Cookbook Chapter 2 Translated to HANA SQL

```
-- Create function for month difference. Can be refined.
drop function months_between;
create function months_between(start_date timestamp, end_date timestamp) RETURNS mb integer
language SQLSCRIPT READS SQL DATA AS
begin
	mb := (year(:end_date) - year(:start_date))*12 + (month(:end_date) - month(:start_date));
end;

-- width_bucket definition.
drop function width_bucket;
create function width_bucket(val Double, start_val Double, end_val Double, nb_buckets integer)
RETURNS bucket integer
language SQLSCRIPT READS SQL DATA AS
begin
	DECLARE b integer := floor((:val - :start_val) * (:nb_buckets / (:end_val - :start_val))) + 1;
	IF b > :nb_buckets THEN
	bucket:= nb_buckets + 1;
	ELSE
	bucket := b;
	END IF;
end;
```

1. Age Histogram

```
select bucket + 15, count(*) from (
	select width_bucket(months_between("dp"."dob", "ad"."admit_dt")/12, 15, 100, 85) as bucket
	from "MIMIC2"."mimic.tables::admissions" "ad", "MIMIC2"."mimic.tables::d_patients" "dp"
	where "ad"."subject_id" = "dp"."subject_id" and months_between("dp"."dob", "ad"."admit_dt") / 12
between 15 and 199)
group by bucket order by bucket
```

2. Height Histogram

```
select bucket, count(*) from (
	select "value1num", floor("value1num" * 200/ (200 - 0)) as bucket
	from "MIMIC2"."mimic.tables::chartevents"
	where "itemid" = 920 and "value1num" is not null and "value1num" between 1 and 499) x
group by bucket order by bucket
```

3. Blood urea nitrogen (BUN) histogram

```
select bucket, count(*) from (
        select width_bucket("le"."valuenum", 0, 280, 280) as bucket
        from "MIMIC2"."mimic.tables::labevents" "le", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (50177) and "le"."subject_id" = "dp"."subject_id" and
months_between("dp"."dob", "le"."charttime")/12 > 15)
group by bucket order by bucket;
```

4. Get Gladow come scale (GSC) histogram

```
select bucket, count(*) from (
        select width_bucket("ce"."value1num", 1, 30, 20) as bucket
        from "MIMIC2"."mimic.tables::chartevents" "ce", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (198) and "ce"."subject_id" = "dp"."subject_id" and
months_between("dp"."dob", "ce"."charttime")/12 > 15)
group by bucket order by bucket;
```

5. Serum glucose histogram

```
select bucket, count(*) from (
        select width_bucket("le"."valuenum", 0.5, 1000, 1000) as bucket
        from "MIMIC2"."mimic.tables::labevents" "le", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (50006, 50112) and "le"."valuenum" is not null and "le"."subject_id" =
"dp"."subject_id" and months_between("dp"."dob", "le"."charttime")/12 > 15)
group by bucket order by bucket;
```

6. Serum HCO3 Histogram

```
select bucket, count(*) from (
        select width_bucket("le"."valuenum", 0, 231, 231) as bucket
        from "MIMIC2"."mimic.tables::labevents" "le"
        where "itemid" in (50022, 50025, 50172))
group by bucket order by bucket;
```

7. Hematocrit (%) Histogram

```
select bucket, count(*) from (
        select width_bucket("ce"."value1num", 0, 150, 150) as bucket
        from "MIMIC2"."mimic.tables::chartevents" "ce", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (813) and "ce"."subject_id" = "dp"."subject_id" and
months_between("dp"."dob", "ce"."charttime")/12 > 15)
group by bucket order by bucket;
```

8. Heart Rate Histogram

```
select bucket, count(*) from (
        select width_bucket("ce"."value1num", 0, 300, 301) as bucket
        from "MIMIC2"."mimic.tables::chartevents" "ce", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" = 211 and "ce"."subject_id" = "dp"."subject_id" and months_between("dp"."dob",
"ce"."charttime")/12 > 15)
group by bucket order by bucket;
```

9. Serum Potassium Histogram

```
select bucket, count(*) from (
        select width_bucket("le"."valuenum", 0, 10, 100) as bucket
        from "MIMIC2"."mimic.tables::labevents" "le", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (50009, 50149) and "le"."subject_id" = "dp"."subject_id" and
months_between("dp"."dob", "le"."charttime")/12 > 15)
group by bucket order by bucket;
```

10. RR interval Histogram

```
select bucket, count(*) from (
        select "value1num", width_bucket("ce"."value1num", 0, 130, 1400) as bucket
        from "MIMIC2"."mimic.tables::chartevents" "ce", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (219, 615, 618) and "ce"."subject_id" = "dp"."subject_id" and
months_between("dp"."dob", "ce"."charttime")/12 > 15)
group by bucket order by bucket;
```

11. Systolic Blood Pressure Histogram

```
select bucket, count(*) from (
        select width_bucket("ce"."value1num", 0, 300, 300) as bucket
        from "MIMIC2"."mimic.tables::chartevents" "ce", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (6, 51, 455, 6701) and "ce"."subject_id" = "dp"."subject_id" and
months_between("dp"."dob", "ce"."charttime")/12 > 15)
group by bucket order by bucket;
```

12. Sodium Histogram

```
select bucket, count(*) from (
        select width_bucket("le"."valuenum", 0, 180, 180) as bucket
        from "MIMIC2"."mimic.tables::labevents" "le", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (50012, 50159) and "le"."subject_id" = "dp"."subject_id" and
months_between("dp"."dob", "le"."charttime")/12 > 15)
group by bucket order by bucket;
```

13. Body temperature Histogram

```
select bucket, count(*) from (
        select width_bucket(
        case
        when "ce"."itemid" in (676, 677) then "ce"."value1num"
        when "ce"."itemid" in (678, 679) then ("ce"."value1num" - 32) * 5 / 9
        end, 30, 45, 160) as bucket
        from "MIMIC2"."mimic.tables::chartevents" "ce", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (676, 677, 678, 679) and "ce"."subject_id" = "dp"."subject_id" and
months_between("dp"."dob", "ce"."charttime")/12 > 15)
group by bucket order by bucket;
```

14. Urine Output Histogram

```
select bucket, count(*) from (
        select width_bucket("ie"."volume", 0, 1000, 200) as bucket
        from "MIMIC2"."mimic.tables::ioevents" "ie", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (55, 56, 57, 61, 65, 69, 85, 94, 96, 288, 405, 428, 473, 651, 715, 1922, 2042,
2068, 2111, 2119, 2130, 2366, 2463,
        2507, 2510, 2592, 2676, 2810, 2859, 3053, 3175, 3462, 3519, 3966, 3987, 4132, 4253, 5927)
        and "ie"."subject_id" = "dp"."subject_id" and months_between("dp"."dob", "ie"."charttime")/12 >
15)
group by bucket order by bucket;
```

15. White Blood Cell Count Histogram

```
select bucket/10, count(*) from (
        select width_bucket("le"."valuenum", 0, 100, 1001) as bucket
        from "MIMIC2"."mimic.tables::labevents" "le", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (50316, 50468) and "le"."valuenum" is not null and "le"."subject_id" =
"dp"."subject_id" and months_between("dp"."dob", "le"."charttime")/12 > 15)
group by bucket order by bucket;
```

Cookbook Using R Integration

1. Trivial R example calculation of mean height

```
--#create view containing heights
drop view height_only_in;
create view height_only_in
as select "value1num"
from "MIMIC2"."mimic.tables::chartevents"
where "itemid" = 920 and "value1num" is not null and "value1num" > 0 and "value1num" < 500;

--#single row to contain mean
drop table output;
create table output(MEAN INTEGER);

--#create SQL-script function including R script
drop procedure avg_height;
create procedure avg_height(IN input1 height_only_in, OUT result output)
language RLANG as
begin
result <- as.data.frame(mean(input1$value1num));
names(result) <- c("MEAN");
end;

--#execute SQL-script function and retrieve result
call avg_height (height_only_in, output) with OVERVIEW;
select * from output;
```

2. Quantile of Heights

```
drop view height_only_in;
create view height_only_in
as SELECT "value1num"
from "MIMIC2"."mimic.tables::chartevents"
where "itemid" = 920 and "value1num" is not null and "value1num" > 0 and "value1num" < 500;

drop table output;
create column table output(ZERO INTEGER, PERCENT25 integer, PERCENT50 integer,
PERCENT75 integer, PERCENT100 integer);

--#create SQL-script function including R script
DROP PROCEDURE avg_height;
CREATE PROCEDURE avg_height(IN input1 height_only_in, OUT result output)
LANGUAGE RLANG AS
BEGIN
```

```
result <- as.data.frame(t(quantile(input1$value1num)));
names(result) <- c("ZERO","PERCENT25","PERCENT50","PERCENT75","PERCENT100");
END;

--#execute SQL-script function and retrieve result
CALL avg_height (height_only_in, output) WITH OVERVIEW;
SELECT * FROM output;
```

3. Death prediction with SVM

```
--Create the table that contains all the information and features that we want for our prediction algorithm
--Note that we create the column 'training' where training examples are selected at random. 1 means it's for
training, 0 for testing.
drop table "MIMIC2"."mimic.prediction::death_features";
create column table "MIMIC2"."mimic.prediction::death_features" as (
        select "icud"."icustay_id" ,
        "icud"."dob" ,
        "icud"."dod" ,
        "icud"."hospital_admit_dt" ,
        "icud"."icustay_admit_age" "age",
        "icud"."weight_first" ,
        "icud"."weight_min" ,
        "icud"."weight_max" ,
        "icud"."sapsi_first",
        "icud"."sofa_first",
        map("icud"."gender",'F',0,'M',1) "gender",
        map("icud"."dod",null,0,1) DEAD,
        1-floor(rand()/0.75) "training"
        from "MIMIC2"."mimic.tables::icustay_detail" "icud"
        where "icud"."weight_first" is not null and
        "icud"."weight_min" is not null and
        "icud"."weight_max" is not null and
        "icud"."sapsi_first" is not null and
        "icud"."sofa_first" is not null and
        "icud"."gender" is not null
);
alter table "MIMIC2"."mimic.prediction::death_features" add (CPREDICT nvarchar(10), CPROB decimal);


-- Create a procedure that will train and predict.
-- This could be split into two different procedure, one for training the other one for testing.
drop procedure "MIMIC2"."proc_death_train_predict";
create procedure "MIMIC2"."proc_death_train_predict" (in input1
"MIMIC2"."mimic.prediction::death_features",
in input2 "MIMIC2"."mimic.prediction::death_features",
out output1 "MIMIC2"."mimic.prediction::death_features")
language RLANG as
```

```
begin
        library("kernlab");
        library(RODBC);

        myconn <-odbcConnect("hana", uid="SYSTEM", pwd="HANA4ever");

        input_training <- input1;
        meta_cols <- c("dob","dod", "hospital_admit_dt","DEAD","training","CPREDICT","CPROB");
        x_train <- data.matrix(input_training[-match(meta_cols, names(input_training))]);
        y_train <- input_training$DEAD;
        model <- ksvm(x_train, y_train, type = "C-bsvc", kernel = "rbfdot", kpar = list(sigma = 0.1), C = 10,
prob.model = TRUE);

        input_test <- input2;
        x_test <- input_test[-match(meta_cols, names(input_test))];

        prob_matrix <- predict(model, x_test, type="probabilities");

        clabel <- apply(prob_matrix, 1, which.max);
        cprob <- apply(prob_matrix, 1, max);
        classlabels = colnames(prob_matrix);
        clabel <- classlabels[clabel];

        output1 <- input2;
        output1$CPREDICT <- clabel;
        output1$CPROB <- cprob;

end;


-- Create the table for training. Subset of the main table.
drop table "MIMIC2"."mimic.prediction::death_train";
create column table "MIMIC2"."mimic.prediction::death_train" as (
        select * from "MIMIC2"."mimic.prediction::death_features" where "training" = 1
);
select count(*) from "MIMIC2"."mimic.prediction::death_prediction";

-- Create the table for testing. Subset of the main table.
drop table "MIMIC2"."mimic.prediction::death_prediction";
create column table "MIMIC2"."mimic.prediction::death_prediction" as (
        select * from "MIMIC2"."mimic.prediction::death_features" where "training" = 0
);

drop table "MIMIC2"."mimic.prediction::death_output";
create column table "MIMIC2"."mimic.prediction::death_output" like
"MIMIC2"."mimic.prediction::death_features";

call "MIMIC2"."proc_death_train_predict"("MIMIC2"."mimic.prediction::death_train",
"MIMIC2"."mimic.prediction::death_prediction","MIMIC2"."mimic.prediction::death_output") with overview;


select 1-sum(abs(DEAD - CPREDICT))/count(*) from "MIMIC2"."mimic.prediction::death_output";
```

Cookbook Using PAL Library

1. Histograms

a. Age

```
drop function months_between;
create function months_between(start_date timestamp, end_date timestamp) RETURNS mb integer
language SQLSCRIPT READS SQL DATA AS
begin
	mb := (year(:end_date) - year(:start_date))*12 + (month(:end_date) - month(:start_date));
end;

-- Tells the system to use the schema _SYS_AFL. This is more or less similar to a namespace. Everything following this will apply to this schema
set schema _SYS_AFL;

-- Here lies the data table for the attributes that BINNING will use. Can put as many parameters as required
drop type AGE_BIN_DATA;
create type AGE_BIN_DATA as table (SUBJECT_ID INTEGER, MB DOUBLE);

-- Regular table used by every PAL function. This is filled later on
drop type AGE_BIN_PARAMS;
create type AGE_BIN_PARAMS as table (NAME VARCHAR (50), INTARGS INTEGER, DOUBLEARGS DOUBLE, STRINGARGS VARCHAR (100));

-- Output table for the results
drop type AGE_BIN_RESULTS;
create type AGE_BIN_RESULTS as table (SUBJECT_ID INTEGER, VAR_TYPE INTEGER, VAR_PRE_RESULT DOUBLE);

-- Signature table that the PAL function will use
drop table AGE_BIN_SIGNATURE;
create column table AGE_BIN_SIGNATURE (ID INTEGER, TYPENAME VARCHAR(100), DIRECTION VARCHAR(100));
insert into AGE_BIN_SIGNATURE values (1, 'AGE_BIN_DATA', 'in');
insert into AGE_BIN_SIGNATURE values (2, 'AGE_BIN_PARAMS', 'in');
insert into AGE_BIN_SIGNATURE values (3, 'AGE_BIN_RESULTS', 'out');

-- Create the procedure
drop procedure _SYS_AFL.AGE_BIN;
drop type _SYS_AFL.AGE_BIN__TT_P1;
drop type _SYS_AFL.AGE_BIN__TT_P2;
drop type _SYS_AFL.AGE_BIN__TT_P3;
call SYSTEM.AFL_WRAPPER_GENERATOR('AGE_BIN','AFLPAL','BINNING', AGE_BIN_SIGNATURE);

-- Use the schema MIMIC2 for the following instructions
set schema MIMIC2;
```

```
-- Create a view to select specific fields from a table.
-- This allows a fine-grained control over what will be used in the algorithm instead of the complete tables
-- and also create relevant features that are not normally in the database
drop view V_BIN_DATA;
create view V_BIN_DATA as
        select "ad"."subject_id" as SUBJECT_ID, months_between("dp"."dob", "ad"."admit_dt")/12 as MB
        from "MIMIC2"."mimic.tables::admissions" "ad", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "ad"."subject_id" = "dp"."subject_id" and months_between("dp"."dob", "ad"."admit_dt") / 12
between 15 and 199;

-- Create a table for the function parameters
drop table #BIN_PARAMS;
drop table BIN_RESULTS;
create local temporary column table #BIN_PARAMS like _SYS_AFL.AGE_BIN_PARAMS;
create column table BIN_RESULTS like _SYS_AFL.AGE_BIN_RESULTS;

-- Populate this parameters table
insert into #BIN_PARAMS values('BINNING_METHOD',0,null,null);
insert into #BIN_PARAMS values('SMOOTH_METHOD',2,null,null);
insert into #BIN_PARAMS values('BIN_NUMBER',85,null,null);
--insert into #BIN_PARAMS values('BIN_DISTANCE',10,null,null);
--insert into #BIN_PARAMS values('SD',1,null,null);


-- Allows us to try different value for the parameters
--update #BIN_PARAMS set INTARGS=15 where name='BINNING_METHOD';


-- Empty the 'out' tables before running the KMeans function
truncate table BIN_RESULTS;

call _SYS_AFL.AGE_BIN(V_BIN_DATA, #BIN_PARAMS, BIN_RESULTS) with OVERVIEW;

-- To get the histogram, either run the following query or look at the data visualization tool available (right
click on a table > Open Data Preview > Select the Analysis tab)
select BIN_RESULTS.VAR_TYPE, count(*) from BIN_RESULTS group by BIN_RESULTS.VAR_TYPE
order by BIN_RESULTS.VAR_TYPE;
```

b. Height

```
drop function months_between;
create function months_between(start_date timestamp, end_date timestamp) RETURNS mb integer
language SQLSCRIPT READS SQL DATA AS
begin
        mb := (year(:end_date) - year(:start_date))*12 + (month(:end_date) - month(:start_date));
end;


-- Tells the system to use the schema _SYS_AFL. This is more or less similar to a namespace. Everything
following this will apply to this schema
set schema _SYS_AFL;


-- Here lies the data table for the attributes that BINNING will use. Can put as many parameters as required
drop type HEIGHT_BIN_DATA;
create type HEIGHT_BIN_DATA as table (SUBJECT_ID INTEGER, H DOUBLE);

-- Regular table used by every PAL function. This is filled later on
drop type HEIGHT_BIN_PARAMS;
create type HEIGHT_BIN_PARAMS as table (NAME VARCHAR (50), INTARGS INTEGER,
DOUBLEARGS DOUBLE, STRINGARGS VARCHAR (100));

-- Output table for the results
drop type HEIGHT_BIN_RESULTS;
create type HEIGHT_BIN_RESULTS as table (SUBJECT_ID INTEGER, VAR_TYPE INTEGER,
VAR_PRE_RESULT DOUBLE);

-- Signature table that the PAL function will use
drop table HEIGHT_BIN_SIGNATURE;
create column table HEIGHT_BIN_SIGNATURE (ID INTEGER, TYPENAME VARCHAR(100), DIRECTION
VARCHAR(100));
insert into HEIGHT_BIN_SIGNATURE values (1, 'HEIGHT_BIN_DATA', 'in');
insert into HEIGHT_BIN_SIGNATURE values (2, 'HEIGHT_BIN_PARAMS', 'in');
insert into HEIGHT_BIN_SIGNATURE values (3, 'HEIGHT_BIN_RESULTS', 'out');

-- Create the procedure
drop procedure _SYS_AFL.HEIGHT_BIN;
drop type _SYS_AFL.HEIGHT_BIN__TT_P1;
drop type _SYS_AFL.HEIGHT_BIN__TT_P2;
drop type _SYS_AFL.HEIGHT_BIN__TT_P3;
call SYSTEM.AFL_WRAPPER_GENERATOR('HEIGHT_BIN','AFLPAL','BINNING',
HEIGHT_BIN_SIGNATURE);

-- Use the schema MIMIC2 for the following instructions
set schema MIMIC2;
```

```
-- Create a view to select specific fields from a table.
-- This allows a fine-grained control over what will be used in the algorithm instead of the complete tables
-- and also create relevant features that are not normally in the database
drop view V_HEIGHT_DATA;
create view V_HEIGHT_DATA as
        select "subject_id" as SUBJECT_ID, "value1num" as H
        from "MIMIC2"."mimic.tables::chartevents"
        where "itemid" = 920 and "value1num" is not null and "value1num" between 1 and 499;

-- Create a table for the function parameters
drop table #HEIGHT_PARAMS;
drop table HEIGHT_RESULTS;
create local temporary column table #HEIGHT_PARAMS like _SYS_AFL.HEIGHT_BIN_PARAMS;
create column table HEIGHT_RESULTS like _SYS_AFL.HEIGHT_BIN_RESULTS;

-- Populate this parameters table
insert into #HEIGHT_PARAMS values('BINNING_METHOD',0,null,null);
insert into #HEIGHT_PARAMS values('SMOOTH_METHOD',2,null,null);
insert into #HEIGHT_PARAMS values('BIN_NUMBER',200,null,null);
--insert into #HEIGHT_PARAMS values('BUN_DISTANCE',10,null,null);
--insert into #HEIGHT_PARAMS values('SD',1,null,null);


-- Allows us to try different value for the parameters
--update #HEIGHT_PARAMS set INTARGS=15 where name='BINNING_METHOD';


-- Empty the 'out' tables before running the KMeans function
truncate table HEIGHT_RESULTS;

call _SYS_AFL.HEIGHT_BIN(V_HEIGHT_DATA, #HEIGHT_PARAMS, HEIGHT_RESULTS) with
OVERVIEW;

-- To get the histogram, either run the following query or look at the data visualization tool available (right
click on a table > Open Data Preview > Select the Analysis tab)
select HEIGHT_RESULTS.VAR_TYPE, count(*) from HEIGHT_RESULTS group by
HEIGHT_RESULTS.VAR_TYPE order by HEIGHT_RESULTS.VAR_TYPE;
```

c. Blood urea nitrogen (BUN)

```
drop function months_between;
create function months_between(start_date timestamp, end_date timestamp) RETURNS mb integer
language SQLSCRIPT READS SQL DATA AS
begin
	mb := (year(:end_date) - year(:start_date))*12 + (month(:end_date) - month(:start_date));
end;


-- Tells the system to use the schema _SYS_AFL. This is more or less similar to a namespace. Everything following this will apply to this schema
set schema _SYS_AFL;


-- Here lies the data table for the attributes that BINNING will use. Can put as many parameters as required
drop type BUN_BIN_DATA;
create type BUN_BIN_DATA as table (SUBJECT_ID INTEGER, BUN DOUBLE);

-- Regular table used by every PAL function. This is filled later on
drop type BUN_BIN_PARAMS;
create type BUN_BIN_PARAMS as table (NAME VARCHAR (50), INTARGS INTEGER, DOUBLEARGS DOUBLE, STRINGARGS VARCHAR (100));

-- Output table for the results
drop type BUN_BIN_RESULTS;
create type BUN_BIN_RESULTS as table (SUBJECT_ID INTEGER, VAR_TYPE INTEGER, VAR_PRE_RESULT DOUBLE);

-- Signature table that the PAL function will use
drop table BUN_BIN_SIGNATURE;
create column table BUN_BIN_SIGNATURE (ID INTEGER, TYPENAME VARCHAR(100), DIRECTION VARCHAR(100));
insert into BUN_BIN_SIGNATURE values (1, 'BUN_BIN_DATA', 'in');
insert into BUN_BIN_SIGNATURE values (2, 'BUN_BIN_PARAMS', 'in');
insert into BUN_BIN_SIGNATURE values (3, 'BUN_BIN_RESULTS', 'out');

-- Create the procedure
drop procedure _SYS_AFL.BUN_BIN;
drop type _SYS_AFL.BUN_BIN__TT_P1;
drop type _SYS_AFL.BUN_BIN__TT_P2;
drop type _SYS_AFL.BUN_BIN__TT_P3;
call SYSTEM.AFL_WRAPPER_GENERATOR('BUN_BIN','AFLPAL','BINNING', BUN_BIN_SIGNATURE);

-- Use the schema MIMIC2 for the following instructions
set schema MIMIC2;

-- Create a view to select specific fields from a table.
-- This allows a fine-grained control over what will be used in the algorithm instead of the complete tables
-- and also create relevant features that are not normally in the database
drop view V_BUN_DATA;
create view V_BUN_DATA as
        select "le"."subject_id" as SUBJECT_ID, "le"."valuenum" as BUN
        from "MIMIC2"."mimic.tables::labevents" "le", "MIMIC2"."mimic.tables::d_patients" "dp"
        where "itemid" in (50177) and "le"."subject_id" = "dp"."subject_id" and
months_between("dp"."dob", "le"."charttime")/12 > 15 and "le"."valuenum" is not null;

-- Create a table for the function parameters
drop table #BUN_PARAMS;
drop table BUN_RESULTS;
create local temporary column table #BUN_PARAMS like _SYS_AFL.BUN_BIN_PARAMS;
create column table BUN_RESULTS like _SYS_AFL.BUN_BIN_RESULTS;

-- Populate this parameters table
insert into #BUN_PARAMS values('BINNING_METHOD',0,null,null);
insert into #BUN_PARAMS values('SMOOTH_METHOD',2,null,null);
insert into #BUN_PARAMS values('BIN_NUMBER',280,null,null);
--insert into #BUN_PARAMS values('BUN_DISTANCE',10,null,null);
--insert into #BUN_PARAMS values('SD',1,null,null);


-- Allows us to try different value for the parameters
--update #BIN_PARAMS set INTARGS=15 where name='BINNING_METHOD';


-- Empty the 'out' tables before running the KMeans function
truncate table BUN_RESULTS;

call _SYS_AFL.BUN_BIN(V_BUN_DATA, #BUN_PARAMS, BUN_RESULTS) with OVERVIEW;

-- To get the histogram, either run the following query or look at the data visualization tool available (right
click on a table > Open Data Preview > Select the Analysis tab)
select BUN_RESULTS.VAR_TYPE, count(*) from BUN_RESULTS group by BUN_RESULTS.VAR_TYPE
order by BUN_RESULTS.VAR_TYPE;
```

2. Clustering

a. KMeans

```
-- Tells PAL to use the schema _SYS_AFL. This is more or less similar to a namespace. Everything following this will apply to this schema
set schema _SYS_AFL;

drop type CE_KM_DATA;
drop type CE_KM_PARAMS;
drop type CE_KM_RESULTS;
drop type CE_KM_CENTER_POINTS;

-- Here lies the data table for the attributes that KMEANS will use. Can put as many parameters as required
create type CE_KM_DATA as table (SUBJECT_ID INTEGER, VALUE1NUM DOUBLE, VALUE2NUM DOUBLE);
create type CE_KM_PARAMS as table (NAME VARCHAR (50), INTARGS INTEGER, DOUBLEARGS DOUBLE, STRINGARGS VARCHAR (100));
create type CE_KM_RESULTS as table (CE_ID INTEGER, CLUSTER_ID INTEGER, DISTANCE DOUBLE);
create type CE_KM_CENTER_POINTS as table (CE_ID INTEGER, VALUE1NUM DOUBLE, VALUE2NUM DOUBLE);

drop table CE_KM_SIGNATURE;
create column table CE_KM_SIGNATURE (ID INTEGER, TYPENAME VARCHAR(100), DIRECTION VARCHAR(100));
insert into CE_KM_SIGNATURE values (1, 'CE_KM_DATA', 'in');
insert into CE_KM_SIGNATURE values (2, 'CE_KM_PARAMS', 'in');
insert into CE_KM_SIGNATURE values (3, 'CE_KM_RESULTS', 'out');
insert into CE_KM_SIGNATURE values (4, 'CE_KM_CENTER_POINTS', 'out');

drop procedure _SYS_AFL.CE_KM;
drop type _SYS_AFL.CE_KM__TT_P1;
drop type _SYS_AFL.CE_KM__TT_P2;
drop type _SYS_AFL.CE_KM__TT_P3;
drop type _SYS_AFL.CE_KM__TT_P4;
call SYSTEM.AFL_WRAPPER_GENERATOR('CE_KM','AFLPAL','KMEANS', CE_KM_SIGNATURE);

-- Use the schema MIMIC2 for the following instructions
set schema MIMIC2;

-- Create a view to select specific fields from a table.
-- This allows a fine-grained control over what will be used in the algorithm instead of the complete tables
-- and also create relevant features that are not normally in the database
drop view V_KM_DATA;
create view V_KM_DATA as
        select "subject_id" as SUBJECT_ID, "value1num" as VALUE1NUM, "value2num" as VALUE2NUM
        from "MIMIC2"."mimic.tables::chartevents"
        where "value1num" is not null and "value2num" is not null;

-- Create a table for the function parameters
drop table #KM_PARAMS;
drop table KM_RESULTS;
drop table KM_CENTER_POINTS;
create local temporary column table #KM_PARAMS like _SYS_AFL.CE_KM_PARAMS;
create column table KM_RESULTS like _SYS_AFL.CE_KM_RESULTS;
create column table KM_CENTER_POINTS like _SYS_AFL.CE_KM_CENTER_POINTS;

-- Populate this parameters table
insert into #KM_PARAMS values('GROUP_NUMBER',15,null,null);
insert into #KM_PARAMS values('DISTANCE_LEVEL',2,null,null);
insert into #KM_PARAMS values('MAX_ITERATION',20,null,null);
insert into #KM_PARAMS values('INIT_TYPE',3,null,null);
insert into #KM_PARAMS values('NORMALIZATION',1,null,null);
insert into #KM_PARAMS values('THREAD_NUMBER',2,null,null);
insert into #KM_PARAMS values('EXIT_THRESHOLD',null,10.0,null);


-- Allows us to try different value for the parameters
update #KM_PARAMS set INTARGS=15 where name='GROUP_NUMBER';
update #KM_PARAMS set INTARGS=2 where name='DISTANCE_LEVEL';
update #KM_PARAMS set INTARGS=0 where name='NORMALIZATION';
update #KM_PARAMS set DOUBLEARGS=0.0001 where name='EXIT_THRESHOLD';


-- Empty the 'out' tables before running the KMeans function
truncate table KM_RESULTS;
truncate table KM_CENTER_POINTS;

call _SYS_AFL.CE_KM(V_KM_DATA, KM_PARAMS, KM_RESULTS, KM_CENTER_POINTS) with
OVERVIEW;
```

b. Anomaly Detection

```
-- Tells the system to use the schema _SYS_AFL. This is more or less similar to a namespace. Everything
following this will apply to this schema
set schema _SYS_AFL;


-- Here lies the data table for the attributes that ANOMALYDETECTION will use. Can put as many
parameters as required
drop type CE_AD_DATA;
create type CE_AD_DATA as table (SUBJECT_ID INTEGER, VALUE1NUM DOUBLE, VALUE2NUM
DOUBLE);

-- Regular table used by every PAL function. This is filled later on
drop type CE_AD_PARAMS;
create type CE_AD_PARAMS as table (NAME VARCHAR (50), INTARGS INTEGER, DOUBLEARGS
DOUBLE, STRINGARGS VARCHAR (100));

-- Output table for the results
drop type CE_AD_RESULTS;
create type CE_AD_RESULTS as table (SUBJECT_ID INTEGER, VALUE1NUM DOUBLE, VALUE2NUM
DOUBLE);

-- Signature table that the PAL function will use
drop table CE_AD_SIGNATURE;
create column table CE_AD_SIGNATURE (ID INTEGER, TYPENAME VARCHAR(100), DIRECTION
VARCHAR(100));
insert into CE_AD_SIGNATURE values (1, 'CE_AD_DATA', 'in');
insert into CE_AD_SIGNATURE values (2, 'CE_AD_PARAMS', 'in');
insert into CE_AD_SIGNATURE values (3, 'CE_AD_RESULTS', 'out');

-- Create the procedure
drop procedure _SYS_AFL.CE_AD;
drop type _SYS_AFL.CE_AD__TT_P1;
drop type _SYS_AFL.CE_AD__TT_P2;
drop type _SYS_AFL.CE_AD__TT_P3;
call SYSTEM.AFL_WRAPPER_GENERATOR('CE_AD','AFLPAL','ANOMALYDETECTION',
CE_AD_SIGNATURE);

-- Use the schema MIMIC2 for the following instructions
set schema MIMIC2;

-- Create a view to select specific fields from a table.
-- This allows a fine-grained control over what will be used in the algorithm instead of the complete tables
-- and also create relevant features that are not normally in the database
drop view V_AD_DATA;
create view V_AD_DATA as
```

```
        select "subject_id" as SUBJECT_ID, "value1num" as VALUE1NUM, "value2num" as VALUE2NUM
        from "MIMIC2"."mimic.tables::chartevents"
        where "value1num" is not null and "value2num" is not null;

-- Create a table for the function parameters
drop table #AD_PARAMS;
drop table AD_RESULTS;
create local temporary column table #AD_PARAMS like _SYS_AFL.CE_AD_PARAMS;
create column table AD_RESULTS like _SYS_AFL.CE_AD_RESULTS;

-- Populate this parameters table
insert into #AD_PARAMS values('GROUP_NUMBER',15,null,null);
insert into #AD_PARAMS values('DISTANCE_LEVEL',2,null,null);
insert into #AD_PARAMS values('OUTLIER_PERCENTAGE',null,0.05,null);
insert into #AD_PARAMS values('OUTLIER_DEFINE',1,null,null);
insert into #AD_PARAMS values('MAX_ITERATION',20,null,null);
insert into #AD_PARAMS values('INIT_TYPE',3,null,null);
insert into #AD_PARAMS values('NORMALIZATION',1,null,null);
insert into #AD_PARAMS values('THREAD_NUMBER',2,null,null);
insert into #AD_PARAMS values('EXIT_THRESHOLD',null,10.0,null);


-- Allows us to try different value for the parameters
update #AD_PARAMS set INTARGS=15 where name='GROUP_NUMBER';
update #AD_PARAMS set INTARGS=2 where name='DISTANCE_LEVEL';
update #AD_PARAMS set INTARGS=0 where name='NORMALIZATION';
update #AD_PARAMS set DOUBLEARGS=0.0001 where name='EXIT_THRESHOLD';


-- Empty the 'out' tables before running the KMeans function
truncate table AD_RESULTS;

call _SYS_AFL.CE_AD(V_AD_DATA, #AD_PARAMS, AD_RESULTS) with OVERVIEW;
```

Cookbook Chapter 3 Translated to HANA SQL

1. Insuline Doses

```
select distinct po."doses_per_24hrs", pm."dose_val_rx"
from "MIMIC2"."mimic.tables::poe_order" po, "MIMIC2"."mimic.tables::poe_med" pm
where po."poe_id" = pm."poe_id" and lower(po."medication") like '%insulin%' and
lower(pm."drug_name_generic") like '%insulin%'
order by po."doses_per_24hrs", pm."dose_val_rx";

-- Find the first ICU admission
select *
from
        (select min("intime") over (partition by "subject_id") as min_intime,
        ie.* from "MIMIC2"."mimic.tables::icustayevents" ie)
where min_intime = "intime"
```

Cookbook Chapter 4 Translated to HANA SQL

1. Elixhauser Comorbidities