Machine Learning using PostgreSQL

Learn predictive algorithms implementation using PostgreSQL & Apache MADlib

Presented By

Abbas Butt

Senior Principal

EnterpriseDB

March 2025

PostgresConf 2025

1. Introduction to Machine Learning (ML)

1.1. Traditional Algorithms vs ML Algorithms

Traditional algorithms are provided with some inputs, they perform some well defined operation on the input and provide one or more outputs. For example imagine an algorithm that implements the equation of a straight line.

$$y = m * x + c$$

In this example x is the input, y is the output whereas m & c are parameters, that can either be hard coded or configured according to situation.

Consider Java code implementing this particular example algorithm:

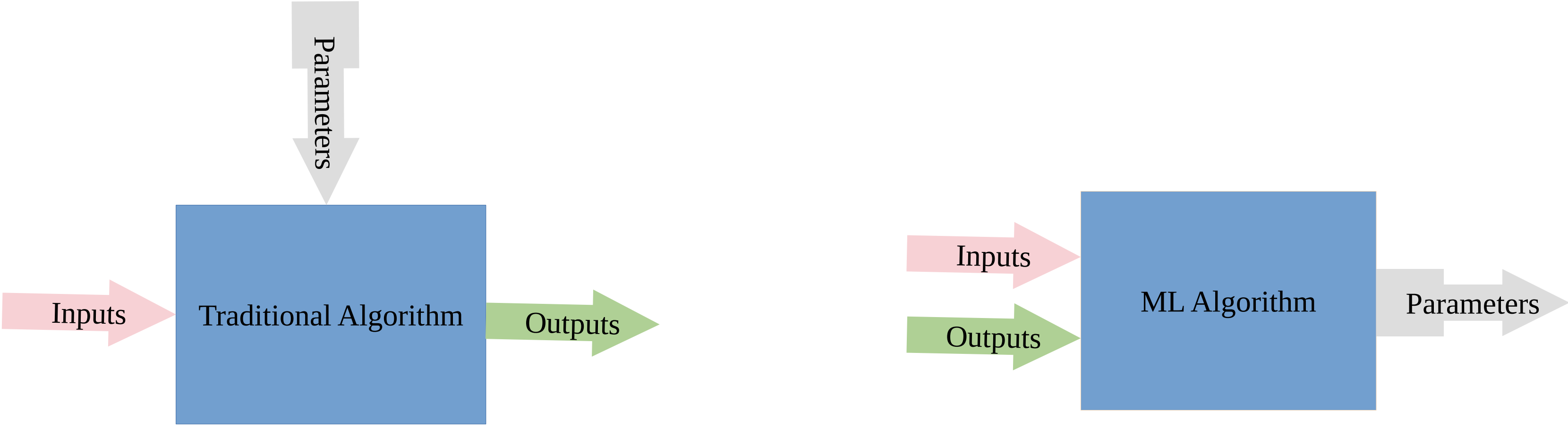
```
public class LinearEquation {
    private double m;
    private double c;

    public LinearEquation(double m, double c) {
        this.m = m;
        this.c = c;
    }

    public double calculateY(double x) {
        return (m * x) + c;
    }

    public static void main(String[] args) {
        LinearEquation equation = new LinearEquation(2.5, 3.0);
        double x = 4.0;
        double y = equation.calculateY(x);

        System.out.println("For x = " + x + ", the value of y = " + y);
    }
}
```

If we repeatedly call this function with different inputs, we will get a table for x & y.

In contrast a machine learning algorithm is provided with data, with both inputs and outputs, and the algorithm is supposed to come up with parameters of the system that could have been used to generate the provided data. The data can have some noise for example if it was gathered as a result of some survey or if there was some human error.

Extending the example of a Linear System, the machine learning algorithm will be provided with a table of input and output and it will come up with the best estimate of the parameters m & c. There are ways to measure the accuracy of the prediction. If in case the data was not generated by a linear system in the first place then the measures of the accuracy of the results will be outside of the desired limits. Also there are ways of data analysis from which a data scientist can conclude what type of a system could have generated data at hand.

By using the predicted parameters m & c one can provide an estimated value of y for an unseen value of x.



1.2. Why implement ML Algorithms using PostgreSQL?

There are many motivations behind implementing machine learning algorithms using PostgreSQL.

- PostgreSQL is where all the data resides. If we implement ML models within the database itself, we do not need to export data to other environments. The model can access the data directly.
- PostgreSQL provides many powerful features for preparing and pre-processing large datasets.
- In-database machine learning is now possible because of powerful extensions like Apache Madlib.

2. Linear Regression Analysis

Linear regression analysis provides a method of exploring whether there is a linear relationship between a dependent random variable 'y' and an independent random variable 'x.' It tries to find a line that most closely fits the data available for 'x' and 'y' using the least squares method.

Consider the straight line equation:

$$y = m * x + c$$

where m is the slope and c is the intercept.

With this background we will explore functions provided by PostgreSQL for linear regression analysis.

2.1. Setting up the database server

Install PostgreSQL 15.10 using the following steps on Ubuntu 22.04

Check python version:

```
which python3
     /usr/bin/python3
python3 --version
     Python 3.10.12
```

Import the repository signing key:

```
sudo apt install curl ca-certificates
sudo install -d /usr/share/postgresql-common/pgdg
sudo curl -o /usr/share/postgresql-common/pgdg/apt.postgresql.org.asc --fail
https://www.postgresql.org/media/keys/ACCC4CF8.asc
```

Create the repository configuration file:

```
sudo sh -c 'echo "deb [signed-by=/usr/share/postgresql-common/pgdg/apt.postgresql.org.asc]
https://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
```

Update the package list:

```
sudo apt update
```

Install the PostgreSQL 15 packages

```
sudo apt -y install postgresql-client-15
sudo apt -y install postgresql-15
sudo apt -y install postgresql-server-dev-15
sudo apt -y install postgresql-plpython3-15
```

Restart PostgreSQL services

```
sudo systemctl restart postgresql@15-main.service
```

Check status of PostgreSQL services

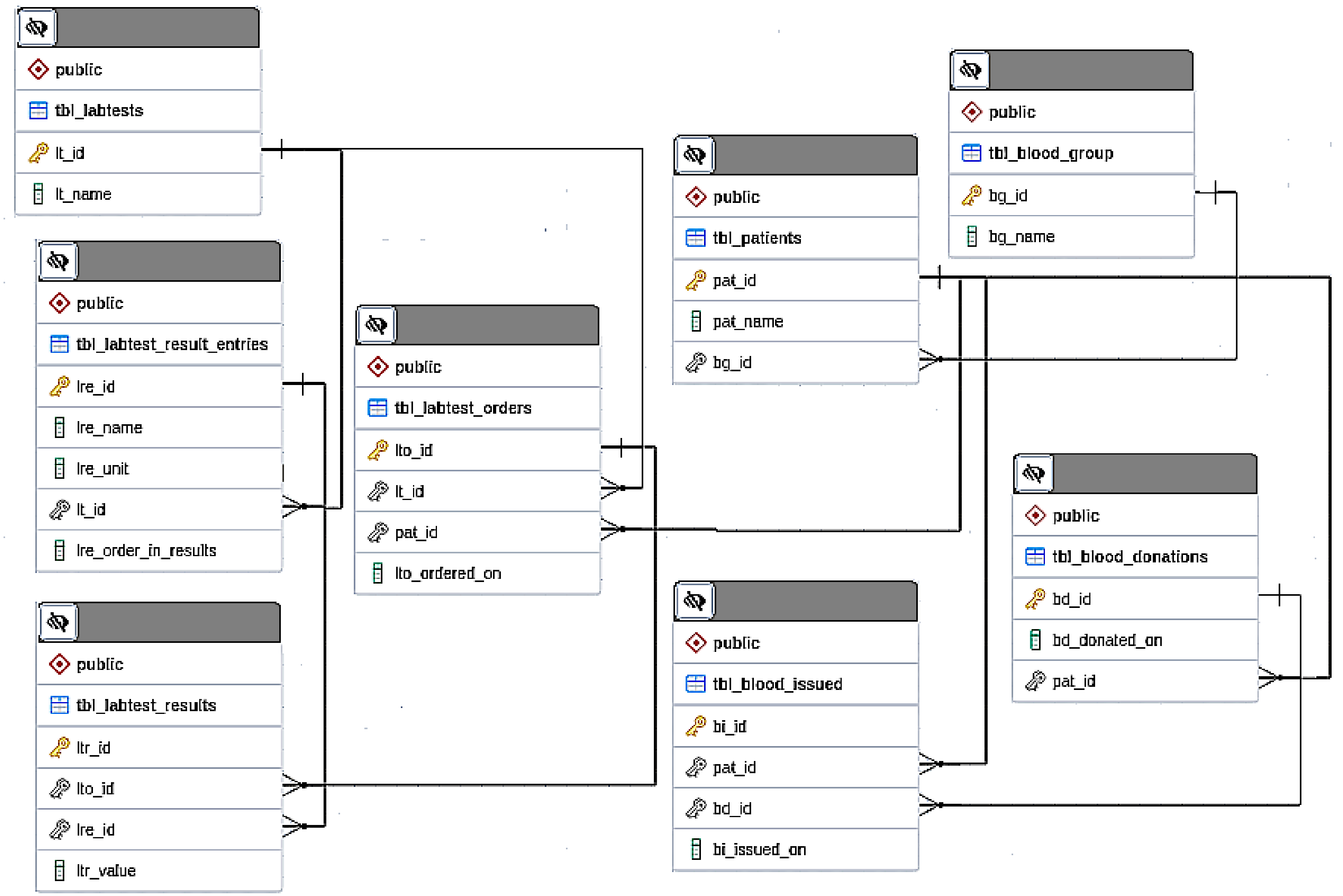
```
sudo systemctl status postgresql@15-main.service
postgresql@15-main.service - PostgreSQL Cluster 15-main
 Loaded: loaded (/lib/systemd/system/postgresql@.service; enabled-runtime; vendor preset: enabled)
  Active: active (running) since Tue 2024-12-31 01:59:20 +04; 20h ago
 Main PID: 14340 (postgres)
   Tasks: 7 (limit: 9382)
  Memory: 61.4M
     CPU: 13.335s
     CGroup: /system.slice/system-postgresql.slice/postgresql@15-main.service
             ├─14340 /usr/lib/postgresql/15/bin/postgres -D /var/lib/postgresql/15/main -c config_file=/etc/postgresql/15/main/postgresql.conf
             ├─14341 "postgres: 15/main: checkpointer " "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" """" "" "" "">
             ├─14342 "postgres: 15/main: background writer " "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "">
             ├─14344 "postgres: 15/main: walwriter  "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "">
             ├─14345 "postgres: 15/main: autovacuum launcher " "" "" "" "" "" "" ""  "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "">
             ├─14346 "postgres: 15/main: logical replication launcher " "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" """ "" "" "" "">
             └─69641 "postgres: 15/main: postgres bbdb 127.0.0.1(34592) idle" "" "" "" "" "" "" "" "" ""  "" "" "" "" "" "" "" "" "" "" "">

pgconf systemd[1]: Starting PostgreSQL Cluster 15-main...
pgconf systemd[1]: Started PostgreSQL Cluster 15-main.
```

2.1.1.Configure the database server

```
postgres=# CREATE DATABASE bbdb;
postgres=# ALTER USER postgres WITH PASSWORD 'abc123';
postgres=# SELECT * FROM pg_available_extensions() WHERE name LIKE 'p%';
        name         | default_version |                                comment
--------------------+-----------------+-----------------------------------------------------------------------
 pg_visibility      | 1.2             | examine the visibility map (VM) and page-level visibility info
 postgres_fdw       | 1.1             | foreign-data wrapper for remote PostgreSQL servers
 pg_surgery         | 1.0             | extension to perform surgery on a damaged relation
 pg_prewarm         | 1.2             | prewarm relation data
 pgrowlocks         | 1.2             | show row-level locking information
 pg_walinspect      | 1.0             | functions to inspect contents of PostgreSQL Write-Ahead Log
 pg_buffercache     | 1.3             | examine the shared buffer cache
 pgstattuple        | 1.5             | show tuple-level statistics
 pgcrypto           | 1.3             | cryptographic functions
 pg_trgm            | 1.6             | text similarity measurement and index searching based on trigrams
 plpgsql            | 1.0             | PL/pgSQL procedural language
 plpython3u         | 1.0             | PL/Python3U untrusted procedural language
 pg_freespacemap    | 1.2             | examine the free space map (FSM)
 pg_stat_statements | 1.10            | track planning and execution statistics of all SQL statements executed
 pageinspect        | 1.11            | inspect the contents of database pages at a low level
(15 rows)
postgres=# select unnest(string_to_array(version(), ',')) as version;
                              version
----------------------------------------------------------------------
 PostgreSQL 15.10 (Ubuntu 15.10-1.pgdg22.04+1) on x86_64-pc-linux-gnu
  compiled by gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
  64-bit
(3 rows)
```

2.2. Setting up the dataset

We will use a sample blood bank database with Entity Relationship Diagram as follows:



The ERD contains a lot of tables but we will focus on five tables only.

2.2.1.tbl_patients

Create the patients table. Patient names have to be unique.

```
CREATE TABLE tbl_patients  (
 pat_id SERIAL PRIMARY KEY,
 pat_name VARCHAR(255) NOT NULL UNIQUE );
```

2.2.2.tbl_labtests

Create table to store the tests that the blood bank will perform before clearing the blood fit for use. Test names must be unique.

```
CREATE TABLE tbl_labtests  (
 lt_id SERIAL PRIMARY KEY,
 lt_name VARCHAR(255) NOT NULL UNIQUE  );
```

2.2.3.tbl_labtest_orders

Create table to store a lab test orders. A lab test order connects a lab test to a patient.

```
CREATE TABLE tbl_labtest_orders  (
 lto_id SERIAL PRIMARY KEY,
 lt_id INT REFERENCES tbl_labtests(lt_id),
 pat_id INT REFERENCES tbl_patients(pat_id),
 lto_ordered_on TIMESTAMP WITH TIME ZONE NOT NULL  );
```

2.2.4.tbl_labtest_result_entries

Create table to store the names and units of the entries in the result of a particular lab test. The column “lre_order_in_results” specifies the order in which the entries should appear in the lab test result. Composite unique constraint is added because more than one result entry of a test cannot have the same order.

```
CREATE TABLE tbl_labtest_result_entries (
 lre_id SERIAL PRIMARY KEY,
 lre_name VARCHAR(255) NOT NULL UNIQUE,
```

```
 lre_unit VARCHAR(255) NOT NULL,
 lt_id INT REFERENCES tbl_labtests(lt_id),
 lre_order_in_results INT NOT NULL,
 UNIQUE (lt_id, lre_order_in_results)  );
```

2.2.5.tbl_labtest_results

Create table to store lab test results. A lab test result assigns a value to all the expected entries in the result of a lab test. Composite unique constraint is added to make sure that value for each result entry is added only once against a certain order.

```
CREATE TABLE tbl_labtest_results  (
 ltr_id SERIAL PRIMARY KEY,
 lto_id INT REFERENCES tbl_labtest_orders(lto_id),
 lre_id INT REFERENCES tbl_labtest_result_entries(lre_id),
 ltr_value double precision,
 UNIQUE (lto_id, lre_id)  );
```

Insert sample data in the tables

Insert blood tests.

```
INSERT INTO tbl_labtests (lt_name) VALUES('Blood Sugar Random');
INSERT INTO tbl_labtests (lt_name) VALUES('Blood Complete Picture');
```

Insert patients.

```
\i /path/to/file/pat.sql
```

Insert test result entries along with units.

```
\i /path/to/file/lab_res_entry.sql
```

Insert “Blood Sugar Random” test orders. The SQL file inserts 3,729 orders.

```
\i /path/to/file/bsr_orders.sql
```

Insert “Blood Complete Picture” test orders. The SQL file inserts 18,282 orders.

```
\i /path/to/file/bcp_orders.sql
```

Insert “Blood Sugar Random” test results for each order. This SQL file inserts 3,729 result entries in “tbl_labtest_results”.

```
\i /path/to/file/bsr_results.sql
```

Insert “Blood Complete Picture” test results for each order. This SQL file inserts 196,040 result entries in “tbl_labtest_results”. Most tests have 11 result entries, however some test results have 10 or 9 entries.

```
\i /path/to/file/bcp_results.sql
```

2.3. Builtin functions for Statistics

Lets compute how many “Blood Sugar Random” and “Blood Complete Picture” tests were ordered till date.

```
bbdb=# SELECT t.lt_name, count(r.lt_id)
       FROM tbl_labtest_orders r, tbl_labtests t
       WHERE r.lt_id = t.lt_id GROUP BY t.lt_name;
        lt_name         | count
------------------------+-------
 Blood Complete Picture | 18282
 Blood Sugar Random     |  3729
(2 rows)
```

Lets calculate the minimum, average and maximum value of each of the result entries for all the lab test orders.

```
bbdb=# SELECT e.lre_name, min(r.ltr_value),
       round( cast(avg(r.ltr_value) as numeric) , 2) as avg, max(r.ltr_value)
       FROM tbl_labtest_results r, tbl_labtest_result_entries e
       WHERE r.lre_id = e.lre_id GROUP BY e.lre_name;
                    lre_name                     | min  |    avg    |   max
-------------------------------------------------+------+-----------+---------
 Glucose-random                                  |   13 |    168.70 |    1000
 Hemoglobin                                      | 1.01 |     43.11 |  290000
 Lymphocytes                                     |    0 |     28.70 |     545
 Mean Corpuscular Hemoglobin Concentration (MCHC)|  3.3 |     48.01 |  260000
 Mean Corpuscular Hemoglobin (MCH)               |    2 |     27.89 |    2834
 Mean Corpuscular Volume (MCV)                   |    5 |     80.81 |    9638
 Mixed Cell Percentage                           |    0 |      7.66 |     545
 Neutrophils                                     |    0 |     63.86 |     100
 Packed Cell Volume (PCV)/Hematocrit (HCT)       |  2.5 |     34.07 |      60
 Platelet Count                                  | 1.01 | 244917.29 | 3465000
 RBC Count                                       | 0.28 |      4.24 |     8.1
 WBC Count                                       |  2.3 |   8837.04 |  291200
(12 rows)
```

2.3.1.Variance and Standard Deviation

Standard Deviation is the square root of the variance. Standard Deviation describes average distance of each value in the data from the mean. Both standard deviation and variance are measures of variability but their units are different. Standard deviation has the same units as the original values, whereas the variance is expressed in squared units.

If the data has been obtained form every member of the population under study, then we can get the exact value of the population variance using the following formula:

$$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N}$$

where

$$\sigma^2 = PopulationVariance$$
$$x_i = i^{th} valueofdata$$
$$\mu = PopulationMean$$
$$N = Totalnumberofvalues$$

If the data has been obtained form only a sample of the population under study, then sample variance is used to estimate the population variance. Sample variance can be calculated using the following formula:

$$S^2 = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N - 1}$$

where

$$S^2 = Sample\,Variance$$
$$x_i = i^{th} valueofdata$$
$$\bar{x} = SampleMean$$
$$N = Totalnumberofvalues$$

PostgreSQL provides functions to calculate sample and population variance and standard deviation.

var_pop, var_samp

This function returns the population or sample variance of the values in the column name provided in the argument. For example:

Lets calculate the population and sample variance of each of the result entries for all the lab test orders.

```
bbdb=# SELECT e.lre_name,
       round( cast(var_pop(r.ltr_value) as numeric) , 2) as var_pop,
       round( cast(var_samp(r.ltr_value) as numeric) , 2) as var_samp
       FROM tbl_labtest_results r, tbl_labtest_result_entries e
       WHERE r.lre_id = e.lre_id GROUP BY e.lre_name;
                    lre_name                     |    var_pop     |    var_samp
-------------------------------------------------+----------------+----------------
 Glucose-random                                  |       13766.92 |       13770.62
 Hemoglobin                                      |     7877525.07 |     7877956.06
 Lymphocytes                                     |         151.51 |         151.52
 Mean Corpuscular Hemoglobin Concentration (MCHC) |     3697935.99 |     3698138.35
 Mean Corpuscular Hemoglobin (MCH)               |         458.98 |         459.00
 Mean Corpuscular Volume (MCV)                   |        6728.27 |        6728.63
 Mixed Cell Percentage                           |          37.43 |          37.44
 Neutrophils                                     |         145.26 |         145.27
 Packed Cell Volume (PCV)/Hematocrit (HCT)       |          33.16 |          33.16
 Platelet Count                                  | 12770035715.71 | 12770734333.56
 RBC Count                                       |           0.45 |           0.45
 WBC Count                                       |    34473705.23 |    34475590.99
(12 rows)
```

stddev_pop, stddev_samp

This function returns the population or sample standard deviation of the values in the column name provided in the argument. For example:

Lets calculate the population and sample standard deviation of each of the result entries for all the lab test orders.

```
bbdb=# SELECT e.lre_name,
       round( cast(stddev_pop(r.ltr_value) as numeric) , 2) as stddev_pop,
       round( cast(stddev_samp(r.ltr_value) as numeric) , 2) as stddev_samp
       FROM tbl_labtest_results r, tbl_labtest_result_entries e
       WHERE r.lre_id = e.lre_id GROUP BY e.lre_name;
                     lre_name                      | stddev_pop | stddev_samp
---------------------------------------------------+------------+-------------
 Glucose-random                                    |     117.33 |      117.35
 Hemoglobin                                        |    2806.69 |     2806.77
 Lymphocytes                                       |      12.31 |       12.31
 Mean Corpuscular Hemoglobin Concentration (MCHC) |    1923.00 |     1923.05
 Mean Corpuscular Hemoglobin (MCH)                 |      21.42 |       21.42
 Mean Corpuscular Volume (MCV)                     |      82.03 |       82.03
 Mixed Cell Percentage                             |       6.12 |        6.12
 Neutrophils                                       |      12.05 |       12.05
 Packed Cell Volume (PCV)/Hematocrit (HCT)         |       5.76 |        5.76
 Platelet Count                                    |  113004.58 |   113007.67
 RBC Count                                         |       0.67 |        0.67
 WBC Count                                         |    5871.43 |     5871.59
(12 rows)
```

2.3.2.Covariance and Correlation Coefficient

Covariance of two random variables is a measure of their combined variability. A positive covariance indicates a linear relationship, a negative covariance indicates an inverse relationship.

Population and sample covariance can be calculated using the following formulas:

$$\sigma_{xy} = \frac{\sum_{i=1}^{N} (x_i - \mu_x) * (y_i - \mu_y)}{N}$$

where

$$\sigma_{xy} = Populati\ o\ nCovariance$$
$$x_i = i^{th}\ valueoffirstrandomvariable$$
$$\mu_x = PopulationMeanoffirstrandomvariable$$
$$y_i = i^{th}\ valueofscondrandomvariable$$
$$\mu_y = PopulationMean\ o\ fsecondrandomvariable$$
$$N = Totalnumberofvalues$$

$$S_{xy} = \frac{\sum_{i=1}^{N} (x_i - \bar{x}) * (y_i - \bar{y})}{N-1}$$

where

$$S_{xy} = SampleCovariance$$
$$x_i = i^{th}\ valueoffirstrandomvariable$$
$$\bar{x} = SampleMeanoffirstrandomvariable$$
$$y_i = i^{th}\ valueofscondrandomvariable$$
$$\bar{y} = SampleMeanofsecondrandomvariable$$
$$N = Totalnumberofvalues$$

Covariance can be any positive or negative number and it will have units of x units times the y units. In order to normalize we can use correlation coefficient which is unit less and lies between -1 and +1. -1 meaning a perfect inverse relationship and +1 indicating a perfect linear relationship between the two random variables.

Correlation coefficient can be computed using the following formulas:

$$\rho = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

where

$$\rho = PopulationCorrelationCoefficient$$
$$\sigma_{xy} = PopulationC\,o\,variance$$
$$\sigma_x = PopulationStandardDeviationforfirstrandomvariable$$
$$\sigma_y = PopulationStandardDeviationforsecondrandomvariable$$

$$r = \frac{S_{xy}}{S_x S_y}$$

where

$$r = SampleCorrelationCoefficient$$
$$S_{xy} = SampleCovariance$$
$$S_x = SampleStandardDeviationforfirstrandomvariable$$
$$S_y = SampleStandardDeviationfor\ secondrandomvariable$$

PostgreSQL provides functions for calculating population and sample covariance and sample correlation coefficient. Please note that function for population correlation coefficient is not provided by PostgreSQL.

2.4. Role of crosstab extension

covar_pop

The covariance between two random variables measures the degree to which two variables change together. Specifically, it indicates whether an increase in one variable tends to be associated with an increase or decrease in the other variable. In simpler terms:

Positive covariance: When one variable increases, the other tends to increase as well, and vice versa.

Negative covariance: When one variable increases, the other tends to decrease.

Zero covariance: There is no consistent pattern of change between the two variables.

This function returns the population covariance of the values in the two column names provided in the arguments. For example, suppose we want to explore relationship between the counts of “Lymphocytes” and “Neutrophils” in the results of the blood tests data.

In order to calculate population covariance we first need the values of the counts of our result entries in columns rather than in rows. For this purpose PostgreSQL provides crosstab function.

Knowing that the lre_id for “HCT” is 2 and for “RBC” is 7, would simplify the queries we are going to write next.

A simple select from the “tbl_labtest_results” displays results like this

```
bbdb=# SELECT lto_id, lre_id, ltr_value FROM tbl_labtest_results
       WHERE lre_id = 2 or lre_id = 7 order by 1,2;
 lto_id | lre_id | ltr_value
--------+--------+-----------
   3730 |      2 |      40.7
   3730 |      7 |      4.86
   3731 |      2 |      36.8
   3731 |      7 |       3.4
   3732 |      2 |      39.3
   3732 |      7 |      4.35
   3733 |      2 |      50.5
   3733 |      7 |      5.16
.....
.....
```

We cannot apply the covar_pop function unless we have the results in the following format:

```
 order_id |   HCT   |   RBC
----------+--------+---------
     3730 |   40.7 |   4.86
     3731 |   36.8 |   3.4
     3732 |   39.3 |   4.35
     3733 |   50.5 |   5.16
....
```

If we can get the results in this format we will simply pass second and third column to the covar_pop function.

PostgreSQL provides crosstab function for exactly this purpose. The function is however provided as an extension and works only if the "tablefunc" is available in the database. The crosstab function takes a SQL query text as input. The query passed to the crosstab function must satisfy the following requirements.

- The query must return exactly three columns. The columns are normally called "row_name," "category," and "value".
- They query must order the results by the first two columns, i.e., it must include ORDER BY 1,2.
- The three columns for the crosstab query should be chosen in such a manner that
  - The "row_name" column must divide the result set into groups.
  - Each group should then have one "value" for each "category".
- For our case "lto_id" divides the results into groups, each group having one value for each "lre_id." Hence we will select "lto_id" as "row_name," "lre_id" as "category," and "ltr_value" as "value."
- Next we need to calculate 'N,' i.e., the number of categories are in each group. For our case we have two categories (2 and 7) in each lab test order.
- The output of the crosstab query will have N+1 columns. In our case the output of the crosstab query will have three columns.
- The column data types in the crosstab query result will be as follows:
  - The first column will be the same as the first column of the select query passed to the corsstab function. In our case it will be int.
  - The rest of the columns will have the same data type as the last column of the select query passed to the corsstab function. In our case it will be double precision.
- The column names in the crosstab query result can be arbitrary.

The query in our case would therefore be:

```
bbdb=# CREATE EXTENSION tablefunc;
bbdb=# SELECT * FROM crosstab('SELECT lto_id, lre_id, ltr_value
                              FROM tbl_labtest_results
                              WHERE lre_id = 2 or lre_id = 7 ORDER BY 1,2')
        ct(order_id int,
           HCT double precision,
           RBC double precision);
 order_id |  hct   |  rbc
----------+-------+-------
     3730 |  40.7 |  4.86
     3731 |  36.8 |   3.4
     3732 |  39.3 |  4.35
     3733 |  50.5 |  5.16
..........
```

Now that we have the columns nicely placed we can create some views:

```
bbdb=# CREATE VIEW view_for_regr1 AS
       SELECT * FROM crosstab('SELECT lto_id, lre_id, ltr_value FROM tbl_labtest_results
                               WHERE (lre_id = 2 or lre_id = 7) ORDER BY 1,2')
       ct(order_id int, HCT double precision, RBC double precision);

bbdb=# SELECT * FROM view_for_regr1;
 order_id |  hct   |  rbc
----------+-------+-------
     3730 |  40.7 |  4.86
     3731 |  36.8 |   3.4
     3732 |  39.3 |  4.35
     3733 |  50.5 |  5.16
..........
```

```
bbdb=# CREATE VIEW view_for_regr2 AS SELECT HCT, RBC FROM view_for_regr1 ORDER BY HCT;

bbdb=# SELECT * FROM view_for_regr2;
  hct  |  rbc
-------+-------
   2.5 |     5
   2.5 |  3.76
     3 |  3.95
   3.2 |   4.1
   3.3 |  3.88
     4 |  4.57
   4.3 |  4.43
   4.3 |  4.73
     5 |   0.7
..........
bbdb=# CREATE VIEW view_for_regr3 AS SELECT round(hct::numeric, 0) as rhct, rbc from view_for_regr2;

bbdb=# SELECT * FROM view_for_regr3;
 rhct |  rbc
------+-------
    3 |     5
    3 |  3.76
    3 |  3.95
    3 |   4.1
    3 |  3.88
    4 |  4.57
    4 |  4.43
    4 |  4.73
..........
```

```
bbdb=# CREATE VIEW view_for_regr4 AS SELECT rhct, avg(rbc) as arbc FROM view_for_regr3 group by rhct ORDER BY 1;
```

This is the most crucial step. We need one averaged Y value for one value of X. This is also called **bucketing**.

```
bbdb=# SELECT * FROM view_for_regr4;
 rhct |        arbc
------+--------------------
    3 |              4.138
    4 |  4.576666666666667
    5 | 0.6499999999999999
    6 |               0.82
    7 |               1.16
    8 |             1.1575
..........
bbdb=# CREATE VIEW view_for_regr5 AS SELECT * FROM view_for_regr4 WHERE rhct > 6 ORDER BY 1;
```

We can neglect small values of HCT .

```
bbdb=# SELECT * FROM view_for_regr5;
 rhct |        arbc
------+--------------------
    7 |               1.16
    8 |             1.1575
    9 |               1.24
   10 |               1.61
   11 | 1.6455555555555557
   12 |               1.24
..........
```

We will use view view_for_regr5 in the rest of our queries.

```
bbdb=# SELECT covar_pop(arbc, rhct) FROM view_for_regr5;
     covar_pop
--------------------
 22.387142975536683
(1 row)
```

A positive values for covariance is an indication of a linear relationship between the two random variables, i.e., HCT and RBC. Note that the function expects Y as the first and X as second parameter. In our case Y is averaged RBC and X is binned HCT.

covar_samp

This function returns the sample covariance of the values in the two column names provided in the arguments. For example, lets try to find the sample covariance between the counts of “HCT” and “RBC” in the results of the blood tests data.

```
bbdb=# SELECT covar_samp(arbc, rhct) FROM view_for_regr5;
     covar_samp
--------------------
 22.817664955835465
(1 row)
```

Note that the function expects Y as the first and X as second parameter. In our case Y is averaged RBC and X is binned HCT.

corr

This function returns the sample correlation coefficient of the values in the two column names provided in the arguments. For Example lets try to find the correlation coefficient between the counts of “HCT” and “RBC” in the results of the blood tests data.

```
bbdb=# SELECT corr(arbc, rhct) FROM view_for_regr5;
        corr
--------------------
 0.9852653040328714
(1 row)
```

A value 0.98 which is very close of 1 shows a linear relationship between HCT and RBC. Note that the function expects Y as the first and X as second parameter. In our case Y is averaged RBC and X is binned HCT.

2.5. Slope & Intercept

regr_slope

This function returns the slope of the least squares linear regression equation determined using the values in the two column names provided in the arguments. For example, lets try to find the slope of the linear regression equation describing the relationship between “HCT” and “RBC” in the results of the blood tests data.

```
bbdb=# SELECT regr_slope(arbc, rhct) FROM view_for_regr5;
     regr_slope
---------------------
 0.09526458250890382
(1 row)
```

Note that the function expects Y as the first and X as second parameter. In our case Y is averaged RBC and X is binned HCT.

regr_intercept

This function returns the y-intercept of the least squares linear regression equation determined using the values in the two column names provided in the arguments. For example, lets try to find the y-intercept of the linear regression equation describing the relationship between “HCT” and “RBC” in the results of the blood tests data.

```
bbdb=# SELECT regr_intercept(arbc, rhct) FROM view_for_regr5;
   regr_intercept
--------------------
 0.8451717554243574
(1 row)
```

Note that the function expects Y as the first and X as second parameter. In our case Y is averaged RBC and X is binned HCT.

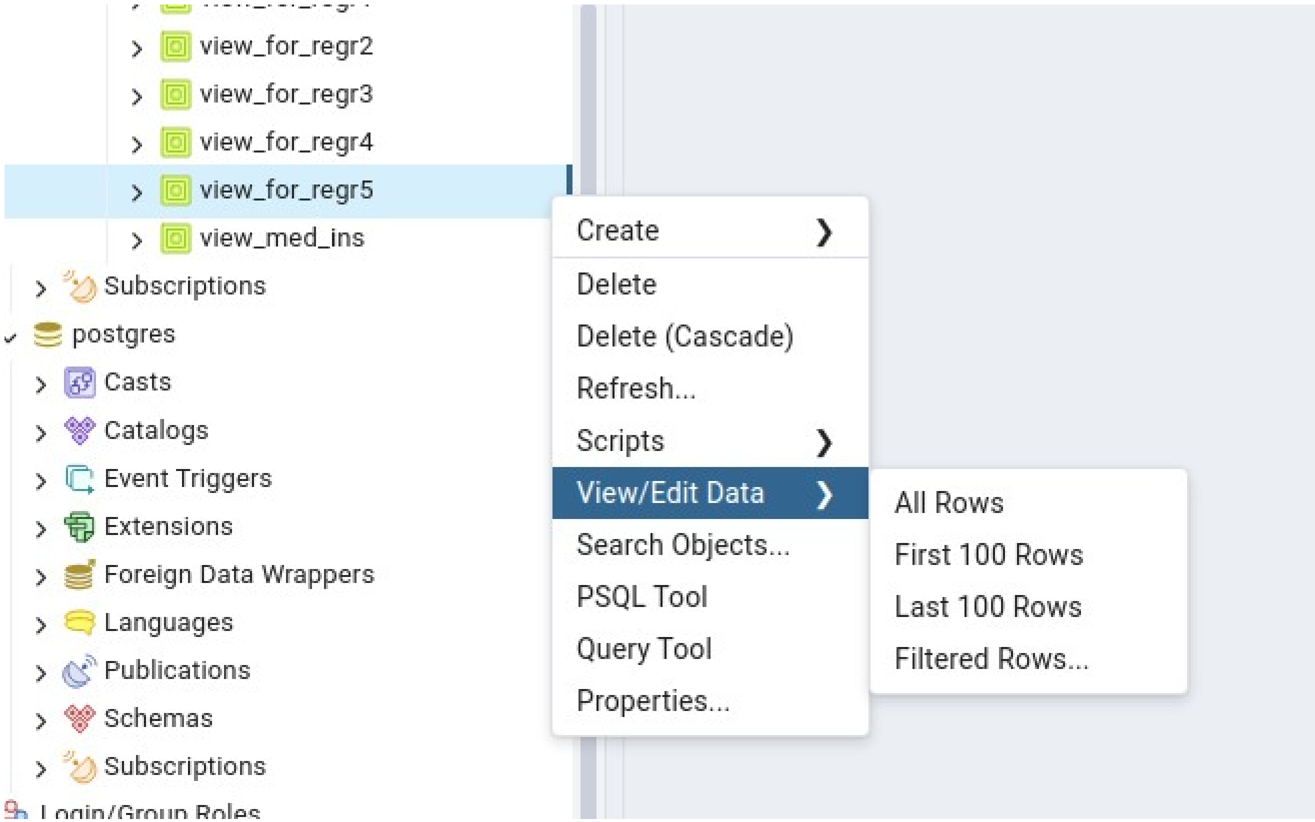
2.6. Role of pgAdmin4

Lets first install pgAdmin4.

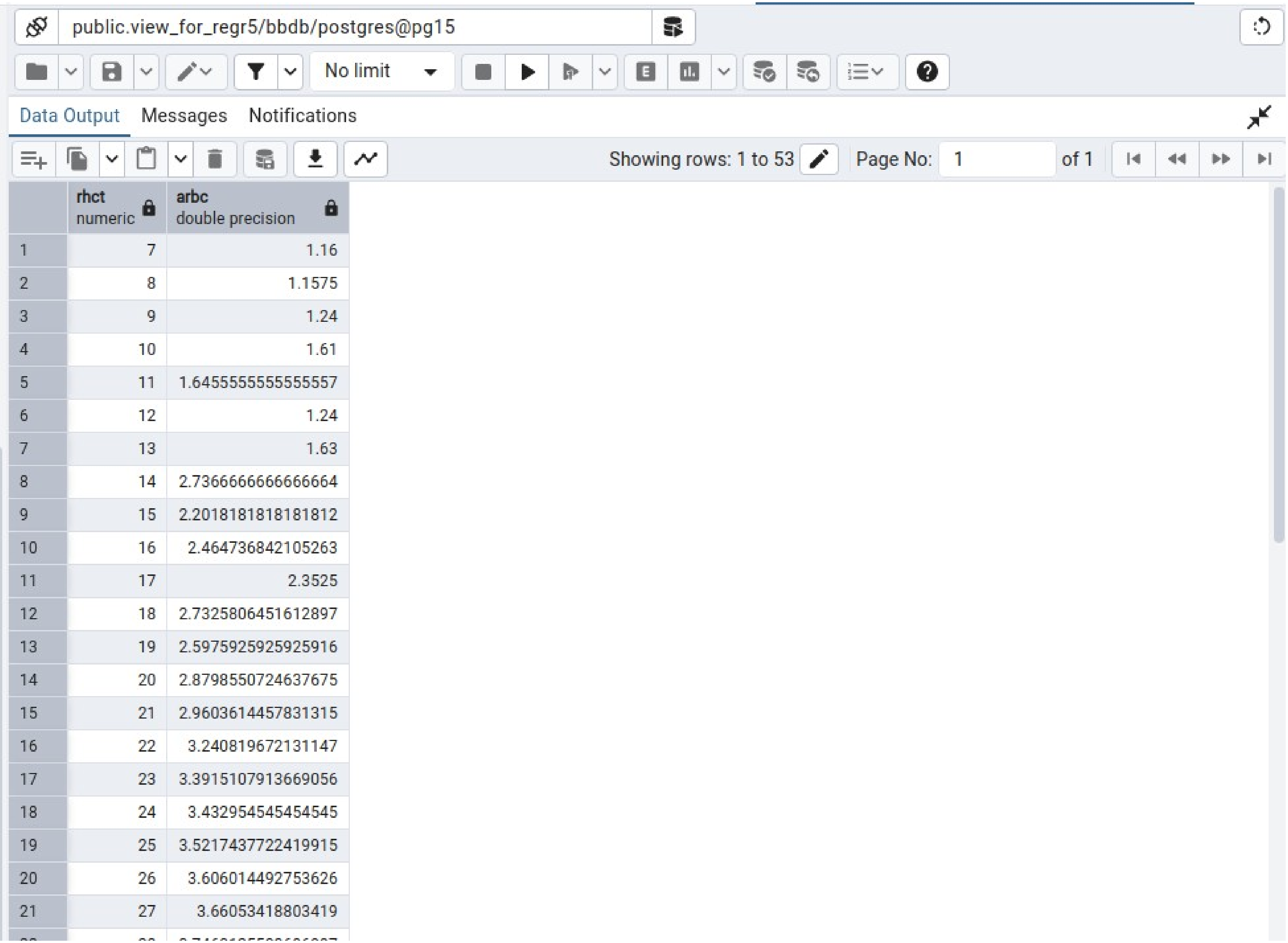
```
curl -fsS https://www.pgadmin.org/static/packages_pgadmin_org.pub |
         sudo gpg --dearmor -o /usr/share/keyrings/packages-pgadmin-org.gpg
sudo sh -c 'echo "deb [signed-by=/usr/share/keyrings/packages-pgadmin-org.gpg]
         https://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/$(lsb_release -cs) pgadmin4 main" >
         /etc/apt/sources.list.d/pgadmin4.list && apt update'
sudo apt install pgadmin4-desktop
```

pgAdmin provides a way to visualize the data as follows:

Select view_for_regr5 from the list and choose View → All Rows

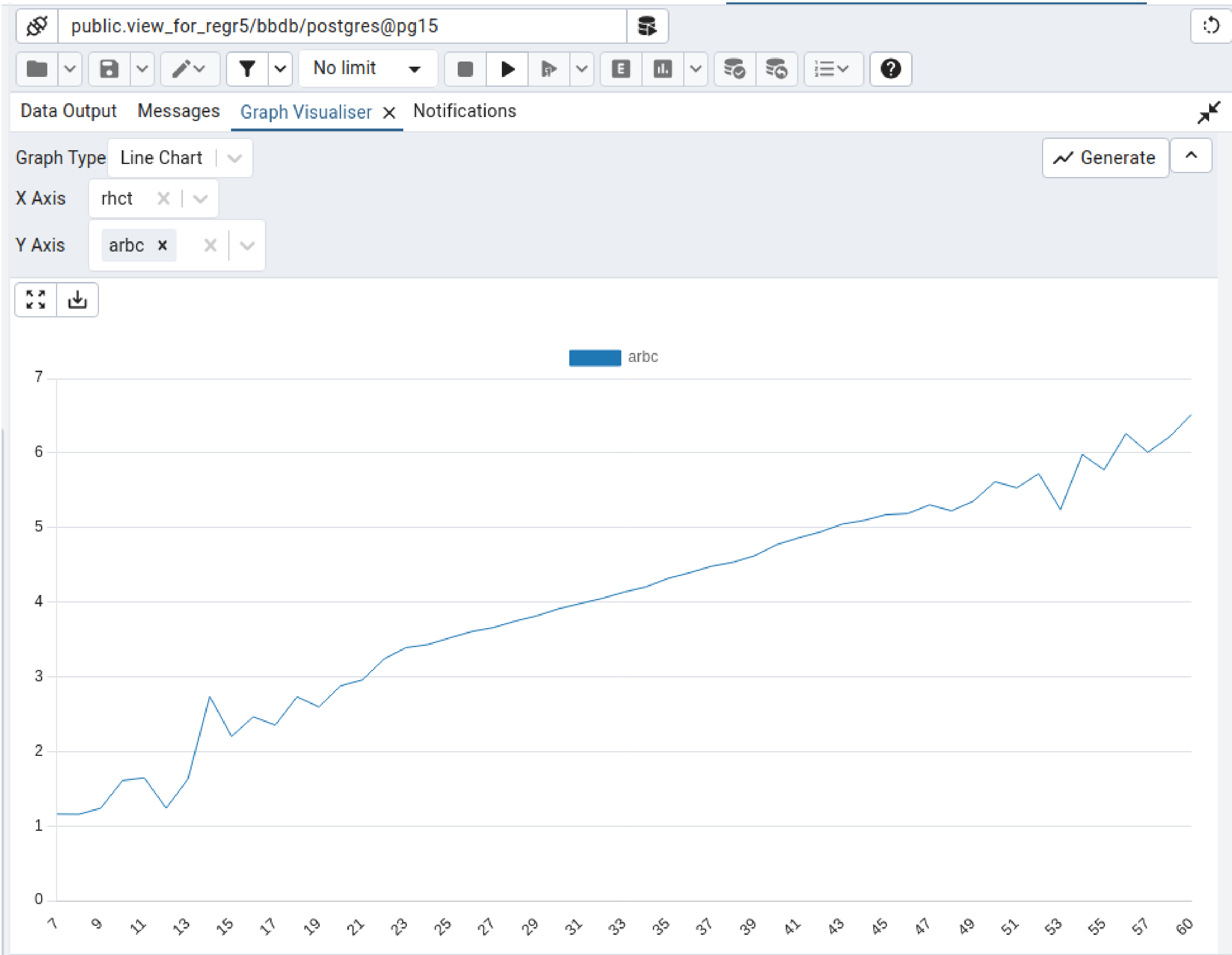


It will show all the rows in the view:



	rhct numeric	arbc double precision
1	7	1.16
2	8	1.1575
3	9	1.24
4	10	1.61
5	11	1.6455555555555557
6	12	1.24
7	13	1.63
8	14	2.7366666666666664
9	15	2.2018181818181812
10	16	2.464736842105263
11	17	2.3525
12	18	2.7325806451612897
13	19	2.5975925925925916
14	20	2.8798550724637675
15	21	2.9603614457831315
16	22	3.240819672131147
17	23	3.3915107913669056
18	24	3.432954545454545
19	25	3.5217437722419915
20	26	3.606014492753626
21	27	3.66053418803419

From this view select "Graph Visualizer". Select the X (rhct) and Y columns (arbc). It will show a graph like this:



Looking at the graph of the dependent variable (arbc) with respect to independent variable (rhct) show that the values calculated by the built in functions provided by PostgreSQL are correct. However we can measure the correctness of the model by using the notion of a L1 and L2 loss.

2.7. Model Accuracy

Loss is a measure of the difference between the predicted and the actual value.

Wait a minute, if we had the actual value, why did we do the prediction anyway?

To measure the loss.

2.7.1. Types of loss

Loss Type	Definition	Formula
L1 Loss	The sum of the absolute values of the difference b/w the predicted values and the actual values.	$L_1 = \sum_{i=1}^{N} \|(y_i - \bar{y})\|$
Mean Absolute Error (MAE)	The average of L1 losses across a set of examples.	L_1 / N
L2 Loss	The sum of the squared difference between the predicted values and the actual values.	$L_2 = \sum_{i=1}^{N} (y_i - \bar{y})^2$
Mean Squared Error (MSE)	The average of L2 losses across a set of examples.	L_2 / N

To calculate L2 loss PostgreSQL provides regr_syy function.

regr_syy

This function returns the sum of squares of the first of the two column names provided in the arguments.

The actual values of the dependent variable are already stored in the table. Lets create a view that provides y and ybar as two columns.

```
bbdb=# CREATE VIEW view_for_regr6 as SELECT arbc as y, (0.1 * rhct + 0.85) as ybar from view_for_regr5;

bbdb=# SELECT regr_syy(y, ybar) FROM view_for_regr6;
      regr_syy
--------------------
 116.43931239094383
(1 row)

bbdb=# SELECT regr_syy(y, ybar)/( SELECT count(*) FROM view_for_regr6) AS MSE FROM view_for_regr6;
        mse
--------------------
 2.1969681583196947
(1 row)

bbdb=# SELECT var_samp(y) AS var_y FROM view_for_regr6;
       var_y
-------------------
 2.239217545979689
(1 row)
```

The MSE and the variance of y are comparable. This means that the model is moderately useful. It cannot be termed as a good model.

2.8. Generalizing Statistical Analysis

In the previous sections we concluded that HCT and RBC are related according to the model:

HCT = 0.1 * RBC + 0.85

However the analysis was performed only on a small subset of data. Is this analysis true for the sample data provided or is this relationship generalizable?

There are two methods of assessing whether the analysis we performed on a subset of data is generalizable or not.

2.8.1. Bootstrap confidence interval for Slope

- Bootstrapping involves resampling the dataset (for example 1000 times) with replacement to create multiple new datasets.
- Regression analysis is performed on each sample, slope is calculated and results are gathered.
- The variance and standard deviation of the resulting slopes is calculated assuming a normal distribution.
- If 90% or 95% of the times the calculated slope is within the desired interval, then it is concluded that the results are generalizable.

2.8.2. Randomization Hypothesis Test for Slope

- Define Null Hypothesis Ho that there is no relationship between x & y and Alternative Hypothesis Ha that there is a relationship between x & y.
- Compute the slope
- Randomly permute (shuffle) the values of y while keeping x fixed at least 1000 times.
- Recalculate the Slope for Each Permuted Dataset
- Compute the p value using

  p = (Number of times shuffled slope >= Observed slope) / Total Shuffles

- If p-value is small (typically < 0.05), reject Ho means that the slope is significant.

The key idea behind shuffling y in a randomized hypothesis test for slope is to break the relationship between x and y under the assumption that no real association exists (the null hypothesis, Ho). In a real dataset, if x and y are truly related, we expect the slope of the regression line to be nonzero.

However, under Ho (no relationship between x and y), the values of y should be random with respect to x. So, by shuffling y and recalculating the slope multiple times, we simulate what the slope would look like if there were no actual relationship. If the observed slope from the real data is far from the slopes obtained from the shuffled datasets, then we conclude that the relationship is statistically significant.

3. Image Denoising

Noise is any unwanted signal. Image denoising is the process of removing noise from an image while preserving important details such as edges and textures. Noise can be introduced in an image due to various factors, including low light conditions, sensor limitations, transmission errors, or environmental interference.

3.1. Types of Noise

3.1.1. Gaussian Noise

It appears as a smooth grain over the image and it is generated by random variations in pixel intensity. It follows a normal distribution. It is caused by sensor noise from cameras.

3.1.2. Salt-and-Pepper Noise

It appears as random black and white pixels. It is caused by sudden changes in pixel intensity due to transmission errors or faulty sensors.

3.1.3. Poisson Noise (Shot Noise)

It occurs due to fluctuations in low-light conditions. It is common in medical imaging.

3.1.4. Speckle Noise

It makes the image look grainy. It is common in radar and ultrasound images.

3.2. Singular Value Decomposition

There are many techniques for de-noising images, but we will demonstrate Singular Value Decomposition (SVD). SVD is a powerful linear algebra technique that can be used for image denoising by reducing noise while preserving important structures.

In SVD

- An image is represented as a matrix A of pixel intensities.
- SVD decomposes this matrix into three matrices

$$A = U\Sigma V^T$$

- U and V^T are orthogonal matrices. An orthogonal matrix is the one with the property that when multiplied by its transpose results in an Identity matrix.
- Σ is a diagonal matrix of singular values, which represent the importance of each feature in the image.
- Larger singular values represent important image features.
- Small singular values represent noise.
- By retaining the top-k singular values, w can remove noise from the image while keeping important structure of the image.
- To reconstruct the image, which will be a clearer version of the original image, we simply multiply the modified matrices:

$$A' = U_k\Sigma_k V_k^T$$

- SVD is effective against Gaussian noise.

3.3. Introduction to MNIST dataset

MNIST is a database of 28x28 pixels gray scale images of handwritten digits. For this section we are using a reduced MNIST datasets in CSV format. In the dataset Gaussian noise has already been added and it will be used to demonstrate image de-noising. The CSV has 784 columns, and 5000 rows. Each row represents pixels of a 28x28 image. It has been converted to INSERT statements of the form:

```
INSERT INTO mnist(img_data) VALUES(ARRAY[0,45,0,32,0, …… 52,0]);
```

which can be easily inserted into a table like

```
CREATE TABLE mnist(id SERIAL PRIMARY KEY, img_data INT[]);
```

After data insertion our dataset looks like this:

```
SELECT id, array_dims(img_data) FROM mnist;
  id  | array_dims
------+------------
    1 | [1:784]
    2 | [1:784]
    3 | [1:784]
    4 | [1:784]
    5 | [1:784]
…..
…..
…..
 4998 | [1:784]
 4999 | [1:784]
 5000 | [1:784]
(5000 rows)
```

3.4. Apache MADlib

MADlib is an open source library that can be installed as an extension in PostgreSQL. The library provides implementation of statistics, linear algebra and machine learning algorithms.

Build and install Apache MADlib binaries

```
export PATH=$PATH:/usr/lib/postgresql/15/bin

sudo mkdir /usr/local/madlib/
sudo mkdir /usr/local/madlib/Versions/2.1.0/
sudo chown abbas:abbas /usr/local/madlib/Versions/2.1.0/

git clone https://github.com/apache/madlib.git

git checkout -b V210 rel/v2.1.0

sudo apt-get install cmake g++ m4 flex bison
sudo apt-get install python3-dev

./configure
```

```
-- The C compiler identification is GNU 11.4.0
-- The CXX compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
…
…

-- Could NOT find Greenplum (missing: GREENPLUM_EXECUTABLE)
-- Using default web-based MathJax
-- Found FLEX: /usr/bin/flex (found suitable version "2.6.4", minimum required is "2.5.33")
```

```
-- Found BISON: /usr/bin/bison (found suitable version "3.8.2", minimum required is "2.4")
-- A complete LaTeX installation could not be found. Compiling the design document will not be possible.
-- Detected Debian version Ubuntu 22.04.5 LTS \n \l


-- Configuring done
-- Generating done
-- Build files have been written to: /home/abbas/Projects/madlib/build
```

```
cd build/
make
```

```
[  0%] Creating directories for 'EP_boost'
[  0%] Performing download step (verify and extract) for 'EP_boost'
-- verifying file...
     file='/home/abbas/Projects/madlib/build/third_party/downloads/boost_1_61_0.tar.gz'
-- verifying file... done
-- extracting...
     src='/home/abbas/Projects/madlib/build/third_party/downloads/boost_1_61_0.tar.gz'
     dst='/home/abbas/Projects/madlib/build/third_party/src/EP_boost'
-- extracting... [tar xfz]
-- extracting... [analysis]
-- extracting... [rename]
-- extracting... [clean up]
-- extracting... done
[  0%] No update step for 'EP_boost'
[  0%] No patch step for 'EP_boost'
[  0%] Performing configure step for 'EP_boost'
[  1%] Completed 'EP_eigen'
[  1%] Built target EP_eigen
[  1%] Built target pythonFiles
[  1%] Built target sqlFiles
[  2%] Copying __init__.py.
[  2%] Copying argparse.py.
[  2%] Copying changelist_1.10.0_1.11.yaml.
[  2%] Copying changelist_1.11_1.12.yaml.
…
…
…
…
…
```

```
[ 98%] Validating and copying stemmer/src/pg_gp/porter_stemmer.sql_in
[ 98%] Validating and copying stemmer/src/pg_gp/test/porter_stemmer.ic.sql_in
[100%] Validating and copying stemmer/src/pg_gp/test/porter_stemmer.sql_in
[100%] Validating and copying svec/src/pg_gp/svec.sql_in
[100%] Validating and copying svec_util/src/pg_gp/sql/gp_sfv_sort_order.sql_in
[100%] Validating and copying svec_util/src/pg_gp/sql/svec_test.sql_in
[100%] Validating and copying svec_util/src/pg_gp/svec_util.sql_in
[100%] Built target sqlFiles_greenplum
```

make install

```
[  0%] Built target EP_boost
[  1%] Built target EP_eigen
[  1%] Built target pythonFiles
[  1%] Built target sqlFiles
[  4%] Built target madpackFiles
[  4%] Built target binaryFiles
[  5%] Built target configFiles
[ 36%] Built target sqlFiles_postgresql
[ 50%] Built target madlib_postgresql_15
[ 69%] Built target pythonFiles_postgresql_15
[100%] Built target sqlFiles_greenplum
Install the project...
-- Install configuration: "RelWithDebInfo"
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses/third_party
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses/third_party/argparse_v1.2.1.txt
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses/third_party/Boost_Software_License_v1.txt
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses/third_party/_M_widen_init.txt
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses/third_party/Eigen_v3.1.2.txt
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses/third_party/UseLATEX_v1.9.4.txt
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses/third_party/PyYAML_v3.10.txt
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses/third_party/libstemmer_porter2.txt
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses/third_party/Python_License_v2.7.1.txt
-- Installing: /usr/local/madlib/Versions/2.1.0/./licenses/MADlib.txt
…
…
…
…
-- Installing: /usr/local/madlib/Versions/2.1.0/ports/greenplum/modules/stats/clustered_variance_coxph.sql_in
-- Installing: /usr/local/madlib/Versions/2.1.0/ports/greenplum/modules/stats/cox_prop_hazards.sql_in
```

```
-- Installing: /usr/local/madlib/Versions/2.1.0/ports/greenplum/modules/stats/robust_variance_coxph.sql_in
-- Installing: /usr/local/madlib/Versions/2.1.0/ports/greenplum/modules/stats/distribution.sql_in
-- Installing: /usr/local/madlib/Versions/2.1.0/ports/greenplum/modules/stats/hypothesis_tests.sql_in
-- Installing: /usr/local/madlib/Versions/2.1.0/ports/greenplum/modules/array_ops
-- Installing: /usr/local/madlib/Versions/2.1.0/ports/greenplum/modules/array_ops/test
-- Installing: /usr/local/madlib/Versions/2.1.0/ports/greenplum/modules/array_ops/test/array_ops.ic.sql_in
-- Installing: /usr/local/madlib/Versions/2.1.0/ports/greenplum/modules/array_ops/test/array_ops.sql_in
-- Installing: /usr/local/madlib/Versions/2.1.0/ports/greenplum/modules/array_ops/array_ops.sql_in
```

This step is required, although not documented:
cp /usr/local/madlib/Versions/2.1.0/ports/postgres/15/lib/libmadlib.so /usr/local/madlib/Versions/2.1.0/lib/

These steps have to be performed by logging in as postgres user
sudo su - postgres
export PGPORT=5432
export PGHOST=127.0.0.1
export PGUSER=postgres
export PGDATABASE=bbdb
export PGPASSWORD=abc123
/usr/local/madlib/Versions/2.1.0/bin/madpack --schema madlib -p postgres -v -l install

```
madpack.py: INFO : Arguments: Namespace(command=['install'], connstr=None, keeplogs=True, platform=['postgres'], schema=['madlib'], testcase='', tmpdir='/tmp/', verbose=True)
madpack.py: INFO : Testing database connection...
madpack.py: INFO : Detected PostgreSQL version 15.10.
madpack.py: INFO : *** Installing MADlib ***
madpack.py: INFO : MADlib tools version    = 2.1.0 (/usr/local/madlib/Versions/2.1.0/bin/../madpack/madpack.py)
madpack.py: INFO : MADlib database version = None (host=127.0.0.1:5432, db=bbdb, schema=madlib)
madpack.py: INFO : Testing PL/Python environment...
madpack.py: INFO : > PL/Python already installed
madpack.py: INFO : > PL/Python version: 3.10.12
madpack.py: INFO : > PL/Python environment OK (version: 3.10.12)
madpack.py: INFO : > Preparing objects for the following modules:
madpack.py: INFO : > - array_ops
…
madpack.py: INFO : > - validation
madpack.py: INFO : Installing MADlib:
madpack.py: INFO : > ... executing /tmp/madlib.c2eaxp88/madlib_install.sql
madpack.py: INFO : psql -a -v ON_ERROR_STOP=1 -h 127.0.0.1 -p 5432 -d bbdb -U postgres --no-password --single-transaction -f /tmp/madlib.c2eaxp88/madlib_install.sql
madpack.py: INFO : > Created madlib schema
madpack.py: INFO : > Created madlib.MigrationHistory table
madpack.py: INFO : > Wrote version info in MigrationHistory table
madpack.py: INFO : MADlib 2.1.0 installed successfully in madlib schema.
INFO: Log files saved in /tmp/madlib.c2eaxp88
```

Check version of the installed Apache MADlib

```
bbdb=# SELECT unnest(string_to_array(madlib.version(), ',')) as version;
                version
---------------------------------------
 MADlib version: 2.1.0
  git revision: rel/v2.1.0
  cmake configuration time: 30 Dec 2024 UTC 10:28:14
  build type: RelWithDebInfo
  build system: Linux-6.8.0-50-generic
  C compiler: gcc 11
  C++ compiler: g++ 11
(8 rows)
```

3.5. Builtin help in MADlib

```
bbdb=# select madlib.svd();
                                    svd
----------------------------------------------------------------------------
                                                                           +
             In linear algebra, the singular value decomposition (SVD) is a+
             factorization of a real or complex matrix, with many useful   +
             applications in signal processing and statistics.             +
             -------                                                       +
             For an overview on usage, run:                                +
             SELECT madlib.svd('usage');                                   +

bbdb=# select madlib.matrix_mult();
                         matrix_mult
--------------------------------------------------------------
                                                             +
 ------------------------------------------------------------+
                        SUMMARY                              +
 ------------------------------------------------------------+
 Functionality: Compute multiplication of two matrices       +
                                                             +
 For more details on the function usage:                     +
     SELECT madlib.matrix_mult('usage');                     +
 For more details on the two input formats (dense or sparse):+
     SELECT madlib.matrix_info();                            +
```

3.6. Image Denoising using SVD MADlib functions

3.6.1.Perform SVD

To compute SVD of the image data stored in the mnist table, we will use this query.
In the first argument we provide the input table. The function will output three tables prefixed with the second argument.
In our case the table names will be svd_u, svd_s and svd_v.
In the third argument we provide the row id of the data set and in the last argument we are providing the number of elements in each row of the img_data array.
This function take some time (10 minutes on my setup).

```
bbdb=# SELECT madlib.svd('mnist', 'svd', 'id', 784);
 svd
-----

(1 row)
```

Lets examine the results generated by this function. It generates three tables.

```
Bbdb=# \d svd_u
                     Table "public.svd_u"
 Column  |        Type        | Collation | Nullable | Default
---------+--------------------+-----------+----------+---------
 row_id  | integer            |           |          |
 row_vec | double precision[] |           |          |

bbdb=# \d svd_s
                    Table "public.svd_s"
 Column |       Type       | Collation | Nullable | Default
--------+------------------+-----------+----------+---------
 row_id | integer          |           |          |
 col_id | integer          |           |          |
 value  | double precision |           |          |

bbdb=# \d svd_v
                     Table "public.svd_v"
 Column  |        Type        | Collation | Nullable | Default
---------+--------------------+-----------+----------+---------
 row_id  | integer            |           |          |
 row_vec | double precision[] |           |          |
```

```
bbdb=# SELECT row_id, array_dims(row_vec) FROM svd_u ORDER BY row_id;

 row_id | array_dims
--------+------------
      1 | [1:784]
      2 | [1:784]
      3 | [1:784]
      4 | [1:784]
      5 | [1:784]
…
…
   4999 | [1:784]
   5000 | [1:784]
(5000 rows)

svd_u is a 5000x784 matrix


bbdb=# SELECT * FROM svd_s;

 row_id | col_id |       value
--------+--------+-------------------
      1 |      1 | 132808.17907540515
      2 |      2 |  30140.58206994838
      3 |      3 |  26841.804225118554
      4 |      4 |  22524.678072356463
      5 |      5 |  21030.462880561656
…
…
    783 |    783 |   1923.9860449982436
    784 |    784 |   1914.0603142643972
    784 |    784 |
(785 rows)

svd_s is a diagonal matrix with 784 elements.
The last value is missing but it does not matter because we will not use it any way.
```

```
bbdb=# SELECT row_id, array_dims(row_vec) FROM svd_v ORDER BY row_id;
 row_id | array_dims
--------+------------
      1 | [1:784]
      2 | [1:784]
      3 | [1:784]
      4 | [1:784]
…
…
    783 | [1:784]
    784 | [1:784]
(784 rows)

svd_v is a 784x784 matrix
```

Next we will generate de-noised images using only the first two singular values.
In our case the matrix dimensions will be:

u_hat will be a 5000x2 matrix
s_hat will be a 2x2 diagonal matrix containing the top two singular values.
v_hat will be a 2x784 matrix

Result will be a 5000x784 matrix containing de-noised images.

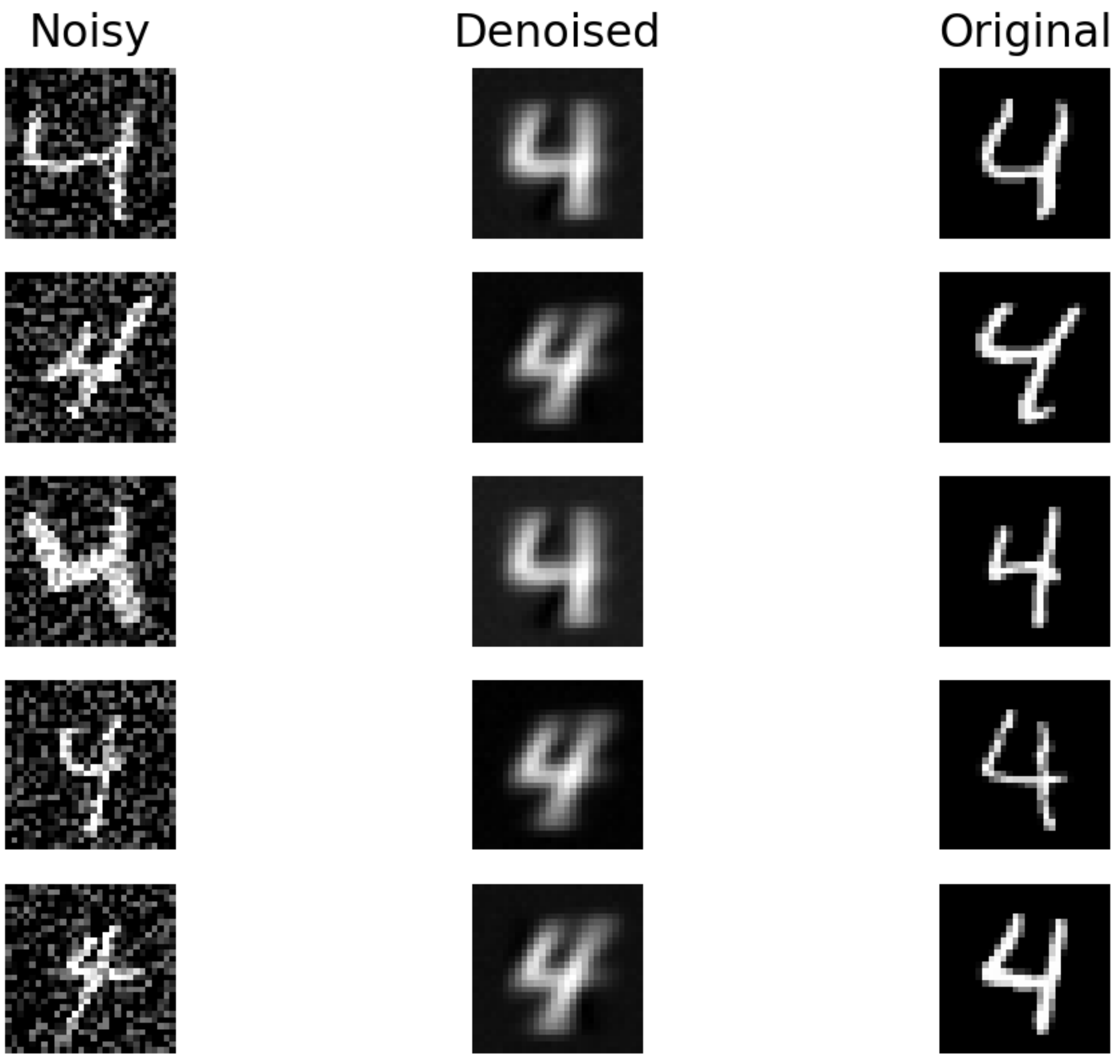
```
bbdb=# CREATE TABLE u_hat AS SELECT row_id, row_vec[1:2] FROM svd_u ORDER BY row_id;
SELECT 5000

bbdb=# SELECT * FROM u_hat ORDER BY row_id;
 row_id |                     row_vec
--------+------------------------------------------------
      1 | {-0.0112228772103092,-0.0147232035793372}
      2 | {-0.0136893677669686,0.0122244217511506}
      3 | {-0.0166623466996464,-0.0237846766312706}
      4 | {-0.0106492722442364,0.00733517661345045}
...
...
   4999 | {-0.0142686640003060,0.0185884052789658}
   5000 | {-0.0154542006225006,-0.0290628179899112}
(5000 rows)
```

```
bbdb=# SELECT madlib.matrix_trans('svd_v', 'row=row_id, val=row_vec', 'svd_vt');
 matrix_trans
--------------
 (svd_vt)
(1 row)

bbdb=# CREATE TABLE v_hat AS SELECT * FROM svd_vt WHERE row_id in (1,2);
SELECT 2

bbdb=# SELECT row_id, array_dims(row_vec) FROM v_hat;
 row_id | array_dims
--------+------------
      2 | [1:784]
      1 | [1:784]
(2 rows)

bbdb=# SELECT row_id, row_vec[1:3] FROM v_hat ORDER BY row_id;
 row_id |                              row_vec
--------+-----------------------------------------------------------------------------
      1 | {-0.016973927343477937,-0.016880827068842436,-0.017114500959360724}
      2 | {-0.002615451183550666,0.003360138322989258,-0.002435955880538372 3}
(2 rows)


bbdb=# SELECT value FROM svd_s WHERE row_id <=2;
       value
--------------------
 132808.17907540515
  30140.58206994838
(2 rows)

bbdb=# CREATE TABLE s_hat(row_id INT, row_vec DOUBLE PRECISION[]);
CREATE TABLE

bbdb=# INSERT INTO s_hat VALUES(1, ARRAY[132808.17907540515, 0]);
INSERT 0 1
bbdb=# INSERT INTO s_hat VALUES(2, ARRAY[0, 30140.58206994838]);
INSERT 0 1
```

```
bbdb=# SELECT * FROM s_hat;
 row_id |          row_vec
--------+---------------------------------------
      1 | {132808.17907540515,    0             }
      2 | {    0             ,30140.58206994838}
(2 rows)


bbdb=# SELECT madlib.matrix_mult('u_hat', 'row=row_id, val=row_vec',  's_hat', 'row=row_id, val=row_vec', 'svd_vs');
 matrix_mult
-------------
 (svd_vs)
(1 row)

bbdb=#
bbdb=#
bbdb=#
bbdb=# \d svd_vs
                     Table "public.svd_vs"
 Column  |        Type        | Collation | Nullable | Default
---------+--------------------+-----------+----------+---------
 row_id  | integer            |           |          |
 row_vec | double precision[] |           |          |


bbdb=# SELECT row_id, array_dims(row_vec) FROM svd_vs ORDER BY row_id;
 row_id | array_dims
--------+------------
      1 | [1:2]
      2 | [1:2]
      3 | [1:2]
      4 | [1:2]
…
…
   4999 | [1:2]
   5000 | [1:2]
(5000 rows)
bbdb=# SELECT madlib.matrix_mult('svd_vs', 'row=row_id, val=row_vec',  'v_hat', 'row=row_id, val=row_vec', 'mnist_denoised');
   matrix_mult
------------------
 (mnist_denoised)
```

```
bbdb=# \d mnist_denoised
                  Table "public.mnist_denoised"
 Column   |         Type         | Collation | Nullable | Default
----------+----------------------+-----------+----------+---------
 row_id   | integer              |           |          |
 row_vec  | double precision[] |           |          |


bbdb=# SELECT row_id, ARRAY_DIMS(row_vec) FROM mnist_denoised ORDER BY row_id;
 row_id | array_dims
--------+------------
      1 | [1:784]
      2 | [1:784]
      3 | [1:784]
      4 | [1:784]
      5 | [1:784]
…
…
   4999 | [1:784]
   5000 | [1:784]
(5000 rows)

bbdb=# SELECT row_id, row_vec[1:5] FROM mnist_denoised ORDER BY row_id;
 row_id |                                             row_vec
--------+------------------------------------------------------------------------------------------------
      1 | {26.460115151935607,23.669587124517253,26.589984805367042,24.65924936729609,23.933246902750355}
      2 | {29.895952351704736,31.92840351285616,30.21765887807985,32.33872639341117,31.58752417871982}
      3 | {39.43650973886871,34.94668402464943,39.61890721007245,36.46684523138222,35.38039519841728}
      4 | {23.42816197924583,24.617611413350645,23.66666070092094,24.994217199182362,24.400160481241905}
      5 | {23.594995018799768,25.525285782552277,23.859905181620295,25.8045977548551,25.216066319056793}
…
…
   4998 | {28.59391120498613,28.754229575833374,28.841420471060935,29.390162274943304,28.64791800722475}
   4999 | {30.700165576615532,33.87165677064835,31.06711691536671,34.14499118423627,33.388025681856924}
   5000 | {37.129096899020524,31.70357513747949,37.260383862162215,33.29838453863276,32.25953895402678}
(5000 rows)
```

3.6.2. Displaying the MNIST images



4. Image Classification

In image classification, an image is assigned a label or a category depending on its visual content. The features of the image are analyzed to assign it a preferred class, label or category. Handwritten digits is a classic example of image classification, where the images are analyzed to determine which digit (represented by a label) is written in the image.

Image classification is used in various applications such as

- Image search engines
- Object detection for surveillance
- Disease detection in X-rays

4.1. Dataset for classification

For this section we are going to use a reduced MNIST handwritten digits dataset in CSV format without any noise. In the dataset each digit is labeled with one of the digits from 0 to 9. This label shows which digit the image contains. If we have a model that can predict digits by processing images we can compare the results of prediction against the expected label. For this purpose the dataset has been divided into two groups. The first group contains 5000 images with known labels and it will be used to train the model. The second group contains 1000 images with known labels and it will be used to test the model accuracy. Once the model achieves the desired accuracy, it can be deployed to predict labels of the data it has never seen with the same accuracy as achieved during testing.

The CSV has 785 columns, and 5000 rows, first column is the label and the rest of the row represents pixels of a 28x28 image. It has been converted to INSERT statements of the form:

```
INSERT INTO mnist_train(label, img_data) VALUES(7,ARRAY[0,0,0,….0,0]);
```

which can be easily inserted into a table like

```
CREATE TABLE mnist_train(id SERIAL PRIMARY KEY, label INT, img_data INT[]);
```

The resulting table has data of the form

```
bbdb=# SELECT id, label, ARRAY_DIMS(img_data) FROM mnist_train ORDER BY id;
  id  | label | array_dims
------+-------+-----------
    1 |     7 | [1:784]
    2 |     2 | [1:784]
...
 4999 |     4 | [1:784]
 5000 |     0 | [1:784]
(5000 rows)
```

Similarly the test data we will have INSERTs of the form

```
INSERT INTO mnist_test(label, img_data) VALUES(7,ARRAY[0,0,0,….0,0]);
```

which can be easily inserted into a table like

```
CREATE TABLE mnist_test(id SERIAL PRIMARY KEY, label INT, img_data INT[]);
```

The resulting table has data of the form

```
bbdb=# SELECT id, label, ARRAY_DIMS(img_data) FROM mnist_test ORDER BY id;
  id  | label | array_dims
------+-------+------------
    1 |     7 | [1:784]
    2 |     6 | [1:784]
...
  999 |     5 | [1:784]
 1000 |     6 | [1:784]
(1000 rows)
```

4.2. Introduction to Neural Networks

It is easier to understand Neural Networks in the context of the current problem. We have a 28x28 pixel handwritten digit and we want a model that is provided with all the pixel values of each image and it is expected to output one of the nine digits which is the one the model thinks is written in the image.

The whole network can therefore be thought of as a function that takes 784 inputs and outputs one of the 10 digits.

$$f(a0, a1, \ldots., a783) = \begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \\ y4 \\ y5 \\ y6 \\ y7 \\ y8 \\ y9 \end{bmatrix}$$

For simplicity of diagram assume the image has 16 pixels only. Also assume there are only 4 outputs. The following picture then shows a neural network.

Each neuron in the hidden layer of the neural network performs an activation function.

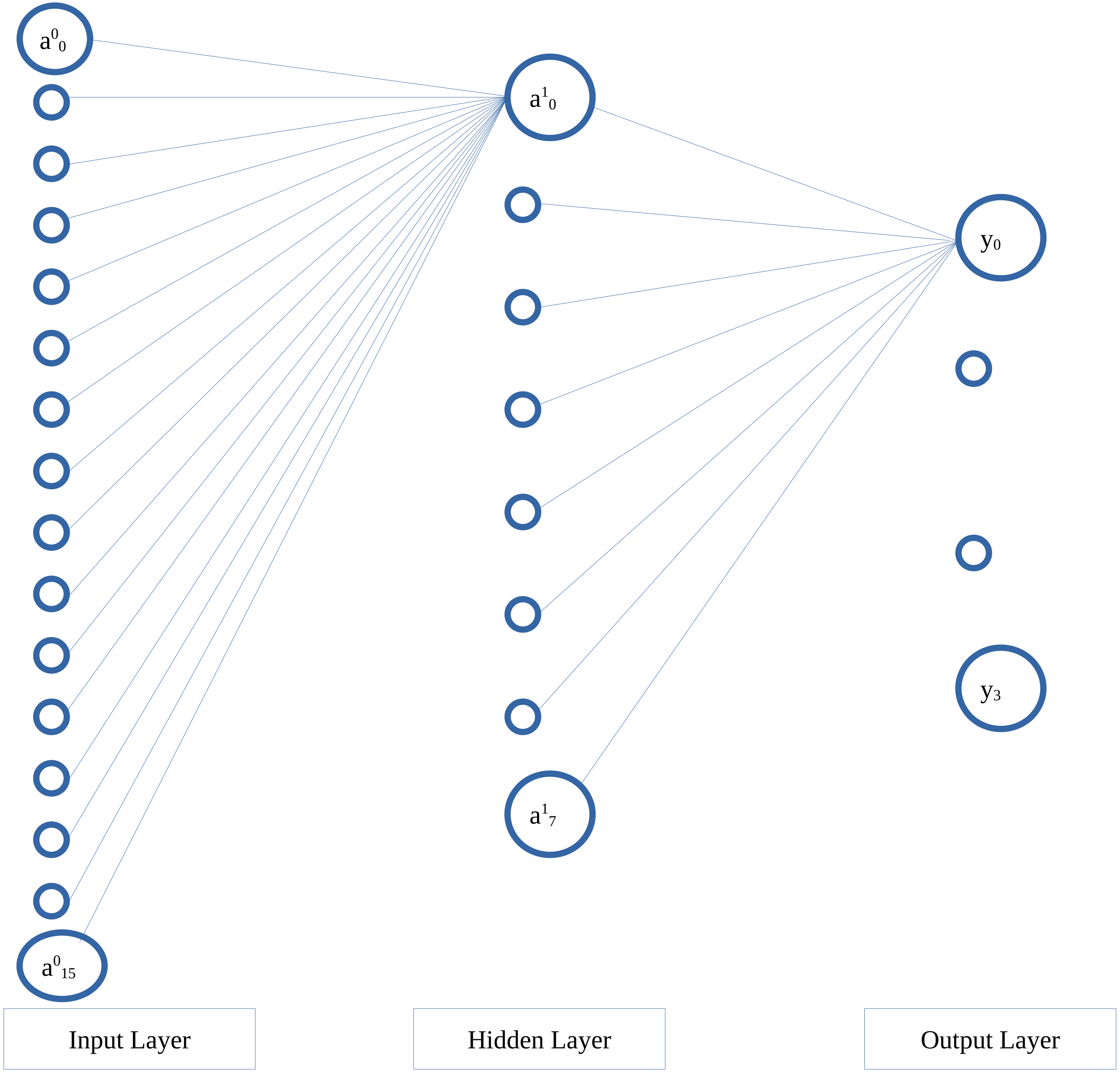
$$a^1_0 = \mathrm{RELU}(w_{0,0}a^0_0 + w_{0,1}a^0_1 + \ldots. + w_{0,15}a^0_{15}) + b_0$$

$$a^1_1 = \mathrm{RELU}(w_{1,0}a^0_0 + w_{1,1}a^0_1 + \ldots. + w_{1,15}a^0_{15}) + b_1$$

…

$$a^1_7 = \mathrm{RELU}(w_{7,0}a^0_0 + w_{7,1}a^0_1 + \ldots. + w_{7,15}a^0_{15}) + b_7$$

In matrix natation we will have $A^1 = \mathrm{RELU}(WA^0+B)$


a^0_0
a^0_{15}
a^1_0
a^1_7
y_0
y_3
Input Layer
Hidden Layer
Output Layer

In neural networks, activation refers to the output value of a neuron after applying an activation function to its weighted input. It determines whether a neuron should be activated or not, helping the network learn complex patterns.

In Activation first we compute weighted sum

Each neuron receives inputs (x_1, x_2, ...) and applies weights (w_1, w_2, ...):

$$z = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

where b is the bias term.

We then Apply Activation Function

The weighted sum (z) is passed through an activation function f(z) to introduce non-linearity:

$$y = f(z)$$

This ensures that the network can learn complex patterns.

Activation Function	Formula	Purpose
Sigmoid	$f(z) = \frac{1}{1+e^{-z}}$	Output between (0,1); used in binary classification
Tanh	$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Output between (-1,1); helps with symmetry
ReLU (Rectified Linear Unit)	$f(z) = \max(0, z)$	Most commonly used; avoids vanishing gradient

What we want to do here is provide this network our training data, which consists of labeled handwritten images, so that the network can adjust the weights and biases so as to improve its performance on training data. After training the network we provide it with test data and compare the predicted labels with the expected labels to measure accuracy.

To start with all weights and biases are set randomly. The network outputs 10 probabilities for each input image. The difference of the probabilities and expected probability is called a cost function, which training tries to reduce using many examples of training data. The algorithm used to minimize the cost is called stochastic gradient descent SGD. The gradient of a function provides us with the direction of the steepest ascent, and negative of that gives us the direction of the steepest descent. In SGD

- Compute ∇C
- Take a small step in $-\nabla C$ direction
- Repeat until target loss is achieved.

4.3. Image classification with Multilayer Perceptron using mlp_* MADlib functions

Now that we have the training and test data in place we can train a multi-layer perceptron model with mlp_classification. First lets discuss the parameters that we need to provide to the function.

```
source_table         We are providing mnist_train
output_table         We are providing mlp_mnist
independent_varname  We are providing img_data
dependent_varname    We are providing label
hidden_layer_sizes   We are providing ARRAY[256], This means there will be one hidden layer with 256 nodes
                     For 2 hidden layers we will provide ARRAY[256, 128]
optimizer_params     We are providing three parameters
                       learning_rate_init=0.001
                       n_iterations=10
                       tolerance=0, Criteria to end iterations. Default 0.001, 0 means run all iterations
activation           We are providing RELU
weights              We are providing default because we do not want to
                     give different weights to different rows during training
                     By default all rows are given equal weights
warm_start           Should the network weights be initialized with the coefficients from the last call
verbose              We are providing TRUE
grouping_col         We are proving default in which case no grouping is used and a single model is generated
```

```
bbdb=# SELECT madlib.mlp_classification('mnist_train','mlp_mnist', 'img_data','label',
ARRAY[256],'learning_rate_init=0.001, n_iterations=10, tolerance=0', 'relu', '1', FALSE, TRUE);
INFO:  Iteration: 1, Loss: <3.313416456516 1206>
INFO:  Iteration: 2, Loss: <0.5545979641316546>
INFO:  Iteration: 3, Loss: <0.268429419468565 2>
INFO:  Iteration: 4, Loss: <0.2169051119270895>
INFO:  Iteration: 5, Loss: <0.18492511031817066>
INFO:  Iteration: 6, Loss: <0.08580012272335123>
INFO:  Iteration: 7, Loss: <0.04336485889630366>
INFO:  Iteration: 8, Loss: <0.043035715694562444>
INFO:  Iteration: 9, Loss: <0.013949502029716327>
 mlp_classification
--------------------
(1 row)
```

Lets check the results generated by the function. The function generates three tables.

```
bbdb=# \d mlp_mnist
                       Table "public.mlp_mnist"
     Column     |        Type        | Collation | Nullable | Default
----------------+--------------------+-----------+----------+---------
 coeff          | double precision[] |           |          |
 loss           | double precision   |           |          |
 num_iterations | integer            |           |          |

bbdb=# SELECT ARRAY_DIMS(coeff), loss, num_iterations FROM mlp_mnist;
 array_dims |         loss          | num_iterations
------------+-----------------------+----------------
 [1:203530] | 0.011351994815696582 |             10
(1 row)
```

```
bbdb=# \d mlp_mnist_summary
                    Table "public.mlp_mnist_summary"
       Column         |        Type       | Collation | Nullable | Default
----------------------+-------------------+-----------+----------+---------
 source_table         | text              |           |          |
 independent_varname  | text              |           |          |
 dependent_varname    | text              |           |          |
 dependent_vartype    | text              |           |          |
 solver               | text              |           |          |
 tolerance            | double precision  |           |          |
 learning_rate_init   | double precision  |           |          |
 learning_rate_policy | text              |           |          |
 momentum             | double precision  |           |          |
 nesterov             | boolean           |           |          |
 rho                  | double precision  |           |          |
 beta1                | double precision  |           |          |
 beta2                | double precision  |           |          |
 eps                  | double precision  |           |          |
 n_iterations         | integer           |           |          |
 n_tries              | integer           |           |          |
 layer_sizes          | integer[]         |           |          |
 activation           | text              |           |          |
 is_classification    | boolean           |           |          |
 classes              | integer[]         |           |          |
 weights              | character varying |           |          |
 grouping_col         | character varying |           |          |
```

```
bbdb=# \x
Expanded display is on.
bbdb=# SELECT * FROM mlp_mnist_summary;
-[ RECORD 1 ]--------+---------------------
source_table         | mnist_train
independent_varname  | img_data
dependent_varname    | label
dependent_vartype    | integer
solver               | sgd
tolerance            | 0
learning_rate_init   | 0.001
learning_rate_policy | constant
momentum             | 0.9
nesterov             | t
rho                  |
beta1                |
beta2                |
eps                  |
n_iterations         | 10
n_tries              | 1
layer_sizes          | {784,256,10}
activation           | relu
is_classification    | t
classes              | {0,1,2,3,4,5,6,7,8,9}
weights              | 1
grouping_col         | NULL


bbdb=# \d mlp_mnist_standardization
           Table "public.mlp_mnist_standardization"
 Column |        Type        | Collation | Nullable | Default
--------+--------------------+-----------+----------+---------
 mean   | double precision[] |           |          |
 std    | double precision[] |           |          |

mean The mean for all input features (used for normalization).
std  The standard deviation for all input features (used for normalization).
```

```
bbdb=# SELECT ARRAY_DIMS(mean), ARRAY_DIMS(std) FROM mlp_mnist_standardization;
 array_dims | array_dims
------------+------------
 [1:784]    | [1:784]
(1 row)
bbdb=# SELECT MIN(value) AS min_mean, MAX(value) AS max_mean FROM mlp_mnist_standardization, unnest(mean) AS t(value);
 min_mean | max_mean
----------+----------
        0 |   135.45
(1 row)

bbdb=# SELECT MIN(value) AS min_std, MAX(value) AS max_std FROM mlp_mnist_standardization, unnest(std) AS t(value);
       min_std        |      max_std
----------------------+--------------------
 0.014140721339450827 | 114.16593357810376
(1 row)
```

First lets discuss the parameters that we need to provide to the function.

model_table	We are providing mlp_mnist
data_table	We are providing mnist_test
id_col_name	We are providing id
output_table	We are providing mlp_predict
pred_type	The type of output requested: 'response' gives the actual prediction, 'prob' gives the probability of each class.

```
bbdb=# SELECT madlib.mlp_predict('mlp_mnist', 'mnist_test', 'id', 'mlp_predict','response');
 mlp_predict
-------------

(1 row)
```

The function generates the following table.

```
bbdb=# \d mlp_predict
                 Table "public.mlp_predict"
     Column      |  Type   | Collation | Nullable | Default
-----------------+---------+-----------+----------+---------
 id              | integer |           |          |
 estimated_label | integer |           |          |
```

Lets check the mis-classifications.

```
bbdb=# SELECT t.id, t.label, p.estimated_label FROM mnist_test t, mlp_predict p
       WHERE t.id = p.id AND t.label != p.estimated_label;
 id  | label | estimated_label
-----+-------+-----------------
   5 |     0 |   5
   8 |     3 |   5
  10 |     7 |   2
  11 |     2 |   8
  15 |     6 |   0
  16 |     7 |   2
  17 |     0 |   5
  20 |     7 |   2
  23 |     3 |   2
  25 |     7 |   2
  37 |     7 |   2
  58 |     9 |   5
  72 |     1 |   8
  75 |     4 |   6
  85 |     4 |   9
 199 |     9 |   5
 215 |     9 |   5
 218 |     3 |   2
 226 |     2 |   3
 228 |     3 |   2
 230 |     3 |   5
 239 |     3 |   5
 246 |     9 |   7
 249 |     7 |   9
 260 |     2 |   3
 281 |     8 |   5
 317 |     8 |   5
 423 |     5 |   3
 434 |     8 |   6
 480 |     3 |   8
 483 |     5 |   3
 501 |     2 |   9
 535 |     7 |   9
 539 |     4 |   9
 579 |     2 |   3
```

```
 588 |        9 |    4
 595 |        1 |    8
 615 |        3 |    5
 618 |        6 |    2
 620 |        8 |    9
 625 |        3 |    8
 635 |        0 |    2
 643 |        9 |    7
 650 |        1 |    8
 654 |        1 |    7
 656 |        3 |    2
 663 |        3 |    2
 665 |        2 |    9
 667 |        0 |    2
 670 |        4 |    5
 671 |        3 |    8
 674 |        8 |    5
 679 |        6 |    2
 680 |        6 |    5
 685 |        7 |    5
 693 |        9 |    7
 698 |        3 |    5
 699 |        6 |    5
 701 |        2 |    5
 708 |        3 |    5
 714 |        9 |    7
 720 |        5 |    0
 727 |        2 |    0
 730 |        5 |    6
 732 |        7 |    5
 733 |        8 |    5
 734 |        9 |    5
 743 |        3 |    2
 745 |        8 |    1
 746 |        4 |    2
 747 |        7 |    5
 750 |        5 |    2
 753 |        2 |    0
 756 |        8 |    5
 766 |        7 |    5
 769 |        2 |    0
```

```
 771 |       5 |   0
 774 |       7 |   5
 778 |       5 |   2
 780 |       2 |   0
 793 |       4 |   3
 809 |       9 |   4
 833 |       2 |   8
 841 |       3 |   2
 842 |       5 |   6
 857 |       9 |   5
 859 |       6 |   8
 868 |       2 |   8
 880 |       0 |   2
 884 |       5 |   6
 891 |       9 |   4
 892 |       9 |   4
 894 |       2 |   8
 902 |       9 |   4
 906 |       3 |   9
 911 |       8 |   5
 913 |       1 |   2
 915 |       3 |   9
 920 |       7 |   9
 925 |       9 |   7
 926 |       3 |   8
 928 |       4 |   6
 934 |       2 |   0
 937 |       8 |   5
 938 |       2 |   3
 942 |       5 |   6
 943 |       3 |   5
 944 |       3 |   5
 945 |       3 |   5
 948 |       4 |   0
 956 |       1 |   2
 971 |       5 |   3
 976 |       3 |   8
 981 |       2 |   3
 983 |       5 |   6
(115 rows)
```

Lets do another example:

```
bbdb=# SELECT madlib.mlp_classification('mnist_train','mlp_mnist3', 'img_data','label',
       ARRAY[256, 128, 64],'learning_rate_init=0.001, n_iterations=10, tolerance=0', 'relu', '1', FALSE, TRUE);

INFO:  Iteration: 1, Loss: <3.2559935255734644>
INFO:  Iteration: 2, Loss: <0.7794475894917671>
INFO:  Iteration: 3, Loss: <0.31973053888558817>
INFO:  Iteration: 4, Loss: <0.1976400571724536>
INFO:  Iteration: 5, Loss: <0.15410044165677772>
INFO:  Iteration: 6, Loss: <0.14106056318784993>
INFO:  Iteration: 7, Loss: <0.1729680779623184>
INFO:  Iteration: 8, Loss: <0.10716053660949071>
INFO:  Iteration: 9, Loss: <0.07296497366155887>
 mlp_classification
--------------------

(1 row)

bbdb=# SELECT madlib.mlp_predict('mlp_mnist3', 'mnist_test', 'id', 'mlp_predict3','response');
 mlp_predict
-------------

(1 row)

bbdb=# SELECT t.id, t.label, p.estimated_label FROM mnist_test t, mlp_predict3 p
       WHERE t.id = p.id AND t.label != p.estimated_label;


bbdb=# SELECT count(*) FROM mnist_test t, mlp_predict3 p
       WHERE t.id = p.id AND t.label != p.estimated_label;
 count
-------
   110
(1 row)
```

5. Concluding Remarks