models as these are higher level models.



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

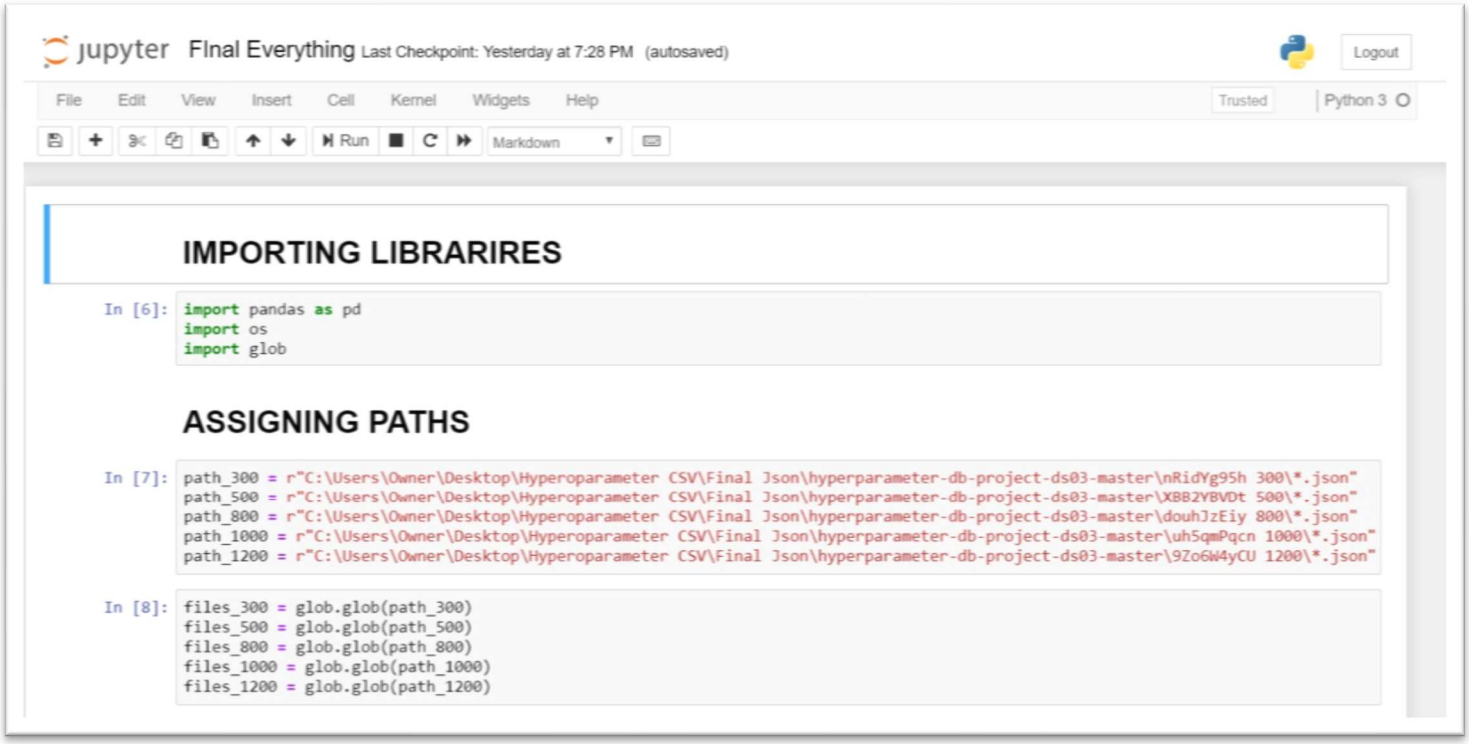
Hyperparameter Database



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.



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

Scraping all the JSON files for each model of Runtime 500

500

```
hp_XGBoost5=pd.DataFrame(columns=dict["XGBoost"])
hp_GLM5=pd.DataFrame(columns=dict["GLM"])
hp_GBM5=pd.DataFrame(columns=dict["GBM"])
hp_DRF5=pd.DataFrame(columns=dict["DRF"])
temp_name_xgb=[]
temp_name_gbm=[]
temp_name_glm=[]
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)
        datat=data1.transpose()
        data1["model_id"]=datat.actual["model_id"]["name"]
        data1["training_frame"]=datat.actual["training_frame"]["name"]
        data2=data1[dict["XGBoost"]].copy()
        data3=data2.drop("default")
        hp_XGBoost5=pd.concat([hp_XGBoost5,data3],join="inner")
        hp_XGBoost5["runtime"]=500
        hp_XGBoost5["name"]=temp_name_xgb
    elif "GLM" in f:
        data1=pd.read_json(f)
        aa=data1['model_id']['actual']['name']
        temp_name_glm.append(aa)
        datat=data1.transpose()
        data1["model_id"]=datat.actual["model_id"]["name"]
        data1["training_frame"]=datat.actual["training_frame"]["name"]
        data2=data1[dict["GLM"]].copy()
        data3=data2.drop("default")
        hp_GLM5=pd.concat([hp_GLM5,data3],join="inner")
        hp_GLM5["runtime"]=500
        hp_GLM5["name"]=temp_name_glm
    elif "GBM" in f:
        data1=pd.read_json(f)
        aa=data1['model_id']['actual']['name']
        temp_name_gbm.append(aa)
        datat=data1.transpose()
        data1["model_id"]=datat.actual["model_id"]["name"]
        data1["training_frame"]=datat.actual["training_frame"]["name"]
        data2=data1[dict["GBM"]].copy()
        data3=data2.drop("default")
        hp_GBM5=pd.concat([hp_GBM5,data3],join="inner")
        hp_GBM5["runtime"]=500
        hp_GBM5["name"]=temp_name_gbm

    elif "DRF" in f:
        data1=pd.read_json(f)
        aa=data1['model_id']['actual']['name']
        temp_name_drf.append(aa)
        datat=data1.transpose()
        data1["model_id"]=datat.actual["model_id"]["name"]
        data1["training_frame"]=datat.actual["training_frame"]["name"]
        data2=data1[dict["DRF"]].copy()
        data3=data2.drop("default")
        hp_DRF5=pd.concat([hp_DRF5,data3],join="inner")
        hp_DRF5["runtime"]=500
        hp_DRF5["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 had one.

- The values in each column of a table are atomic (No multi-value attributes allowed
  - Cleaned the data so there we 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

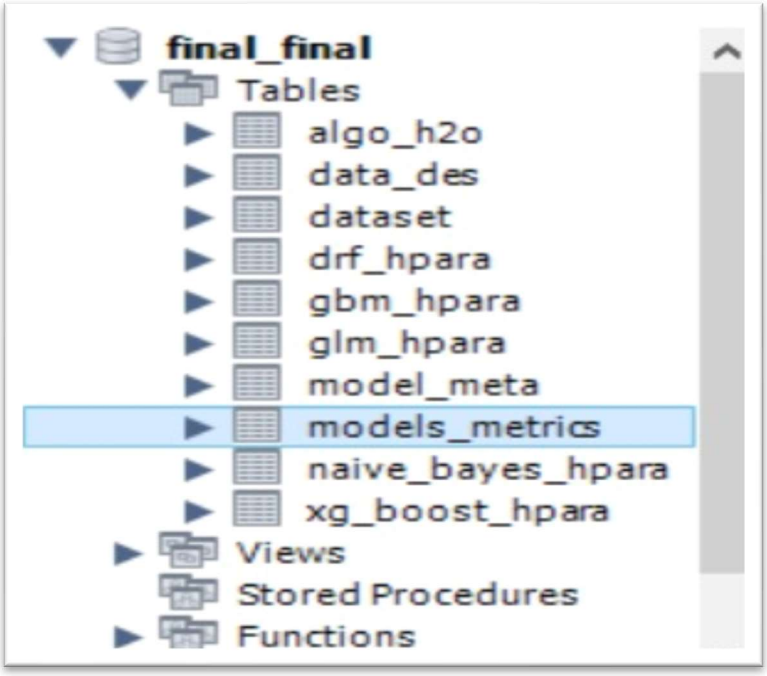
FINAL FINAL Creating Database

Limit to 1000 rows

```
1  CREATE DATABASE IF NOT EXISTS FINAL_FINAL;
2  USE FINAL_FINAL;
3  CREATE TABLE dataset
4  (
5      id VARCHAR(10),
6      dataset_name VARCHAR(100),
7      data_source LONGTEXT,
8      dataset_type VARCHAR(100),
9      industry_sector VARCHAR(100),
10     class_or_regression VARCHAR(100),
11     tot_col INT,
12     tot_row INT,
13     PRIMARY KEY(id)
14 );
```

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

Final Tables



VIEWS:

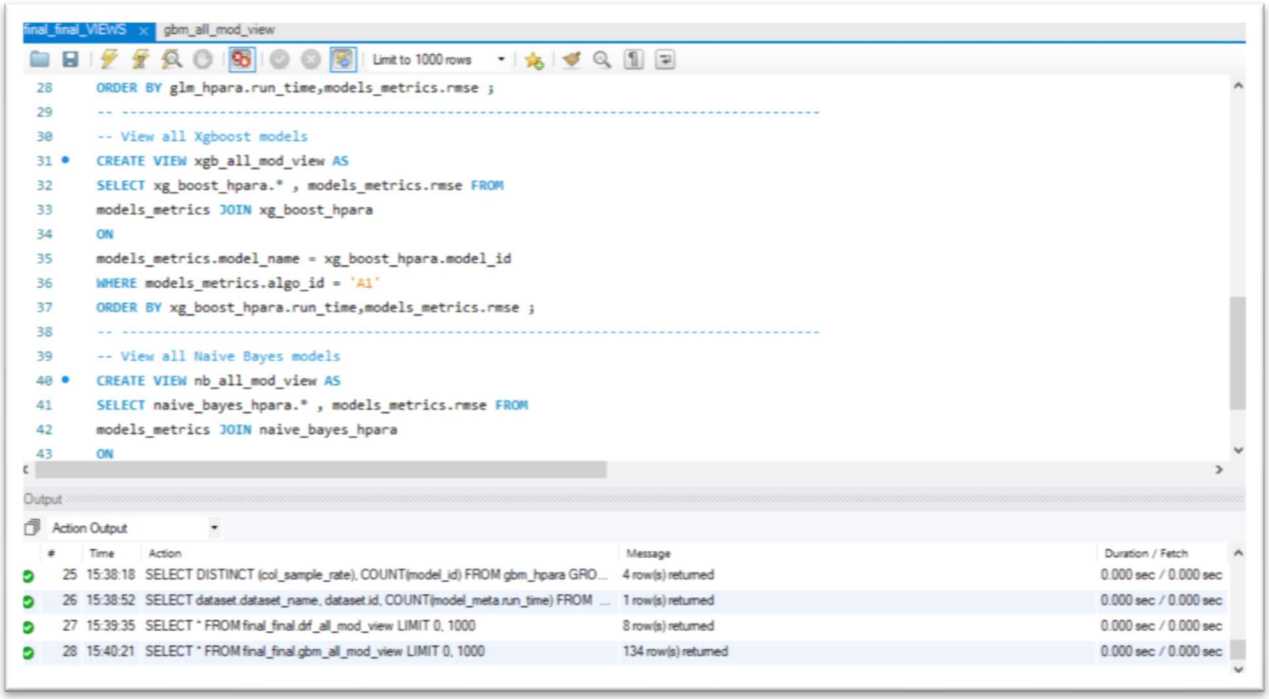
View can be described as virtual table which derived its data from one or more than one table columns. It is stored in the database. **View** can be created using tables of same database or different database. It is **used to** implement the security mechanism in the **SQL** Server

VIEW 1

GBM VIEW

```
USE final_final;
-- ----------------------------------------------------------------------------------------
-- View all GBM models
CREATE VIEW gbm_all_mod_view AS
SELECT gbm_hpara.* , models_metrics.rmse FROM
models_metrics JOIN gbm_hpara
ON
models_metrics.model_name = gbm_hpara.model_id
WHERE models_metrics.algo_id = 'A2'
ORDER BY gbm_hpara.run_time,models_metrics.rmse ;
-- ----------------------------------------------------------------------------------------
```

GBM VIEW OUTPUT

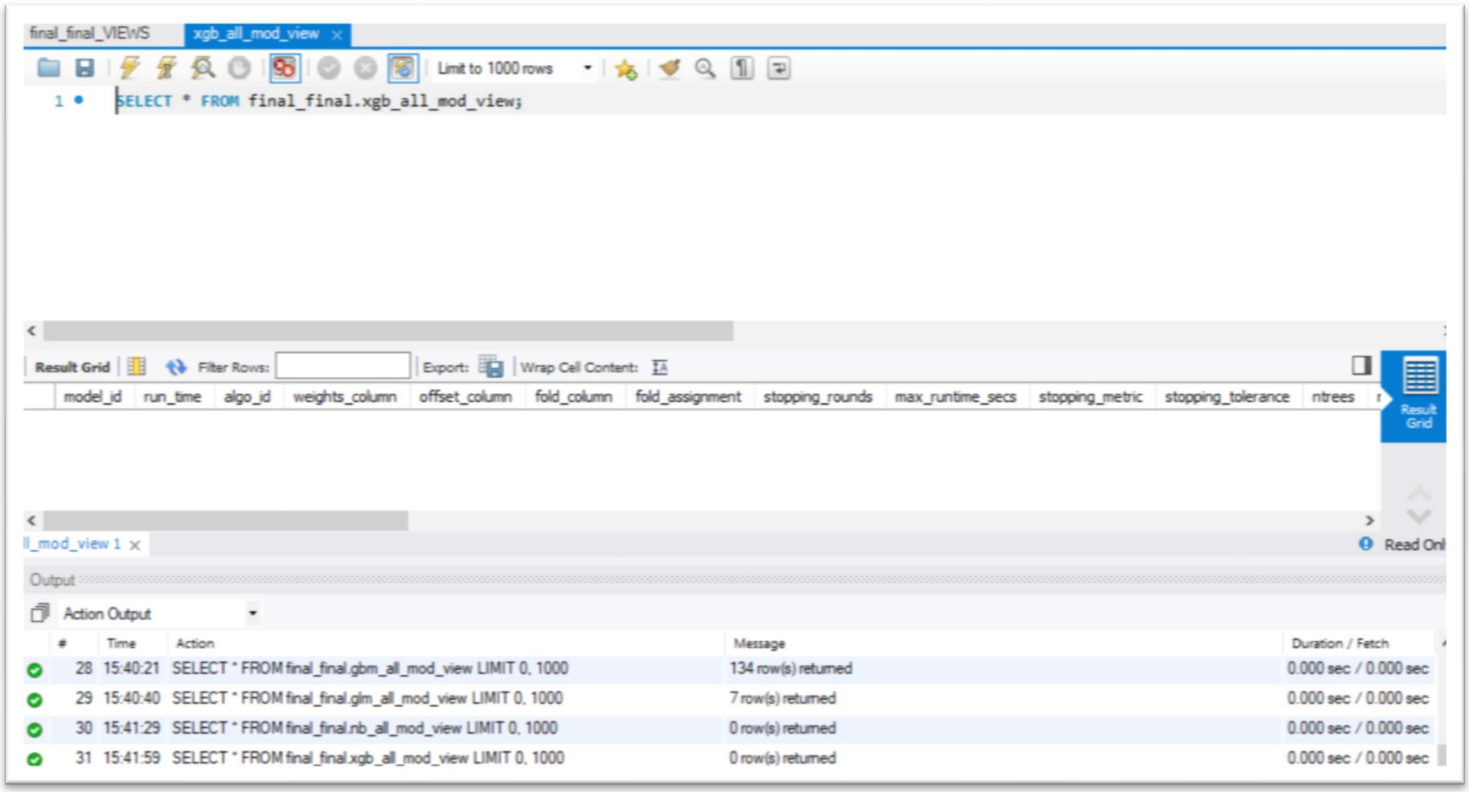


VIEW 2:

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

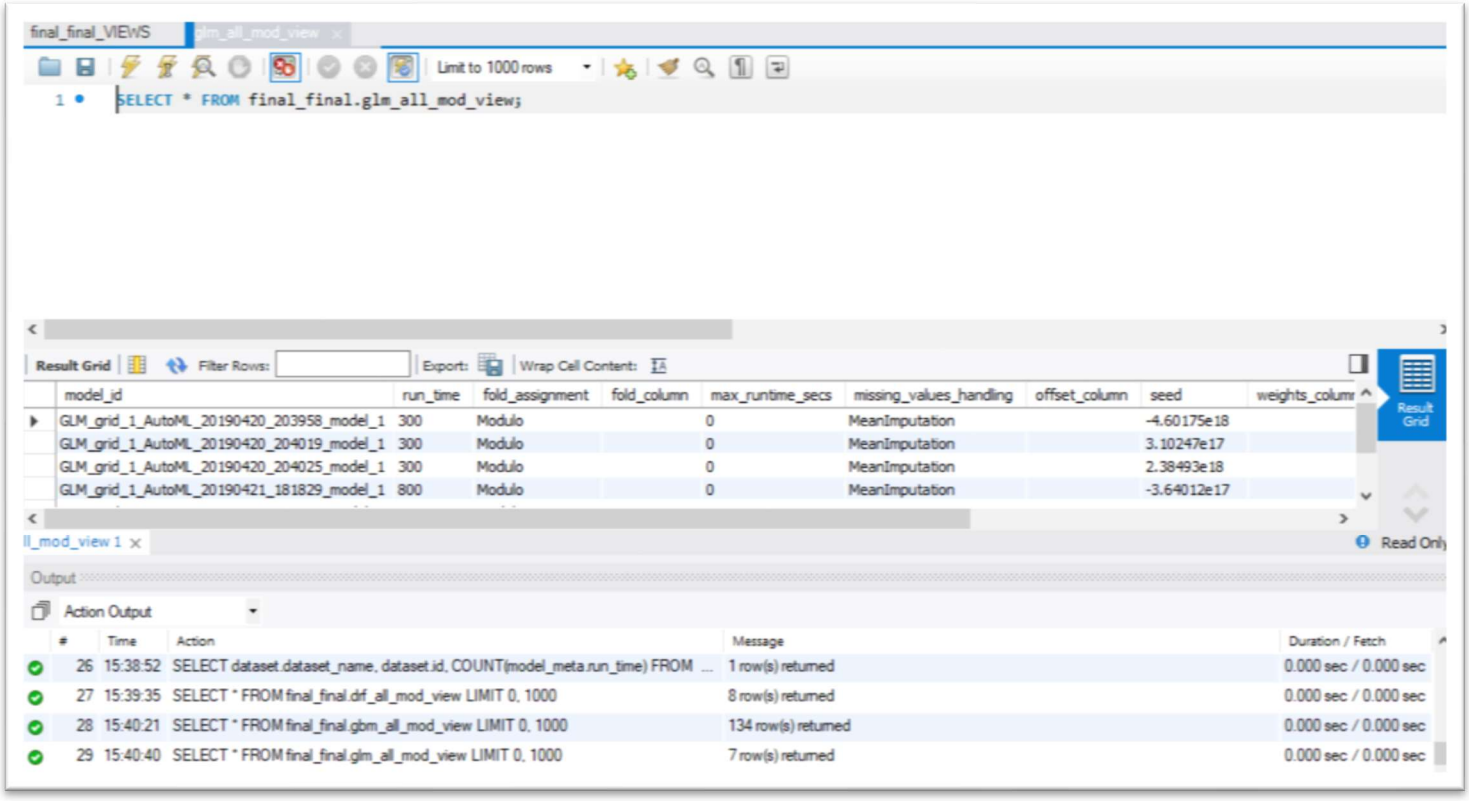


VIEW 3

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

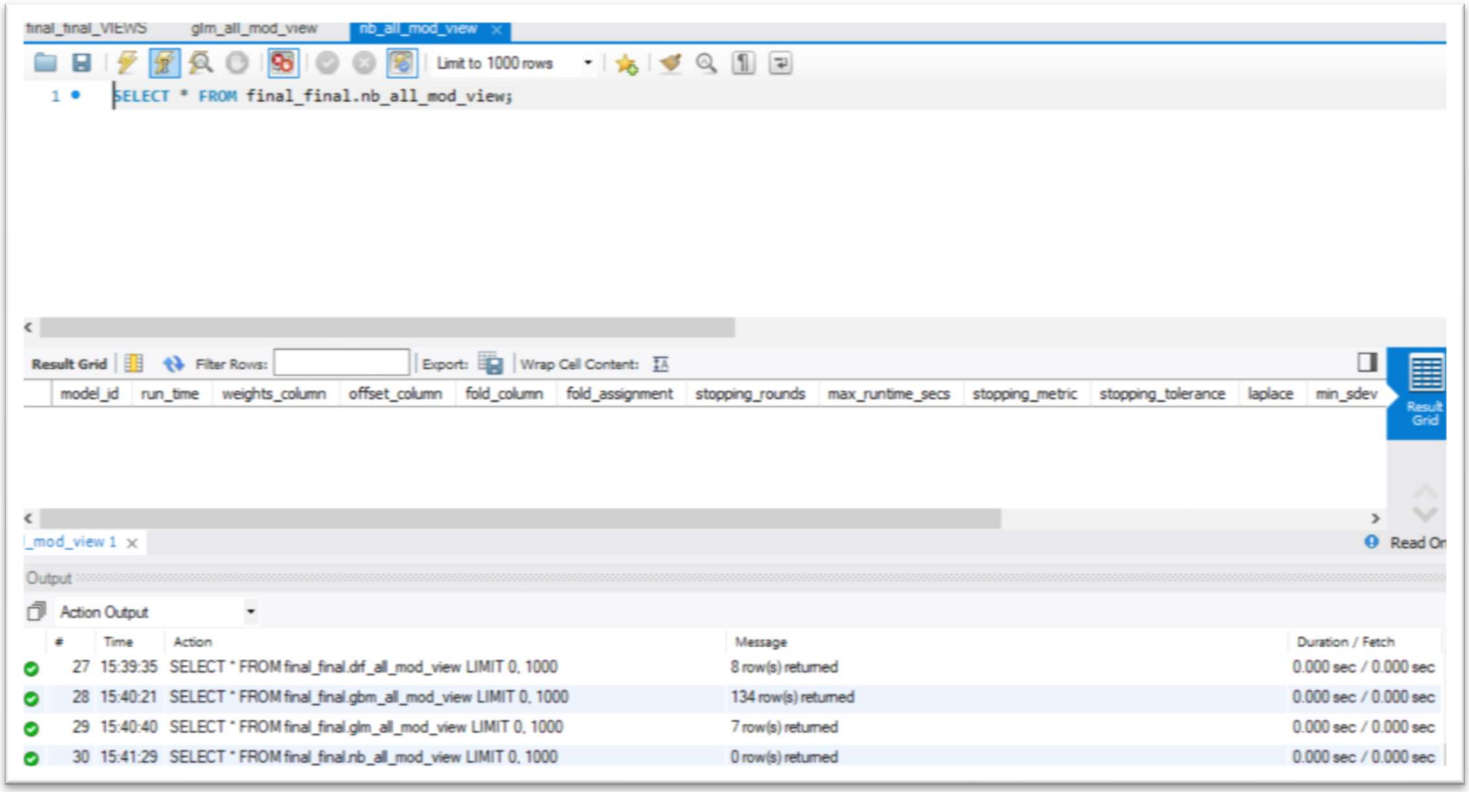


VIEW 4

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

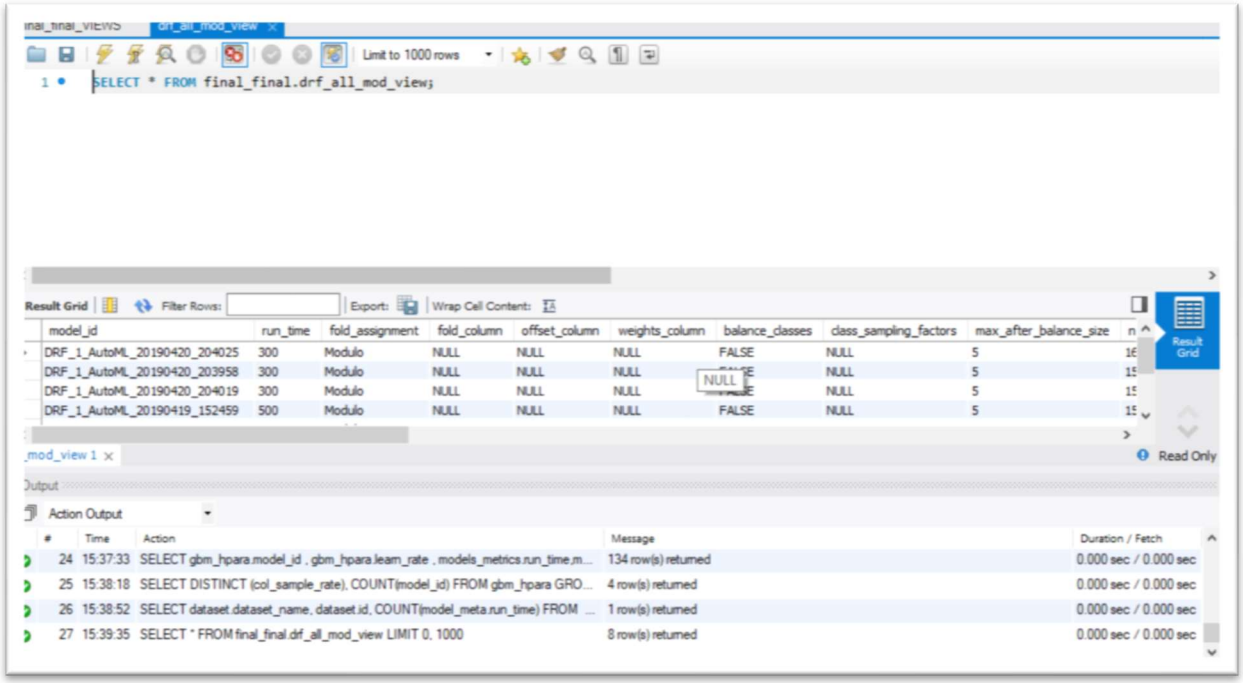


VIEW 5

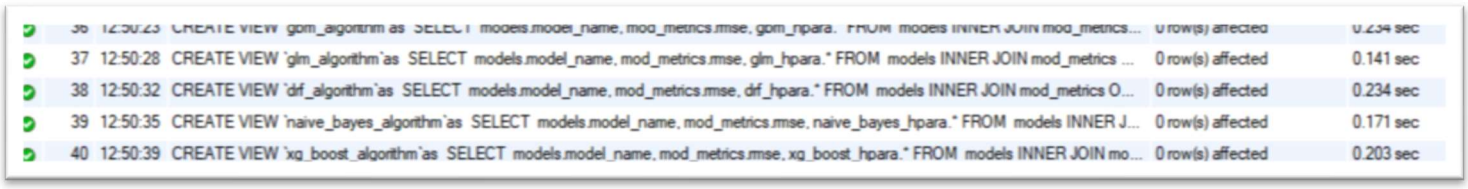
DRF VIEW INPUT

```
-- 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

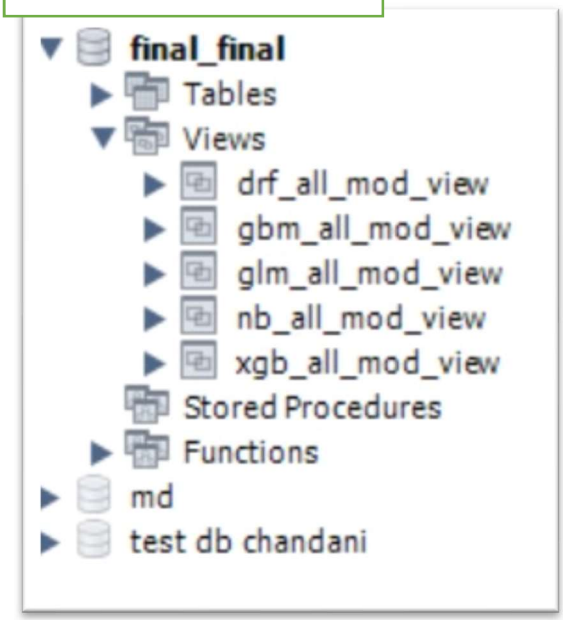


RUN TIME FOR ALL VIEWS.

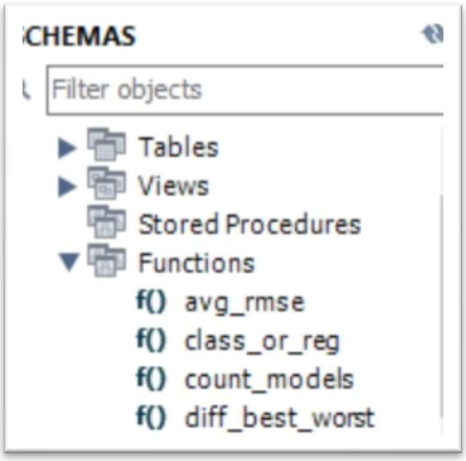


37	12:50:28	CREATE VIEW `glm_algorithm`as SELECT models.model_name, mod_metrics.rmse, glm_hpara.* FROM models INNER JOIN mod_metrics ...	0 row(s) affected	0.141 sec
38	12:50:32	CREATE VIEW `drf_algorithm`as SELECT models.model_name, mod_metrics.rmse, drf_hpara.* FROM models INNER JOIN mod_metrics O...	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 J...	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 mo...	0 row(s) affected	0.203 sec

ALL VIEWS

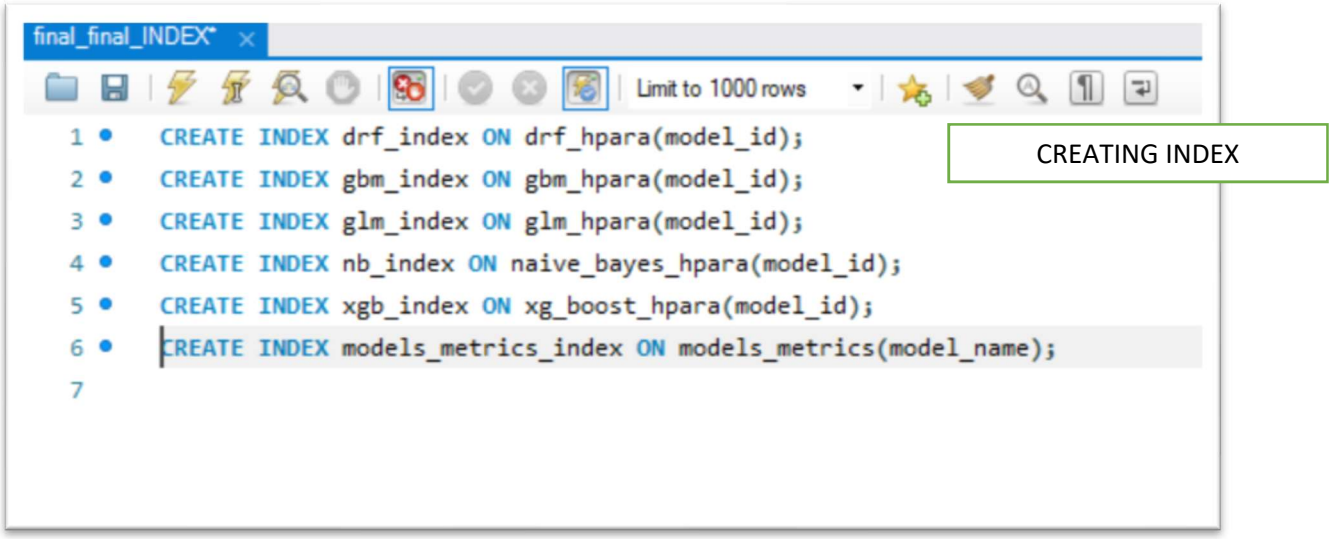


ALL FUNCTIONS

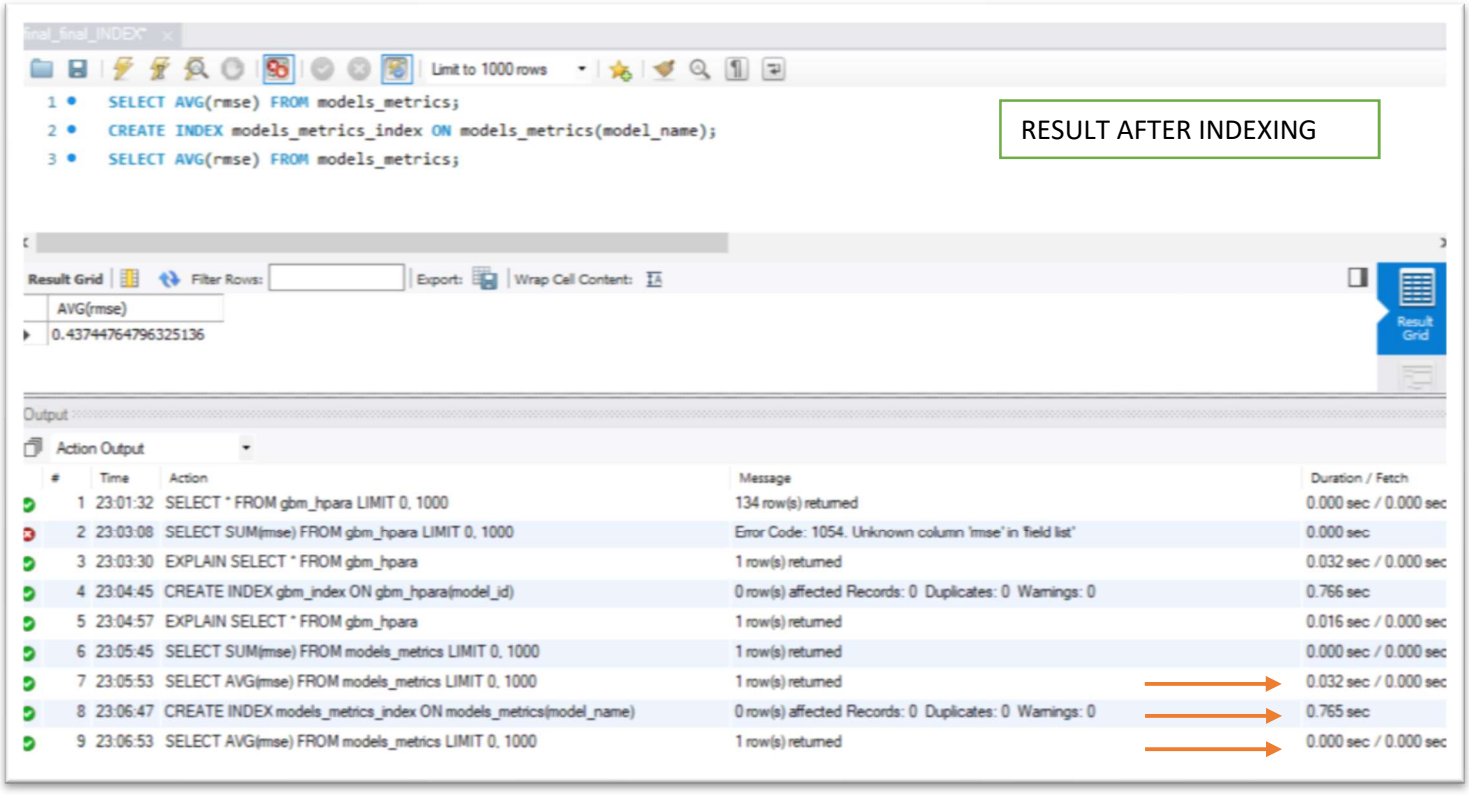


INDEXING

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.



Showing the example of how indexing improved the performance



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.

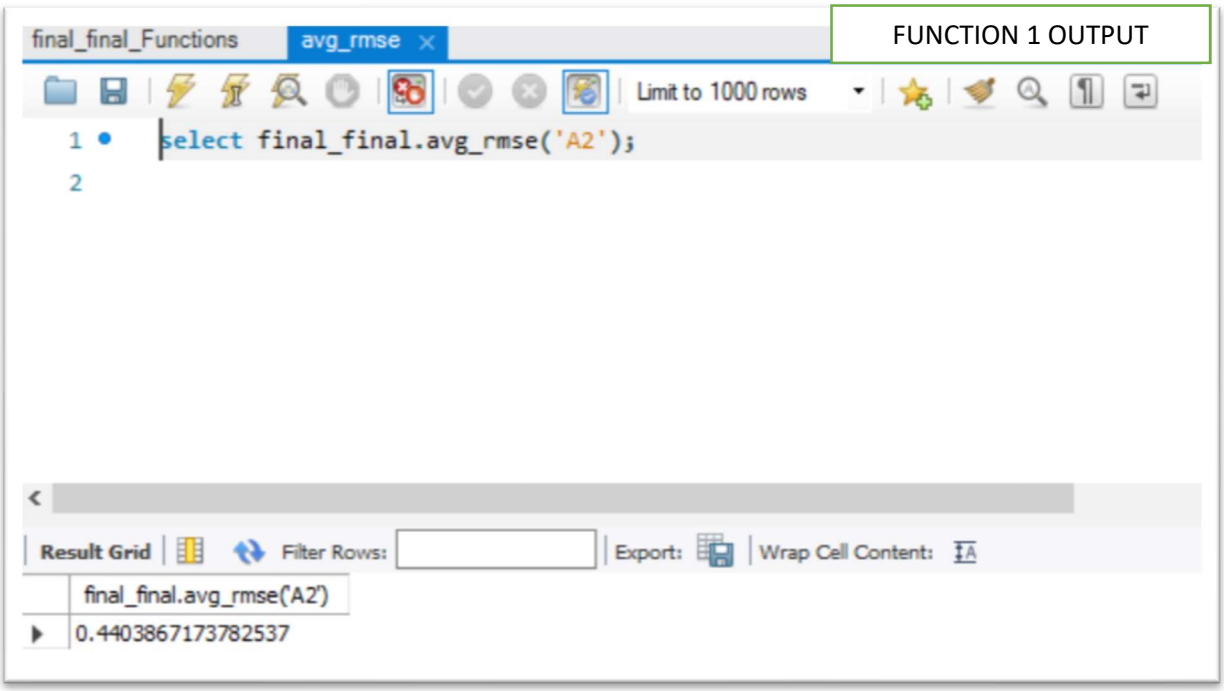
AVERAGE Function:

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

FUNCTION 1 INPUT

```
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 OUTPUT



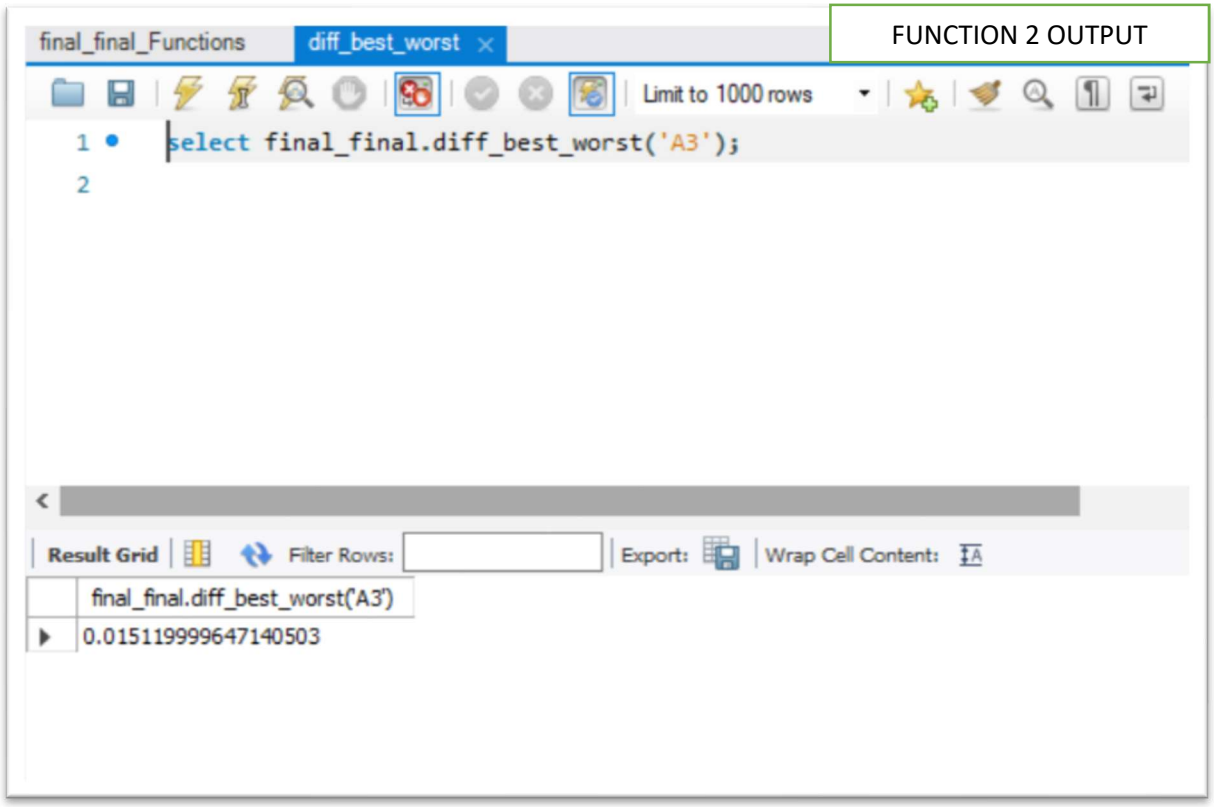
CHANGE IN RMSE:

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.

FUNCTION 2 INPUT

```
-- 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 OUTPUT



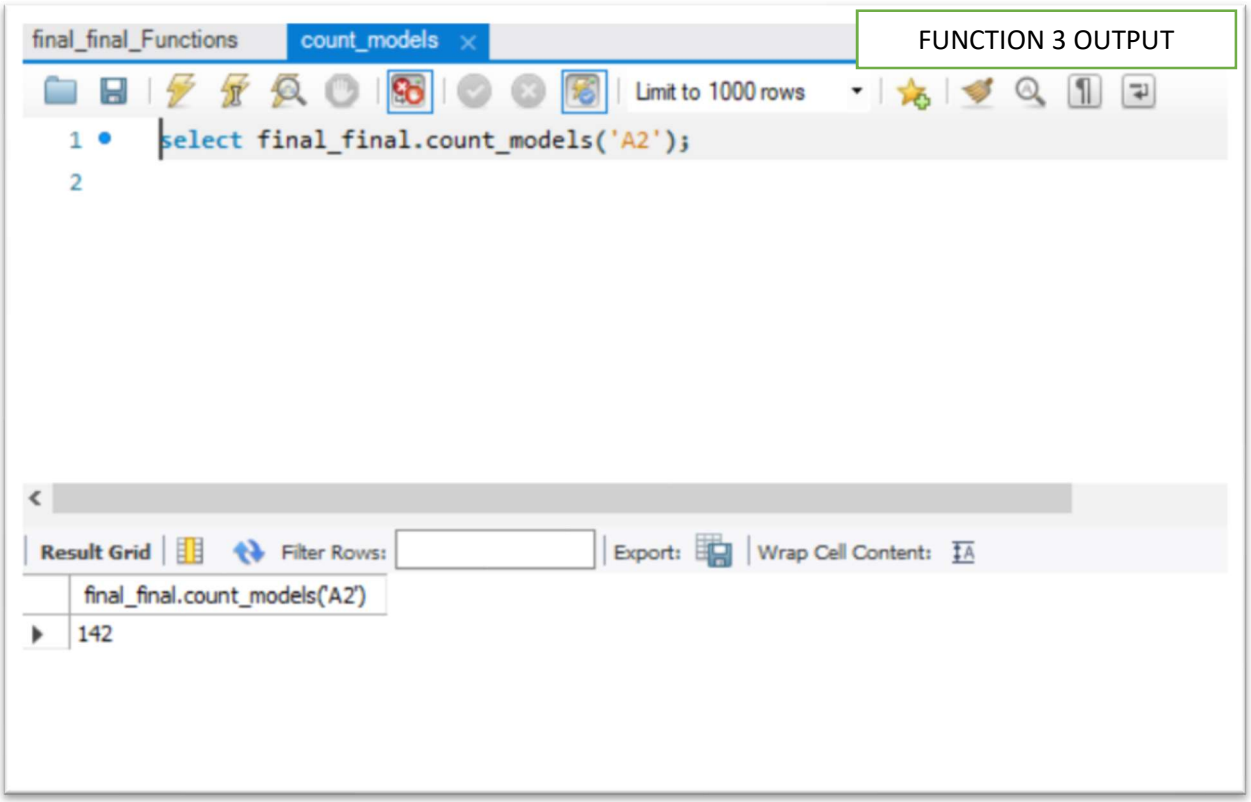
MODELS OF EACH ALGORITHM:

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

FUNCTION 3 INPUT

```
-- 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 OUTPUT



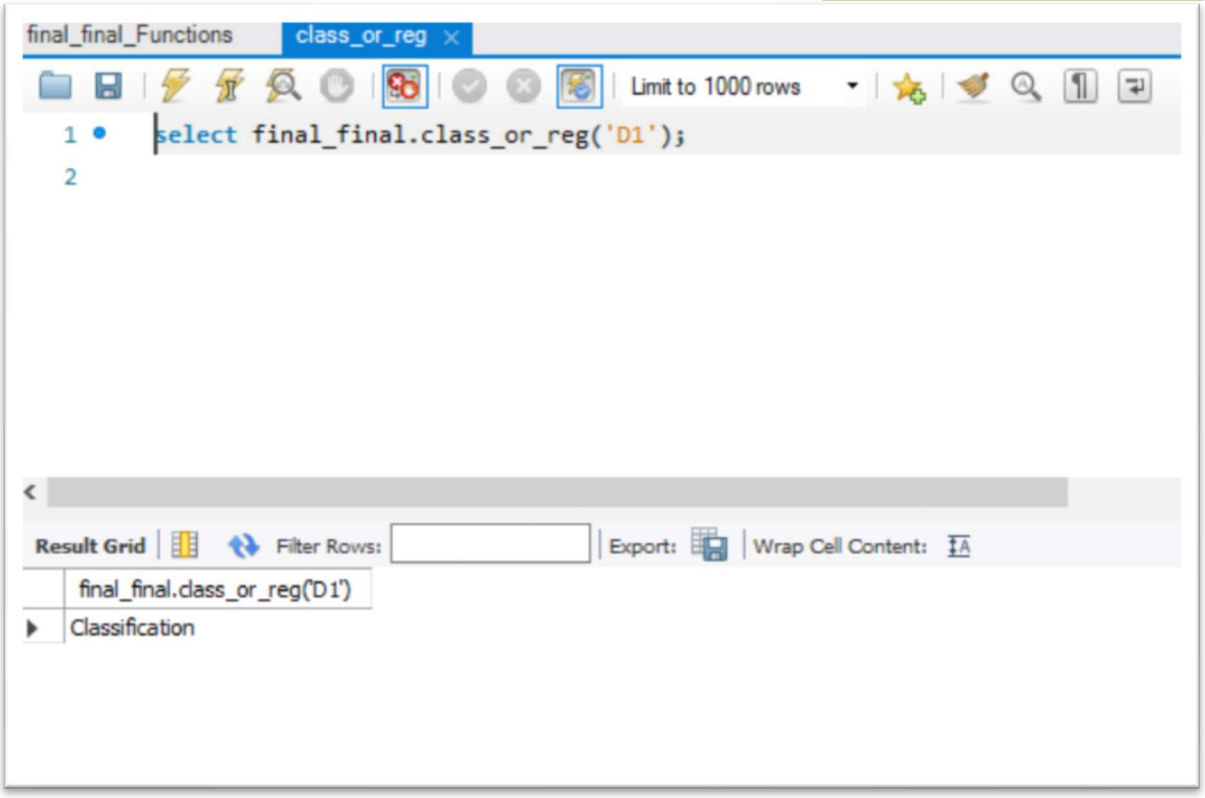
PROBLEM TYPE :

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



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
SELECT
gbm_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
gbm_hpara JOIN models_metrics ON
gbm_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 1 OUTPUT

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 / Fe
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 gbm_hpara.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 glm_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

Action Output

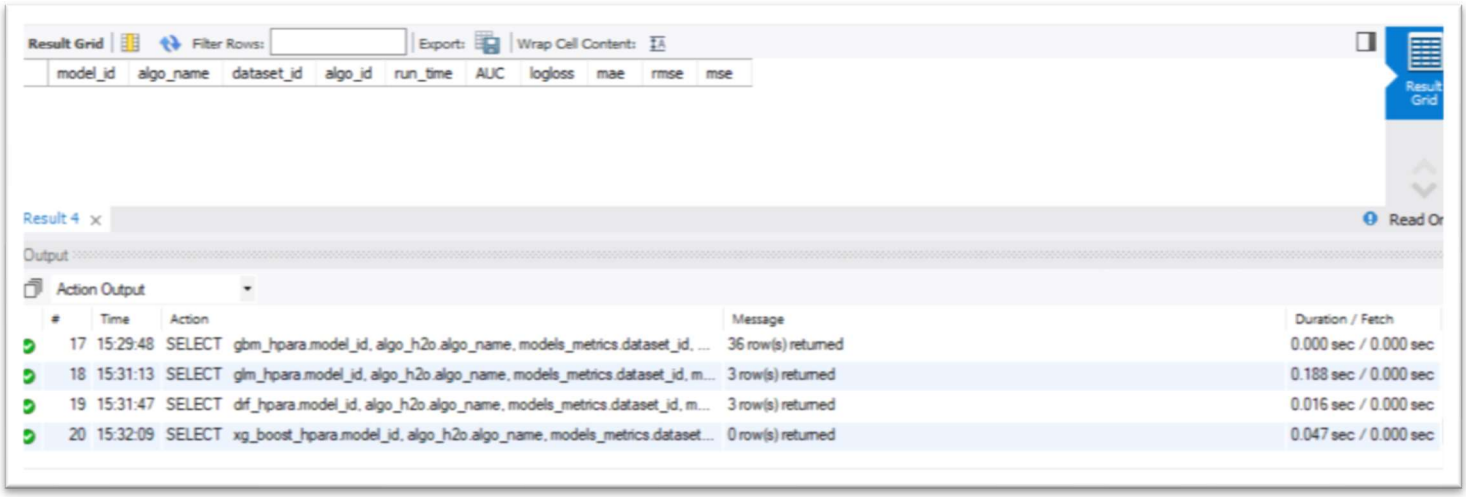
#	Time	Action	Message	Duration / Fetch
16	15:19:56	select final_final.diff_best_worst('A3') LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 s
17	15:29:48	SELECT gbm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, ...	36 row(s) returned	0.000 sec / 0.000 s
18	15:31:13	SELECT glm_hpara.model_id,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



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

model_id	algo_name	dataset_id	algo_id	run_time	AUC	logloss	mae	rmse	mse

Result 5

Output

Action Output

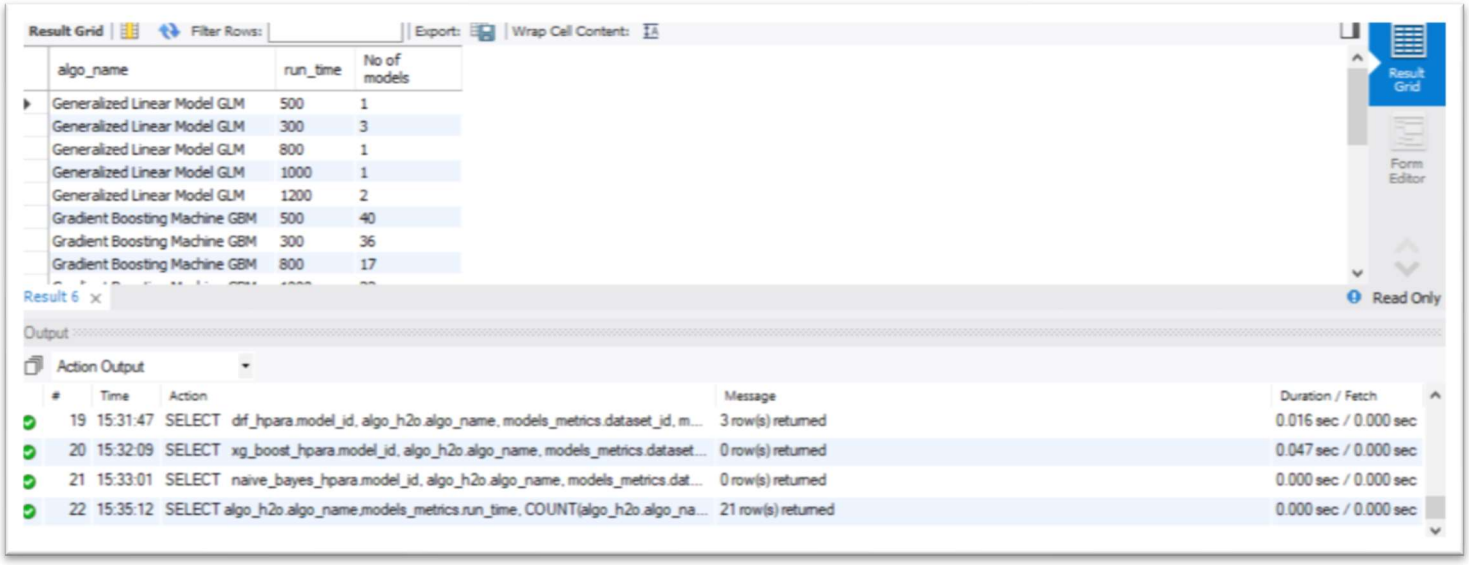
#	Time	Action	Message	Duration / Fetch
18	15:31:13	SELECT glm_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, m...	3 row(s) returned	0.188 sec / 0.000 sec
19	15:31:47	SELECT drf_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset_id, m...	3 row(s) returned	0.016 sec / 0.000 sec
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

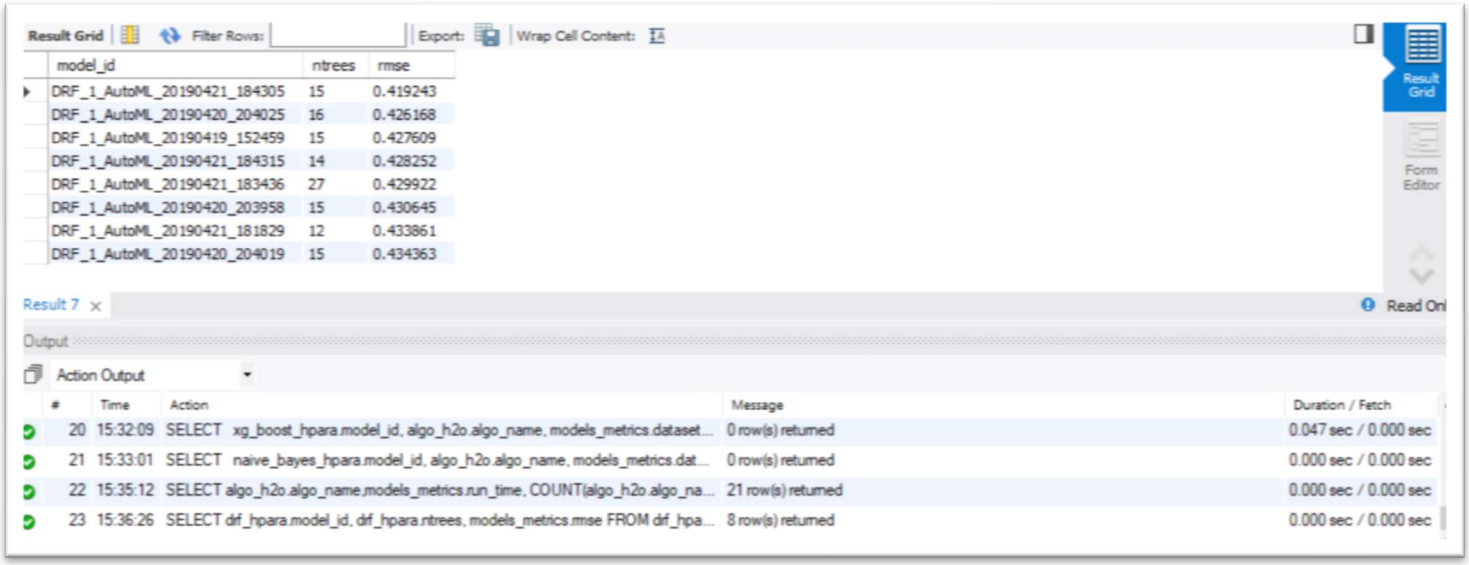


USE CASE 7: How many trees should I include in my DRF?

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 | Read Only

Output

Action Output

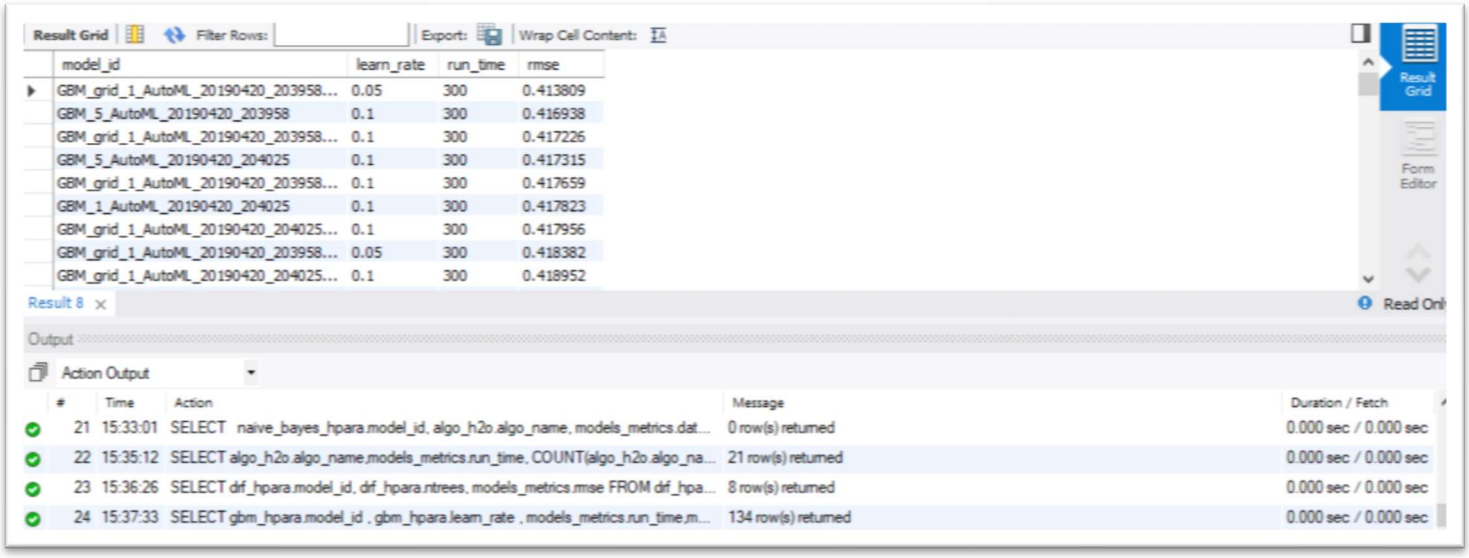
#	Time	Action	Message	Duration / Fetch
20	15:32:09	SELECT xg_boost_hpara.model_id, algo_h2o.algo_name, models_metrics.dataset...	0 row(s) returned	0.047 sec / 0.000 sec
21	15:33:01	SELECT naive_bayes_hpara.model_id, algo_h2o.algo_name, models_metrics.dat...	0 row(s) returned	0.000 sec / 0.000 sec
22	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
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



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

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

Result 8 | Read Only

Output

Action Output

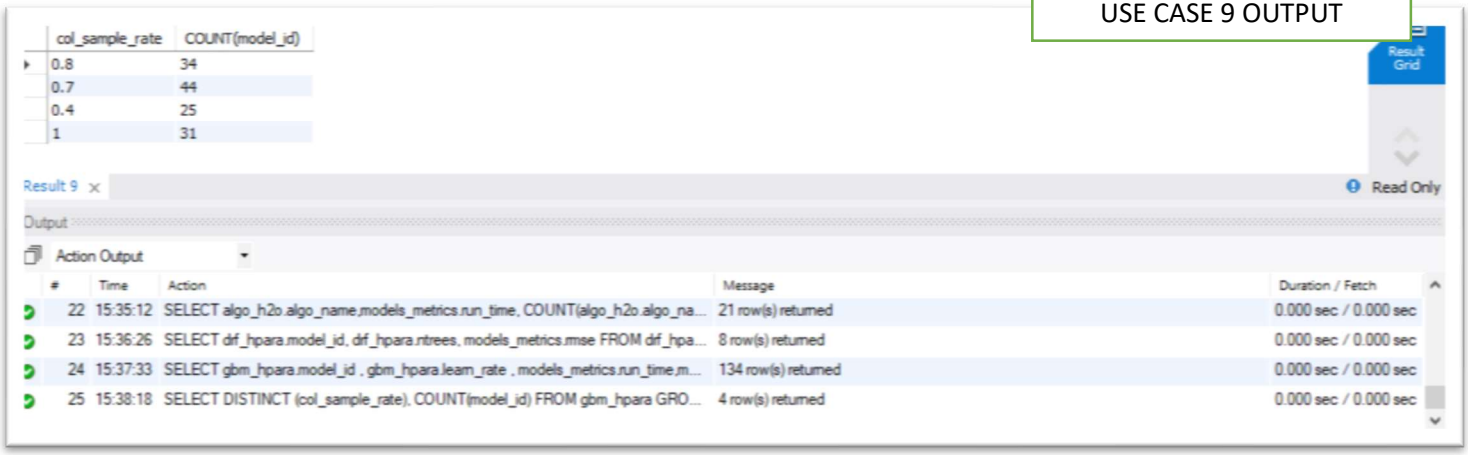
#	Time	Action	Message	Duration / Fetch
21	15:33:01	SELECT naive_bayes_hpara.model_id, algo_h2o.algo_name, models_metrics.dat...	0 row(s) returned	0.000 sec / 0.000 sec
22	15:35:12	SELECT algo_h2o.algo_name,models_metrics.run_time, COUNT(algo_h2o.algo_na...	21 row(s) returned	0.000 sec / 0.000 sec
23	15:36:26	SELECT drf_hpara.model_id, drf_hpara.ntrees, models_metrics.rmse FROM drf_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 valuse 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)
0.8	34
0.7	44
0.4	25
1	31

Result 9 | Read Only

Output

Action Output

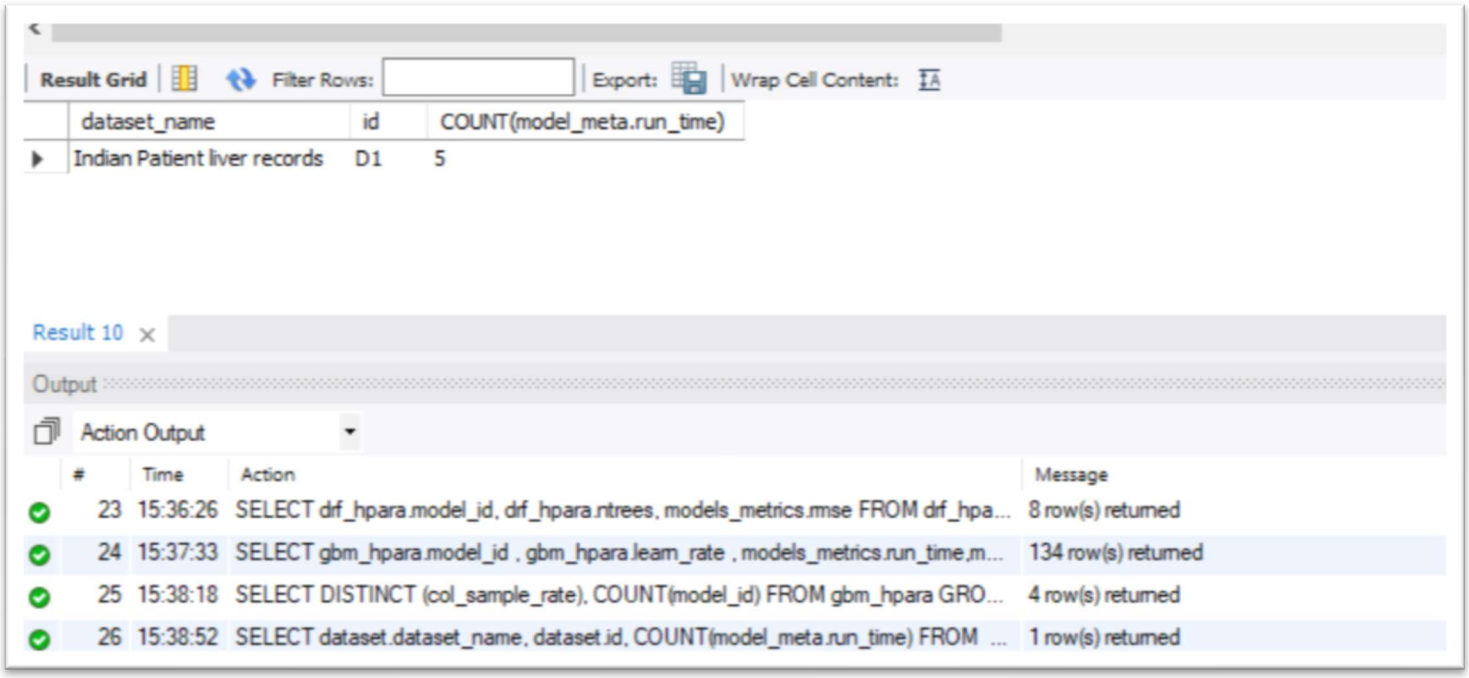
#	Time	Action	Message	Duration / Fetch
22	15:35:12	SELECT algo_h2o.algo_name,models_metrics.run_time, COUNT(algo_h2o.algo_na...	21 row(s) returned	0.000 sec / 0.000 sec
23	15:36:26	SELECT drf_hpara.model_id, drf_hpara.ntrees, models_metrics.rmse FROM drf_hpa...	8 row(s) returned	0.000 sec / 0.000 sec
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



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
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