

School of Computer Science

Data Wrangling in Fulfilment of

DATA9910

Maksymilian Drzezdzon

C15311966

Degree: TU060/1

Module Coordinator: Brendan Tierney

Declaration of Ownership: I declare that the attached work is entirely my own and that all sources have been acknowledged.

**Date: 2020/11/25**

Section A – Data Analysis

This section will focus on examining a medical insurance dataset of 1338 rows and 7 columns.[1] Collected variables include age, sex, BMI, number of children, does the person smoke, region of residence within the US and their annual insurance charges.

1. CREATE TABLE INSURANCE
2. (
3. AGE NUMBER(38,0),
4. SEX VARCHAR(50),
5. BMI NUMBER(38,3),
6. CHILDREN NUMBER(38,0),
7. SMOKER VARCHAR(50),
8. REGION VARCHAR(50) ,
9. CHARGES NUMBER(38,5)
10. );

Script to create and import table into SQL developer.

The main question for this part of the analysis is “what factors affect medical insurance costs the most”. With a limited amount of time for each section only a few areas will be explored to provide some insight meanwhile showcasing SQLs analytical functions with hypothesis testing, correlations and descriptive analytics etc.

Results for each query will be added within the code snippet for single row and column results using a /\* result here\*/ or with another snippet for multi row and column results, adding limit 1 to a query that only returns a single value is redundant.

A five-number summary is a set of descriptive statistics used to analyse a dataset; this was completed on each column of numeric data. A five-number summary consists of the min, max, median and first and third quartiles of data, a template would look like:

The next five queries are part of the five-number summary and are run against each numeric column to get a better understanding of the dataset, the column below is the charges column, representing annual medical insurance charges.

select round(max(charges),2) "max charges cost" from insurance; /\* 63,770.43 dollars \*/

The function above selects the charges from the insurance table, gets the highest value and rounds it to the 2nd decimal place and slaps on a “max charges cost” label.

select round(min(charges),2) "min charges cost" from insurance; /\* 1,121.87 dollars \*/

The function selects the charges from the insurance table, gets the lowest value and rounds it to the 2nd decimal place and adds a “min charges cost” label.

select median(charges) "middle charges cost value" from insurance; /\* 9,382.03 dollars \*/

The function above selects the charges from the insurance table, gets the median value which separates the second and third quartiles, then adds a “middle charges cost” label.

The above three queries were just used to get an idea of potential outliers.

1. select round(sum(charges), 2)"insurance costs in first quartile" from (select charges, ntile(4) over(order by charges) as QUARTILE from insurance) where quartile = 1; /\* 955,784.96 dollars \*/

2 select round(sum(charges), 2)"insurance costs in third quartile" from (select charges, ntile(4) over(order by charges) as QUARTILE from insurance) where quartile = 3; /\* 4,050,700.59 dollars \*/

The two above queries sum up and round to two decimal places any data that falls into the bottom twenty five percent. This is completed by the nested query that selects the charges column and puts it into four buckets with ntile(4) followed by selecting the charges rows with the over() function and the ordering it by charges from the insurance table. Lastly its tagged as quartile and used to grab the first and third quartile from the four buckets with where quartile is equal to three. An appropriate label is added to each resulting column.

The goal of this query was to get a more tangible idea of how much of the overall cost is under which quartile, if sum() wasn’t used the query would return a list of all records with an ntile number which isn’t interpretable.

3. select STATS\_ONE\_WAY\_ANOVA(bmi, charges, 'F\_RATIO') f\_ratio from insurance; /\*1.11\*/

/\*an f ratio close to 1 means that the null hypothesis is true, meaning there is no correlation between the two\*/

4. select STATS\_ONE\_WAY\_ANOVA(age, charges, 'SIG') p\_value from insurance; /\* 0.0000000000000001 \*/

/\* a p-value of more than 0.05 is not statistically significant meaning there is evidence for age having an impact on health insurance charges\*/

 These two queries where used to calculate the significance toward finding out what values have a statistical significance on insurance costs. However, what was a profound predictor of medical insurance was age and if the person smoked or not, both yielding a p value of close to 0. [3]

update insurance set smoker = 0 where smoker = 'no';

update insurance set smoker = 1 where smoker = 'yes';

5. select STATS\_ONE\_WAY\_ANOVA(smoker, charges, 'SIG') p\_value from insurance;

update insurance set smoker = 'no' where smoker = '0';

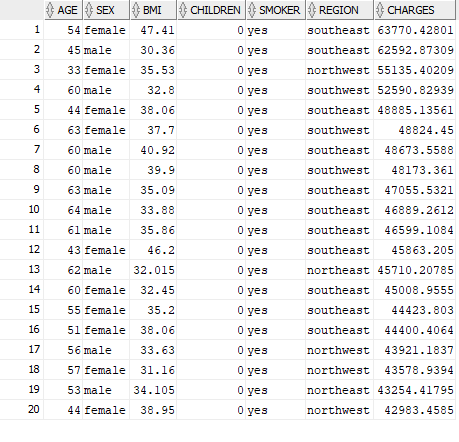
update insurance set smoker = 'yes' where smoker = '1';

 A quick test was run to make sure that the values yes and no were being interpreted properly by SQL and were converted to a 1 or 0 for the one-way anova, the result was still statistically significant

6. select STATS\_F\_TEST(smoker, charges) two\_tailed\_test from insurance; /\* 0.0000000000000000000000000000000000000000000000000003 \*/

This is the further illustrated by the two tailed test, a two tailed test ranges from 0 to 1 the closer the result is to 0 the more statistically significant it is that smoking does have an impact on medical charge costs. Alternately the ‘TWO\_SIDED\_SIG’ argument can be used which would yield the same result as a two tailed test is run by default when no test parameter is passed in. [2] The goal here was to confirm findings from the one-way ANOVA with a two tailed test.

select \* from insurance order by charges desc fetch next 20 rows only;



The top 20 highest medical insurance charges are all middle aged or older smokers, with no kids. The goal was to see if the above statistics are true and what parameters each individual would have when paying a premium on insurance.

The query selects all rows from the insurance table and orders it by the charges column limiting it to 20 rows, originally the query had children in it but as all individuals have no kids that variable was then removed as its redundant.

7. select PERCENT\_RANK(50, 'yes') within group

(ORDER BY age, smoker)

from insurance; /\*0.73\*/

7.5 select PERCENT\_RANK(50, 'no') within group

(ORDER BY age, smoker)

from insurance; /\*0.71\*/

 The goal of the above query was to see the significance of smoking on an individual ranking trying this out across a range of ages its usually 1-3%. The query takes a percent rank of an individual at the age of 50 that smokes from the group ordered by age and smoke columns from the insurance table.

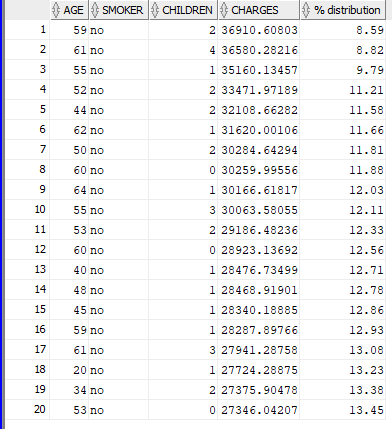
8. select age, smoker, children, charges,

round(cume\_dist() over (order by charges desc)\*100,2)"% distribution"

from insurance

order by smoker asc, "% distribution" asc

fetch next 20 rows only;



The above query grabs a cumulative distribution of individuals by their medical charges, smoking habits and % of distribution in descending order. The goal was to validate previous findings.

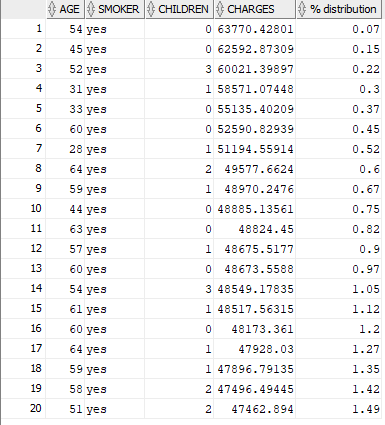
9. select age, smoker, children, charges,

round(cume\_dist() over (order by charges desc)\*100,2)"% distribution"

from insurance

order by charges desc, "% distribution" asc

fetch next 20 rows only;



When results are sorted by charges rather than whether someone is a smoker or not, the difference in medical charges almost doubles in some cases. Individuals in these groups cover a low amount of the overall however

10. select age, smoker, children, charges,

round(cume\_dist() over (order by charges desc)\*100,2)"% distribution"

from insurance

where (age = 18 and smoker = 'yes')

order by charges desc

fetch next 20 rows only;

select age, smoker, children, charges,

round(cume\_dist() over (order by charges desc)\*100,2)"% distribution"

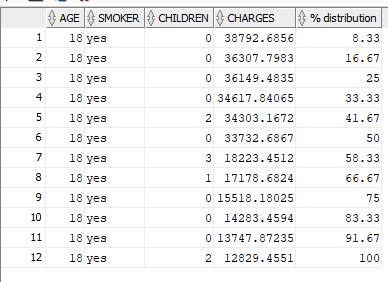
from insurance

where (age = 18 and smoker = 'no')

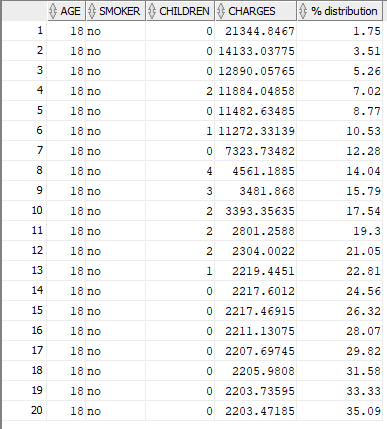
order by charges desc

fetch next 20 rows only;

when compared to eighteen-year-olds that smoke, which there are fewer categories and a much higher distribution.



When compared to the same group but instead that dont smoke, its much more spread out with lower distributions. The goal of these last few queries was to get an idea of how many people are paying more for insurance charges and have unhealthy habits, smoking and BMI were used to asses that. Maybe a campaign that offered younger customers health insurance discounts if they didn’t smoke could incentivise people to look after their health and save money. This might not be at the benefit of the insurance company but it’s an observation nonetheless.



Section B – Data Audit Report

A table was created with the following, some names had to be change because they interfered with SQL command names, such as job was changed to job\_type, day was changed to cday for campaign day etc etc.

The dataset used is from a bank telemarketing campaign that tries to sell long term deposits as a means of bouncing back from the 2008 recession.

CREATE TABLE BANK

(

age NUMBER,

job\_type VARCHAR(255),

marital VARCHAR(255),

education VARCHAR(255),

credit\_default VARCHAR(255),

balance NUMBER,

housing VARCHAR(255),

loan VARCHAR(255),

contact VARCHAR(255),

cday VARCHAR(255),

cmonth VARCHAR(255),

cduration NUMBER,

campaign NUMBER,

pdays NUMBER,

previous NUMBER,

poutcome VARCHAR(255),

y VARCHAR(255)

);

 Tasks:

Each column needs to be checked for incompleteness, empty values, is data consistent? Does it need transformations? Does it have dark data (can’t be used because it isn’t good enough).

set SERVEROUTPUT ON;

needs to be set in order for print output to be visible in the console for analysis.

This is the script that was used to asses’ data for each column, replacing age with marital, education, job\_type. The format below was used to asses values in the dataset, because most of these columns are categorical its easy to search for unique input and check for anomalies.

declare cursor cursor1

is

select distinct age from bank order by age;

record1 cursor1%rowtype;

begin

for record1 in cursor1 loop

DBMS\_OUTPUT.PUT\_line('data returned ' || record1.age);

end loop;

end;

Job categories returned admin.

Job categories returned blue-collar

Job categories returned entrepreneur

Job categories returned housemaid

Job categories returned management

Job categories returned retired

Job categories returned self-employed

Job categories returned services

Job categories returned student

Job categories returned technician

Job categories returned unemployed

Job categories returned unknown

Marital status returned divorced

Marital status returned married

Marital status returned single

Education returned primary

Education returned secondary

Education returned tertiary

Education returned unknown

Through this process data was assessed for consistency, any outliers etc. No inconsistencies, formatting errors or empty values were found, meaning that it has passed the first check. From this procedure it was found that the contact, cday, cmonth, campaign, pdays columns could be dropped and potentially reduce the storage needed as those field aren’t very useful and would qualify for being dark data. [4] Contact details are kept in order to contact customers but for the sake of training models they don’t provide any use.

If need be they can be removed with:

Alter Table bank

Drop column <col\_name\_from above>

Alternately for data cleaning rows with an Unknown status can be deleted with

Delete from bank where education = ‘unknown’;

Delete from bank where job\_type = ‘unknown’;

The admin occupation was cleaned up as it has a ‘.’ At the end of it for the sake of consistency with the command below.

UPDATE bank

SET job\_type = 'admin'

WHERE job\_type = 'admin.';

/\* 5,171 rows updated. \*/

declare

sig number;

mean number := 1;

stdev number := 1;

begin

SYS.dbms\_stat\_funcs.normal\_dist\_fit ('mdrzezdzon', 'bank', 'age', 'KOLMOGOROV\_SMIRNOV', mean, stdev, sig);

SYS.dbms\_stat\_funcs.POISSON\_DIST\_FIT ('mdrzezdzon', 'bank', 'age', 'KOLMOGOROV\_SMIRNOV', stdev, sig);

SYS.dbms\_stat\_funcs.normal\_dist\_fit ('mdrzezdzon', 'bank', 'balance', 'KOLMOGOROV\_SMIRNOV', mean, stdev, sig);

SYS.dbms\_stat\_funcs.POISSON\_DIST\_FIT ('mdrzezdzon', 'bank', 'balance', 'KOLMOGOROV\_SMIRNOV', stdev, sig);

SYS.dbms\_stat\_funcs.normal\_dist\_fit ('mdrzezdzon', 'bank', 'cduration', 'KOLMOGOROV\_SMIRNOV', mean, stdev, sig);

SYS.dbms\_stat\_funcs.POISSON\_DIST\_FIT ('mdrzezdzon', 'bank', 'cduration', 'KOLMOGOROV\_SMIRNOV', stdev, sig);

end;

|  |  |  |
| --- | --- | --- |
| Columns | Normal Distribution | Poisson Distribution |
| Age | 0.99 | 0.99 |
| Balance | 0.83 | 0.82 |
| Cduration | 0.99 | 0.99 |

Later columns were tested on how well they are distributed by fitting a normal and poisson distribution, this is done for the next section to make check data quality for modeling. It seems that data is normally distributed, there are no missing values, the balance column is of less importance but it was checked for anomalies.

Its worth mentioning that you can also gather performance statistics among others but that is out of scope for this project, an example is added below.

declare

begin

dbms\_stats.gather\_table\_stats('mdrzezdzon', 'bank', estimate\_percent => dbms\_stats.auto\_sample\_size);

end;

DECLARE

v\_age bank.age%type;

v\_mode\_age bank.age%type;

v\_avg\_age bank.age%type;

v\_job\_type bank.job\_type%type;

v\_marital bank.marital%type;

begin

select round(STDDEV(age),2)

into v\_age

from bank;

select round(stats\_mode(age),2)

into v\_mode\_age

from bank;

select round(AVG(age),2)

into v\_avg\_age

from bank;

select stats\_mode(job\_type)

into v\_job\_type

from bank;

select stats\_mode(marital)

into v\_marital

from bank;

DBMS\_OUTPUT.put\_line('Standard of deviation Age: ' || v\_age);

DBMS\_OUTPUT.put\_line('Average Age: ' || v\_avg\_age);

DBMS\_OUTPUT.put\_line('Most Common Age: ' || v\_mode\_age);

DBMS\_OUTPUT.put\_line('Most Common Job: ' || v\_job\_type);

DBMS\_OUTPUT.put\_line('Most Common Marital status: ' || v\_marital);

end;

Standard of deviation Age: 10.62

Average Age: 40.94

Most Common Age: 32

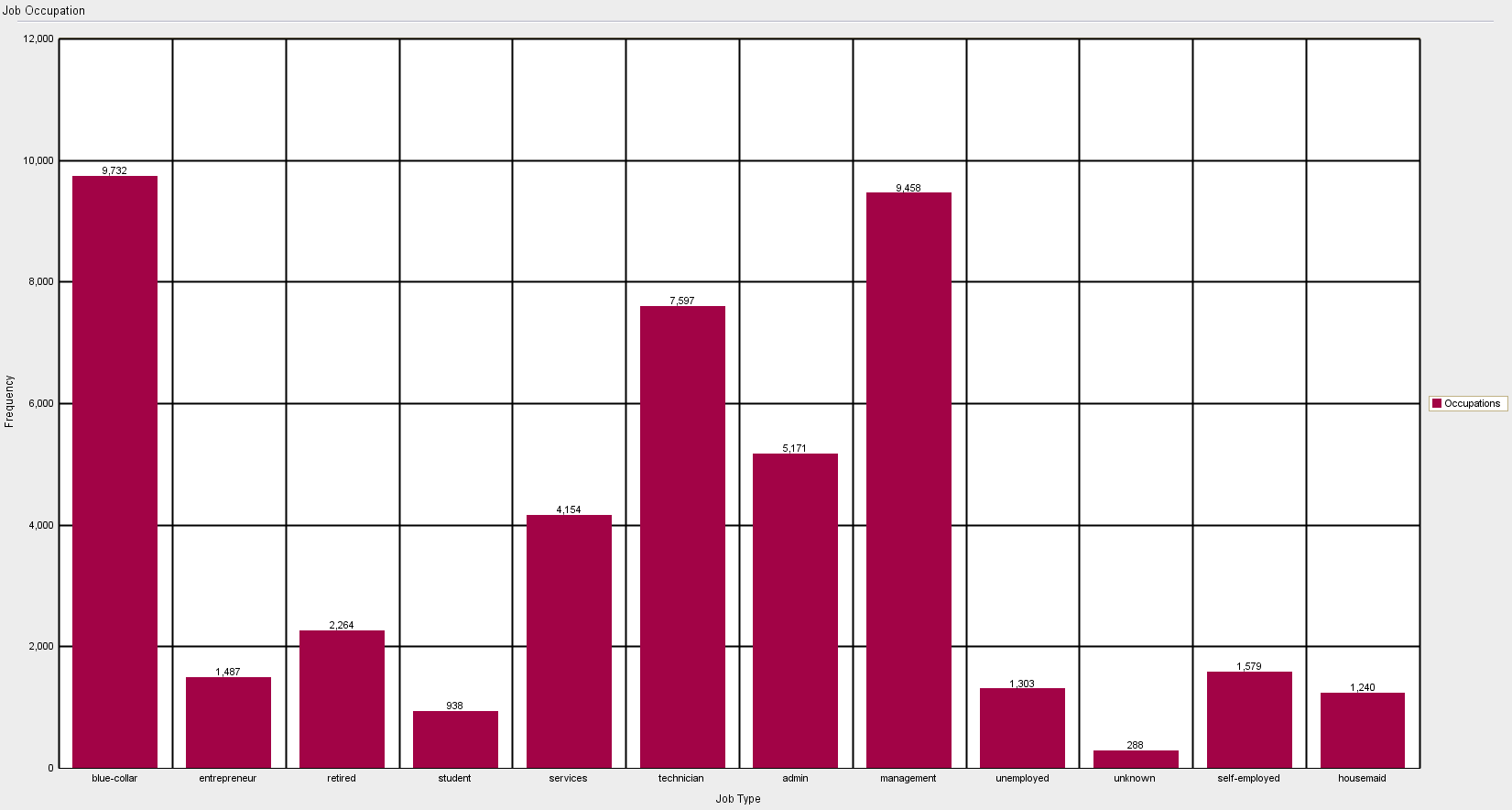
Most Common Job: blue-collar

Most Common Marital status: married

A useful tool I found while working on this report is the SQL developers report function, I won’t be focusing on it too much because I don’t think it applies to the requirements of the project but I think its worth mentioning, the bar graph was generated with SQL via SQL developer, I think it’s a better way to assess the distribution of categorical data.

Usually the 'Occupations' wouldn’t be used but in order to generate the report an additional column must be added to map the count(job\_type) result and job\_type data.

select job\_type, 'Occupations', count(job\_type) from bank group by job\_type;



In PL/SQL it would look like

set serveroutput on;

DECLARE

begin

DBMS\_OUTPUT.PUT\_LINE('');

DBMS\_OUTPUT.PUT\_LINE('Job Type Distribution');

for rec in (select job\_type, round((Count(job\_type)\* 100 / (Select Count(\*) From bank)), 2)"%" from bank group by job\_type) loop

DBMS\_OUTPUT.PUT\_LINE('| ' || rec.job\_type || ' | %' || rec."%" || '|');

end loop;

DBMS\_OUTPUT.PUT\_LINE('');

DBMS\_OUTPUT.PUT\_LINE('Marital Status Distribution');

for rec in (select marital, round((Count(marital)\* 100 / (Select Count(\*) From bank)), 2)"%" from bank group by marital) loop

DBMS\_OUTPUT.PUT\_LINE('| ' || rec.marital || ' | %' || rec."%" || '|');

end loop;

DBMS\_OUTPUT.PUT\_LINE('');

DBMS\_OUTPUT.PUT\_LINE('Education Distribution');

for rec in (select education, round((Count(education)\* 100 / (Select Count(\*) From bank)), 2)"%" from bank group by education) loop

DBMS\_OUTPUT.PUT\_LINE('| ' || rec.education || ' | %' || rec."%" || '|');

end loop;

end;

Results are below

Job Type Distribution

| blue-collar | %21.48|

| retired | %4.97|

| entrepreneur | %3.27|

| student | %1.79|

| services | %9.27|

| technician | %17.03|

| admin | %11.58|

| management | %21.34|

| unemployed | %2.95|

| self-employed | %3.57|

| housemaid | %2.77|

Marital Status Distribution

| married | %60.07|

| divorced | %11.64|

| single | %28.29|

Education Distribution

| tertiary | %30.7|

| primary | %15.74|

| secondary | %53.55|

There doesn’t seem to be anything unusual with the percentage distributions which makes it ok to use for analysis, the unknown values were removed as its data that isn’t very useful, other variables potentially could have been used to guess each unknown occupation, however these rows were few in number so they were just removed from the dataset.

Section C – Machine Learning

Tasks create 3 models use 4 different algorithms, prepare data and create models [5][7]

Setting notes and configs can be found here [9] by searching for the algorithm or the algorithm name plus the settings keyword to see what options need to put added to the settings table. There isn’t a way for me to link each block of text to each algorithm in the report as oracles documentation does not have anchors for each table.

First before a model can be created, an ID column needs to exist, in this case it does not so it needs to be created with a sequence and the next value function and added to the existing bank table like so

CREATE SEQUENCE seq\_bank;

alter table bank add(

row\_id integer default seq\_bank.nextval

);

Data preparation can be done with the following pl/sql command.

BEGIN

EXECUTE IMMEDIATE

'CREATE OR REPLACE VIEW

bank\_train\_data AS

SELECT \* FROM bank

SAMPLE (60) SEED (42)';

EXECUTE IMMEDIATE

'CREATE OR REPLACE VIEW

bank\_test\_data AS

SELECT \* FROM bank

MINUS

SELECT \* FROM bank\_train\_data';

END;

The above command creates two views that split the dataset into train 60% of data and test 40% of data.

Following that the target variable which is if the customer got a long-term deposit account need to be numerical, since this datasets target variable is binary it can be changed to a 1 or 0 as of right now it’s a yes or no. This is changed with the following SQL command

update bank set y = 0 where y = 'no';

update bank set y = 1 where y = 'yes';

Then a settings table is created to store config

create table bank\_model\_settings(

setting\_name varchar2(30),

setting\_value varchar2(4000)

);

All steps above are done once as preparation before building any models can start.

**GLM-Regression – Generalized Linear Model**

Then fill settings table.

begin

insert into bank\_model\_settings values(dbms\_data\_mining.ALGO\_NAME, dbms\_data\_mining.ALGO\_GENERALIZED\_LINEAR\_MODEL);

-- ROW DIAGNOSTICS

insert into bank\_model\_settings values(dbms\_data\_mining.GLMS\_DIAGNOSTICS\_TABLE\_NAME, 'GLMS\_BANK\_DIAG');

-- DATA PREP

insert into bank\_model\_settings values(dbms\_data\_mining.PREP\_AUTO, dbms\_data\_mining.PREP\_AUTO\_ON);

-- FEATURE SELECTION

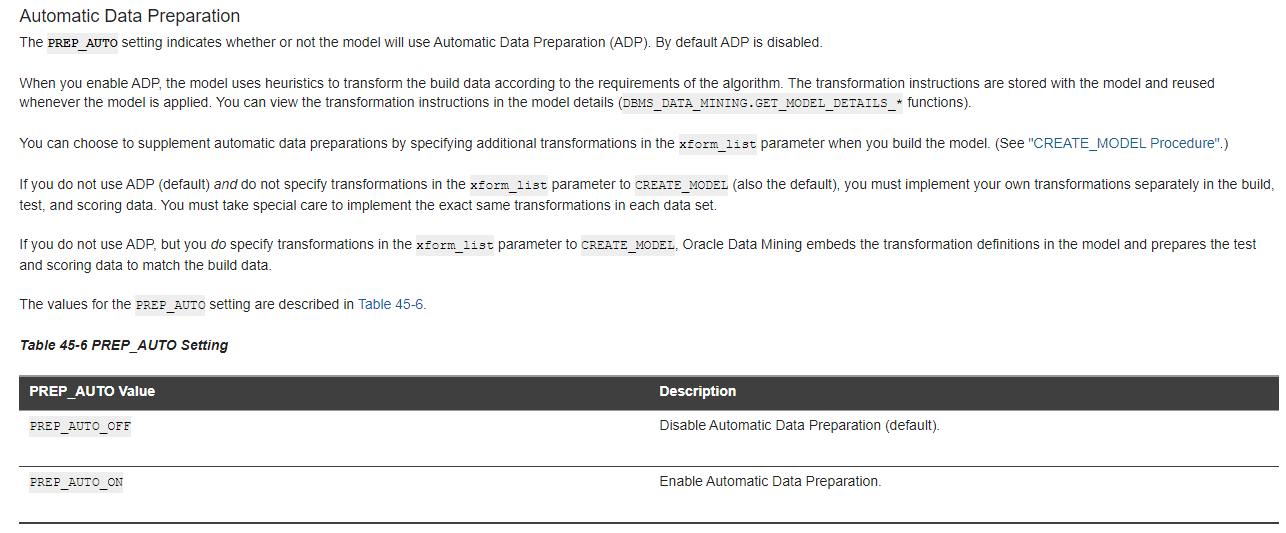
insert into bank\_model\_settings values(dbms\_data\_mining.GLMS\_FTR\_SELECTION, dbms\_data\_mining.GLMS\_FTR\_SELECTION\_ENABLE);

-- FEATURE GENERATION

insert into bank\_model\_settings values(dbms\_data\_mining.GLMS\_FTR\_GENERATION, dbms\_data\_mining.GLMS\_FTR\_GENERATION\_ENABLE);

end;

Then using oracles dbms\_data\_mining library you can then use auto preparation to handle splitting, feature selection etc which makes it much easier to setup.[6]

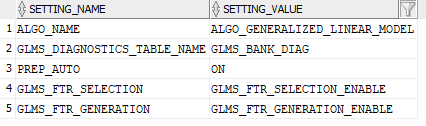


Settings can be inspected with. (just to be sure this table will be wiped clean so that each model is built with the specified config) [8] the mining function can be found here [10] as can be seen here



However, the selected function for GLM models returns the same level of statistics per row thus another one was used to report it in one table for the whole model below.

select \* from bank\_model\_settings;



After that a model can be created with

begin

dbms\_data\_mining.create\_model(

model\_name => 'GLM\_REGRESSION\_BANK',

mining\_function => dbms\_data\_mining.REGRESSION,

data\_table\_name => 'bank\_train\_data',

case\_id\_column\_name => 'row\_id',

target\_column\_name => 'y',

settings\_table\_name => 'bank\_model\_settings'

);

end;

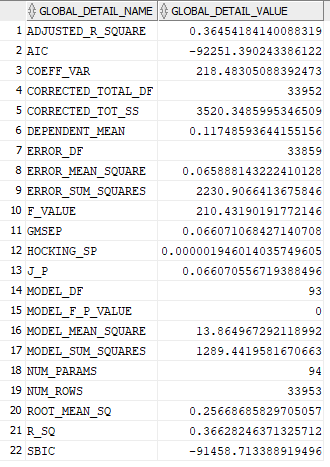
To get results from the model just created the following SQL command is used

select \*

from table(dbms\_data\_mining.get\_model\_details\_global('GLM\_REGRESSION\_BANK'))

order by global\_detail\_name;

Which yields the following results table.



In this scenario the model is wrong 25-26% of the time indicated by the root mean SQ, also applying the model to the test further can be done with

begin

dbms\_data\_mining.apply(

'GLMR\_REGRESSION\_BANK',

'bank\_test\_data',

'row\_id',

'bank\_test\_predictions'

);

end;

select \* from bank\_test\_predictions;

**GLM-Classification – Generalized Linear Model**

All settings are the same as above apart from my mining function

begin

dbms\_data\_mining.create\_model(

model\_name => 'GLMC\_REGRESSION\_BANK',

mining\_function => dbms\_data\_mining.Classification,

data\_table\_name => 'bank\_train\_data',

case\_id\_column\_name => 'row\_id',

target\_column\_name => 'y',

settings\_table\_name => 'bank\_model\_settings'

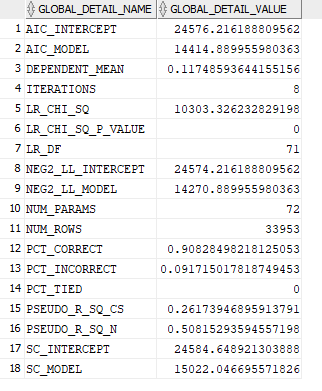
);

end;

select \*

from table(dbms\_data\_mining.get\_model\_details\_global('GLMC\_REGRESSION\_BANK'))

order by global\_detail\_name;



**Neural Network**

The bank model settings table was emptied with delete from bank\_model\_settings and then filled again with the correct config for the neural net, this model has a lot of default values that are usually the same in python so those were left be.

begin

insert into bank\_model\_settings values(dbms\_data\_mining.ALGO\_NAME, dbms\_data\_mining.ALGO\_NEURAL\_NETWORK);

-- DATA PREP

insert into bank\_model\_settings values(dbms\_data\_mining.PREP\_AUTO, dbms\_data\_mining.PREP\_AUTO\_ON);

end;

The model is then created with

begin

dbms\_data\_mining.create\_model(

model\_name => 'NEURAL\_NETWORK\_BANK',

mining\_function => dbms\_data\_mining.Classification,

data\_table\_name => 'bank\_train\_data',

case\_id\_column\_name => 'row\_id',

target\_column\_name => 'y',

settings\_table\_name => 'bank\_model\_settings'

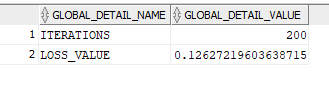
);

end;

reviewed with

select \*

from table(dbms\_data\_mining.get\_model\_details\_\*('NEURAL\_NETWORK\_BANK'));



Prepare test sample with

begin

dbms\_data\_mining.apply(

'NEURAL\_NETWORK\_BANK',

'bank\_test\_data',

'row\_id',

'NN\_bank\_test\_predictions'

);

end;

Evaluated for accuracy with

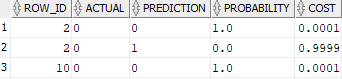
select

NN\_bank\_test\_predictions.row\_id, bank.y as actual, NN\_bank\_test\_predictions.prediction as prediction,

round(NN\_bank\_test\_predictions.probability) as probability, round(NN\_bank\_test\_predictions.cost, 4) as cost

from NN\_bank\_test\_predictions

inner join bank on NN\_bank\_test\_predictions.row\_id=bank.row\_id;



And then compared with original data to see model accuracy with

with bank\_data as

(select count(row\_id) as bank\_row\_count from bank),

model\_data as

(select count(\*) as model\_row\_count

from NN\_bank\_test\_predictions

inner join bank on NN\_bank\_test\_predictions.row\_id = bank.row\_id

where bank.y = NN\_bank\_test\_predictions.prediction and round(NN\_bank\_test\_predictions.probability) = 1)

select round(model\_row\_count/bank\_row\_count, 2) as Model\_Accuracy from bank\_data, model\_data;

This Neural Net has an approx. accuracy of 22% when the probability column is accounted for and 25% when it isn’t with

and round(NN\_bank\_test\_predictions.probability) = 1).



**Naive Bayes**

The bank model settings table was emptied with delete from bank\_model\_settings and then filled again with the correct config for Naïve Bayes, this model has a lot of default values that are usually the same in python so those were left be.

begin

insert into bank\_model\_settings values(dbms\_data\_mining.ALGO\_NAME, dbms\_data\_mining.ALGO\_NAIVE\_BAYES);

-- DATA PREP

insert into bank\_model\_settings values(dbms\_data\_mining.PREP\_AUTO, dbms\_data\_mining.PREP\_AUTO\_ON);

end;

begin

dbms\_data\_mining.create\_model(

model\_name => 'NAIVE\_BAYES\_BANK',

mining\_function => dbms\_data\_mining.Classification,

data\_table\_name => 'bank\_train\_data',

case\_id\_column\_name => 'row\_id',

target\_column\_name => 'y',

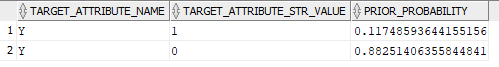
settings\_table\_name => 'bank\_model\_settings'

);

end;

select target\_attribute\_name, target\_attribute\_str\_value, prior\_probability

from table(dbms\_data\_mining.GET\_MODEL\_DETAILS\_NB('NAIVE\_BAYES\_BANK'));



begin

dbms\_data\_mining.apply(

'NAIVE2\_BAYES\_BANK',

'bank\_test\_data',

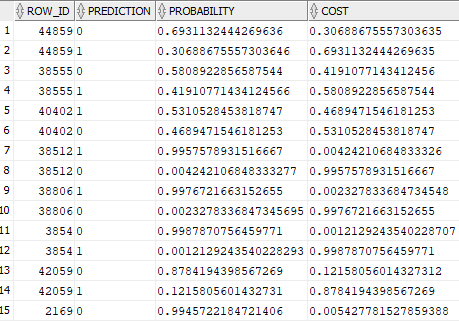
'row\_id',

'NB\_bank\_test\_predictions'

);

end;

select \* from NB\_bank\_test\_predictions fetch first 15 rows only;



Testing models had to be revaluated because I wasn’t able to use SQL developers Data Mining capabilities via creating a test node and using built in evaluation features since I didn’t have the module installed, I was able to get past all blocks except the sys admin password, a significant amount of time was spent tying to edit it, create password files etc in order to set it up to evaluate models using these features, but in the end all that work was void.

Instead, models were evaluated with SQL commands.

XGBoost

Support Vector Machine

https://docs.oracle.com/en/database/oracle/oracle-database/21/arpls/DBMS\_DATA\_MINING.html#GUID-12408982-E738-4D0F-A2BC-84D895E07ABB

Appendix

Oracle 1996, 2007 <https://docs.oracle.com/database/121/SQLRF/functions186.htm#SQLRF06318> [2]

<https://docs.oracle.com/database/121/SQLRF/functions190.htm#SQLRF06322> [3]

<https://docs.oracle.com/en/database/oracle/machine-learning/oml4sql/21/dmprg/prepare-data.html#GUID-E1AB599C-1921-4BD7-B06B-FC466180A460> [5]

<https://docs.oracle.com/database/121/ARPLS/d_datmin.htm#ARPLS608> [6]

<https://docs.oracle.com/en/database/oracle/machine-learning/oml4sql/21/dmprg/oml4sql-data-dictionary-vews.html#GUID-06AF74F5-39D7-4B0F-996E-35CA4C904EA0> [7]

<https://docs.oracle.com/en/database/oracle/machine-learning/oml4sql/21/dmprg/oml4sql-data-dictionary-vews.html#GUID-8262B256-1DFD-40C1-B56C-8E391B5AA303> [8]

<https://docs.oracle.com/database/121/ARPLS/d_datmin.htm#ARPLS609> [9]

<https://docs.oracle.com/en/database/oracle/machine-learning/oml4sql/21/dmprg/CREATE_MODEL-procedure.html#GUID-705252EF-2DCA-49D4-A49B-454A308CCDCD> [10]

<https://docs.oracle.com/cd/E55747_01/doc.41/e58114/test.htm#DMRUG816[11>]

<https://docs.oracle.com/cd/E55747_01/doc.41/e58114/evalapply.htm#DMRUG689> [12]

<https://docs.oracle.com/cd/E55747_01/doc.41/e58114/evalapply.htm#DMRUG718> [13]

<https://docs.oracle.com/cd/E55747_01/doc.41/e58114/test.htm#DMRUG854> [14]

<https://www.ibm.com/support/knowledgecenter/en/SSHRBY/com.ibm.swg.im.dashdb.apdv.plsql.doc/doc/c0053861.html>

https://docs.oracle.com/database/121/ARPLS/d\_datmin.htm#ARPLS65818

https://www.oracletutorial.com/oracle-analytic-functions/

References

Miri Choi, 2017, [https://www.kaggle.com/mirichoi0218/insurance [1](https://www.kaggle.com/mirichoi0218/insurance%20%5b1)]

Christopher Tozzi, January 2020 [https://www.precisely.com/blog/data-quality/how-to-measure-data-quality-7-metrics [4](https://www.precisely.com/blog/data-quality/how-to-measure-data-quality-7-metrics%20%5b4)]