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;
/* 0.00000000000000000000000000000000000000000000000003 */
```

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;
```

	AGE	SEX	BMI	CHILDREN	SMOKER	REGION	CHARGES
1	54	female	47.41	0	yes	southeast	63770.42801
2	45	male	30.36	0	yes	southeast	62592.87309
3	33	female	35.53	0	yes	northwest	55135.40209
4	60	male	32.8	0	yes	southwest	52590.82939
5	44	female	38.06	0	yes	southeast	48885.13561
6	63	female	37.7	0	yes	southwest	48824.45
7	60	male	40.92	0	yes	southeast	48673.5588
8	60	male	39.9	0	yes	southwest	48173.361
9	63	male	35.09	0	yes	southeast	47055.5321
10	64	male	33.88	0	yes	southeast	46889.2612
11	61	male	35.86	0	yes	southeast	46599.1084
12	43	female	46.2	0	yes	southeast	45863.205
13	62	male	32.015	0	yes	northeast	45710.20785
14	60	female	32.45	0	yes	southeast	45008.9555
15	55	female	35.2	0	yes	southeast	44423.803
16	51	female	38.06	0	yes	southeast	44400.4064
17	56	male	33.63	0	yes	northwest	43921.1837
18	57	female	31.16	0	yes	northwest	43578.9394
19	53	male	34.105	0	yes	northeast	43254.41795
20	44	female	38.95	0	yes	northwest	42983.4585

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;
```

	AGE	SMOKER	CHILDREN	CHARGES	% distribution
1	59	no	2	36910.60803	8.59
2	61	no	4	36580.28216	8.82
3	55	no	1	35160.13457	9.79
4	52	no	2	33471.97189	11.21
5	44	no	2	32108.66282	11.58
6	62	no	1	31620.00106	11.66
7	50	no	2	30284.64294	11.81
8	60	no	0	30259.99556	11.88
9	64	no	1	30166.61817	12.03
10	55	no	3	30063.58055	12.11
11	53	no	2	29186.48236	12.33
12	60	no	0	28923.13692	12.56
13	40	no	1	28476.73499	12.71
14	48	no	1	28468.91901	12.78
15	45	no	1	28340.18885	12.86
16	59	no	1	28287.89766	12.93
17	61	no	3	27941.28758	13.08
18	20	no	1	27724.28875	13.23
19	34	no	2	27375.90478	13.38
20	53	no	0	27346.04207	13.45

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;
```

	AGE	SMOKER	CHILDREN	CHARGES	% distribution
1	54	yes	0	63770.42801	0.07
2	45	yes	0	62592.87309	0.15
3	52	yes	3	60021.39897	0.22
4	31	yes	1	58571.07448	0.3
5	33	yes	0	55135.40209	0.37
6	60	yes	0	52590.82939	0.45
7	28	yes	1	51194.55914	0.52
8	64	yes	2	49577.6624	0.6
9	59	yes	1	48970.2476	0.67
10	44	yes	0	48885.13561	0.75
11	63	yes	0	48824.45	0.82
12	57	yes	1	48675.5177	0.9
13	60	yes	0	48673.5588	0.97
14	54	yes	3	48549.17835	1.05
15	61	yes	1	48517.56315	1.12
16	60	yes	0	48173.361	1.2
17	64	yes	1	47928.03	1.27
18	59	yes	1	47896.79135	1.35
19	58	yes	2	47496.49445	1.42
20	51	yes	2	47462.894	1.49

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.

	AGE	SMOKER	CHILDREN	CHARGES	% distribution
1	18	yes	0	38792.6856	8.33
2	18	yes	0	36307.7983	16.67
3	18	yes	0	36149.4835	25
4	18	yes	0	34617.84065	33.33
5	18	yes	2	34303.1672	41.67
6	18	yes	0	33732.6867	50
7	18	yes	3	18223.4512	58.33
8	18	yes	1	17178.6824	66.67
9	18	yes	0	15518.18025	75
10	18	yes	0	14283.4594	83.33
11	18	yes	0	13747.87235	91.67
12	18	yes	2	12829.4551	100

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.

	AGE	SMOKER	CHILDREN	CHARGES	% distribution
1	18	no	0	21344.8467	1.75
2	18	no	0	14133.03775	3.51
3	18	no	0	12890.05765	5.26
4	18	no	2	11884.04858	7.02
5	18	no	0	11482.63485	8.77
6	18	no	1	11272.33139	10.53
7	18	no	0	7323.73482	12.28
8	18	no	4	4561.1885	14.04
9	18	no	3	3481.868	15.79
10	18	no	2	3393.35635	17.54
11	18	no	2	2801.2588	19.3
12	18	no	2	2304.0022	21.05
13	18	no	1	2219.4451	22.81
14	18	no	0	2217.6012	24.56
15	18	no	0	2217.46915	26.32
16	18	no	0	2211.13075	28.07
17	18	no	0	2207.69745	29.82
18	18	no	0	2205.9808	31.58
19	18	no	0	2203.73595	33.33
20	18	no	0	2203.47185	35.09

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;
```

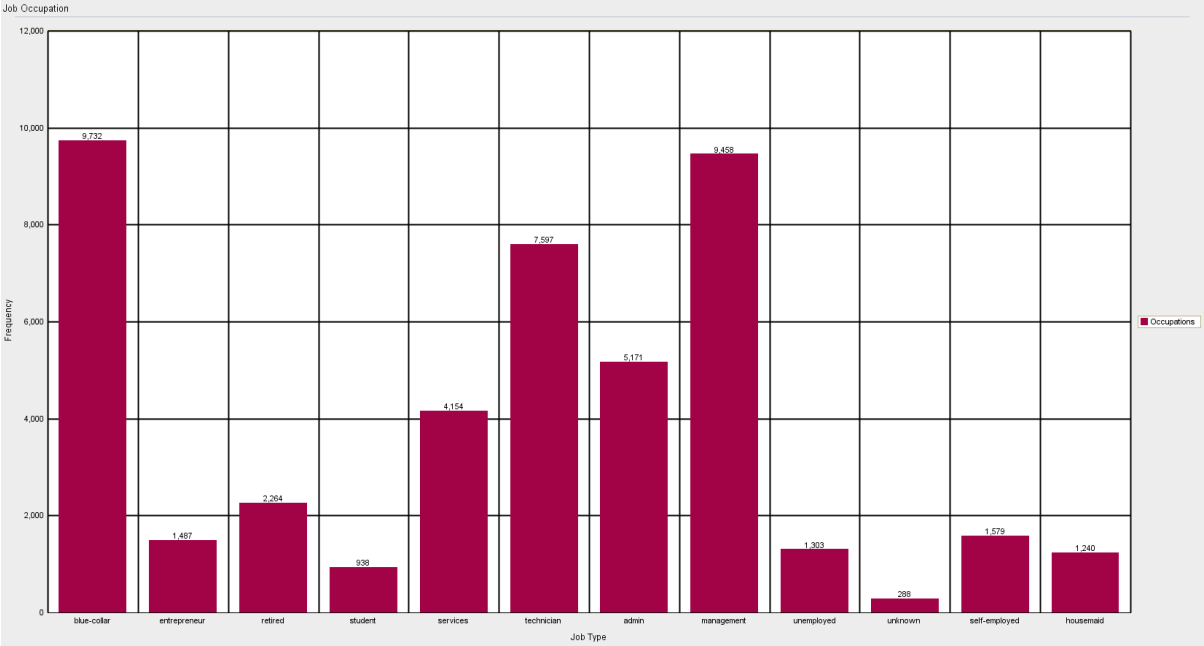
```
DECLARE
    v_age bank.age%type;
    v_mode_age bank.age%type;
    v_avg_age bank.age%type;
    v_job_type bank.job_type%type;
    v_marital bank.marital%type;

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 (~173 out of 48,000+) so they were just removed from the dataset. Percentages might differ by 0.01 as the round function was used for legibility.

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.

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 trying 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.[11][12][13][14]

Instead, models were evaluated with SQL commands. Potentially feature engineering functions could have been used to increase the performance gap between different models.

1. 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]

Automatic Data Preparation

The `PREP_AUTO` setting indicates whether or not the model will use Automatic Data Preparation (ADP). By default ADP is disabled.

When you enable ADP, the model uses heuristics to transform the build data according to the requirements of the algorithm. The transformation instructions are stored with the model and reused whenever the model is applied. You can view the transformation instructions in the model details (`DBMS_DATA_MINING.GET_MODEL_DETAILS_*` functions).

You can choose to supplement automatic data preparations by specifying additional transformations in the `xform_list` parameter when you build the model. (See "CREATE_MODEL Procedure".)

If you do not use ADP (default) *and* do not specify transformations in the `xform_list` parameter to `CREATE_MODEL` (also the default), you must implement your own transformations separately in the build, test, and scoring data. You must take special care to implement the exact same transformations in each data set.

If you do not use ADP, but you *do* specify transformations in the `xform_list` parameter to `CREATE_MODEL`, Oracle Data Mining embeds the transformation definitions in the model and prepares the test and scoring data to match the build data.

The values for the `PREP_AUTO` setting are described in Table 45-6.

Table 45-6 PREP_AUTO Setting

PREP_AUTO Value	Description
`PREP_AUTO_OFF`	Disable Automatic Data Preparation (default).
`PREP_AUTO_ON`	Enable Automatic Data Preparation.

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

GET_MODEL_DETAILS_GLM Function	Returns details about a Generalized Linear Model
GET_MODEL_DETAILS_GLOBAL Function	Returns high-level statistics about a model
GET_MODEL_DETAILS_KM Function	Returns details about a *k*-Means model
GET_MODEL_DETAILS_NB Function	Returns details about a Naive Bayes model
GET_MODEL_DETAILS_NMF Function	Returns details about a Non-Negative Matrix Factorization model
GET_MODEL_DETAILS_OC Function	Returns details about an O-Cluster model
GET_MODEL_DETAILS_SVD Function	Returns details about a Singular Value Decomposition model
GET_MODEL_DETAILS_SVM Function	Returns details about a Support Vector Machine model with a linear kernel
GET_MODEL_DETAILS_XML Function	Returns details about a Decision Tree model

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;
```

	SETTING_NAME	SETTING_VALUE
1	ALGO_NAME	ALGO_GENERALIZED_LINEAR_MODEL
2	GLMS_DIAGNOSTICS_TABLE_NAME	GLMS_BANK_DIAG
3	PREP_AUTO	ON
4	GLMS_FTR_SELECTION	GLMS_FTR_SELECTION_ENABLE
5	GLMS_FTR_GENERATION	GLMS_FTR_GENERATION_ENABLE

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.

	GLOBAL_DETAIL_NAME	GLOBAL_DETAIL_VALUE
1	ADJUSTED_R_SQUARE	0.36454184140088319
2	AIC	-92251.390243386122
3	COEFF_VAR	218.48305088392473
4	CORRECTED_TOTAL_DF	33952
5	CORRECTED_TOT_SS	3520.3485995346509
6	DEPENDENT_MEAN	0.11748593644155156
7	ERROR_DF	33859
8	ERROR_MEAN_SQUARE	0.065888143222410128
9	ERROR_SUM_SQUARES	2230.9066413675846
10	F_VALUE	210.43190191772146
11	GMSEP	0.066071068427140708
12	HOCKING_SP	0.00000194601403574960 5
13	J_P	0.066070556719388496
14	MODEL_DF	93
15	MODEL_F_P_VALUE	0
16	MODEL_MEAN_SQUARE	13.864967292118992
17	MODEL_SUM_SQUARES	1289.4419581670663
18	NUM_PARAMS	94
19	NUM_ROWS	33953
20	ROOT_MEAN_SQ	0.25668685829705057
21	R_SQ	0.36628246371325712
22	SBIC	-91458.713388919496

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;
```

Other regression models did not have such an abundance of statistics when tables were queried for comparison.

1.5 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;
```

	GLOBAL_DETAIL_NAME	GLOBAL_DETAIL_VALUE
1	AIC_INTERCEPT	24576.216188809562
2	AIC_MODEL	14414.889955980363
3	DEPENDENT_MEAN	0.11748593644155156
4	ITERATIONS	8
5	LR_CHI_SQ	10303.326232829198
6	LR_CHI_SQ_P_VALUE	0
7	LR_DF	71
8	NEG2_LL_INTERCEPT	24574.216188809562
9	NEG2_LL_MODEL	14270.889955980363
10	NUM_PARAMS	72
11	NUM_ROWS	33953
12	PCT_CORRECT	0.90828498218125053
13	PCT_INCORRECT	0.091715017818749453
14	PCT_TIED	0
15	PSEUDO_R_SQ_CS	0.26173946895913791
16	PSEUDO_R_SQ_N	0.50815293594557198
17	SC_INTERCEPT	24584.648921303888
18	SC_MODEL	15022.046695571826

This model seems to be preforming very well as its percent for correct classifications is 90-91% leaving ~9% for wrong predictions.

2. 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'));
```

	GLOBAL_DETAIL_NAME	GLOBAL_DETAIL_VALUE
1	ITERATIONS	200
2	LOSS_VALUE	0.12627219603638715

Apply model to test data

```
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;
```

	ROW_ID	ACTUAL	PREDICTION	PROBABILITY	COST
1	2	0	0	1.0	0.0001
2	2	0	1	0.0	0.9999
3	10	0	0	1.0	0.0001

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).
```

	MODEL_ACCU...
1	0.22

	MODEL_ACCURACY
1	0.25

3. 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;
```

Model was created with

```
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'));
```

Reviewed results

	TARGET_ATTRIBUTE_NAME	TARGET_ATTRIBUTE_STR_VALUE	PRIOR_PROBABILITY
1	Y	1	0.11748593644155156
2	Y	0	0.88251406355844841

Apply model to test data

```
begin
    dbms_data_mining.apply(
        'NAIVE2_BAYES_BANK',
        'bank_test_data',
        'row_id',
        'NB_bank_test_predictions'
    );
end;
```

Check results

```
select * from NB_bank_test_predictions fetch first 15 rows only;
```

	ROW_ID	PREDICTION	PROBABILITY	COST
1	44859	0	0.6931132444269636	0.30688675557303635
2	44859	1	0.30688675557303646	0.6931132444269635
3	38555	0	0.5808922856587544	0.4191077143412456
4	38555	1	0.41910771434124566	0.5808922856587544
5	40402	1	0.5310528453818747	0.4689471546181253
6	40402	0	0.4689471546181253	0.5310528453818747
7	38512	1	0.9957578931516667	0.00424210684833326
8	38512	0	0.004242106848333277	0.9957578931516667
9	38806	1	0.9976721663152655	0.002327833684734548
10	38806	0	0.0023278336847345695	0.9976721663152655
11	3854	0	0.9987870756459771	0.0012129243540228707
12	3854	1	0.0012129243540228293	0.9987870756459771
13	42059	0	0.8784194398567269	0.12158056014327312
14	42059	1	0.1215805601432731	0.8784194398567269
15	2169	0	0.9945722184721406	0.005427781527859388

Evaluating results with

```
select
    NB_bank_test_predictions.row_id, bank.y as actual,
NB_bank_test_predictions.prediction as prediction,
    round(NB_bank_test_predictions.probability) as probability,
round(NB_bank_test_predictions.cost, 4) as cost
from NB_bank_test_predictions
    inner join bank on NB_bank_test_predictions.row_id=bank.row_id;
```

```
with
    bank_data
    as
    (
        select count(row_id) as bank_row_count
        from bank
    ),
    model_data
    as
    (
        select count(*) as model_row_count
        from NB_bank_test_predictions
            inner join bank on NB_bank_test_predictions.row_id =
bank.row_id
        where bank.y = NB_bank_test_predictions.prediction and
round(NB_bank_test_predictions.probability) = 1
    )
select round(model_row_count/bank_row_count, 2) as Model_Accuracy
from bank_data, model_data;
```

	MODEL_ACCURACY
1	0.25

This model scored an accuracy of 25% and 22% when taking the predictions column into account.

4. Support Vector Machine

SVM was modelled with the classification and regression method similar to the GLM models, however the regression results were too small to be interpreted using the approach above. This section will cover the classification version of the model

Created settings with

```
begin
    insert into bank_model_settings values(dbms_data_mining.ALGO_NAME,
dbms_data_mining.ALGO_SUPPORT_VECTOR_MACHINES);
    -- DATA PREP
    insert into bank_model_settings values(dbms_data_mining.PREP_AUTO,
dbms_data_mining.PREP_AUTO_ON);
end;
```

also tried

```
 insert into bank_model_settings
values(dbms_data_mining.SVMS_KERNEL_FUNCTION,
dbms_data_mining.SVMS_GAUSSIAN);
```

To test results and see the effect of the model

Model was created with

```
begin
    dbms_data_mining.create_model(
        model_name => 'SVM_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;
```

```
/* There isnt much info in the global or the SVM details, just iteration
count which is 30 for classification and 26 for regression versions*/
```

```
select *
from
table(dbms_data_mining.get_model_details_global('SVM_REGRESSION_BANK'))
order by global_detail_name;
```

```
 select *
from table(dbms_data_mining.GET_MODEL_DETAILS_SVM ('SVM_REGRESSION_BANK'));
```

Model was applied with, as per usual

```
begin
    dbms_data_mining.apply(
        'SVM_REGRESSION_BANK',
        'bank_test_data',
        'row_id',
        'SVM_R_bank_test_predictions'
    );
end;
```

The model was then checked using the same method

```
select
    bank.y as actual, SVM_bank_test_predictions.prediction as prediction,
    round(SVM_bank_test_predictions.probability) as probability,
round(SVM_bank_test_predictions.cost, 4) as cost
from SVM_bank_test_predictions
    inner join bank on SVM_bank_test_predictions.row_id=bank.row_id;
```

```
with
    bank_data
    as
    (
        select count(row_id) as bank_row_count
        from bank
    ),
    model_data
    as
    (
        select count(*) as model_row_count
        from SVM_bank_test_predictions
            inner join bank on SVM_bank_test_predictions.row_id =
bank.row_id
        where bank.y = SVM_bank_test_predictions.prediction and
round(SVM_bank_test_predictions.probability) = 1
    )
select round(model_row_count/bank_row_count, 2) as Model_Accuracy
from bank_data, model_data;
```

What's concerning is that the result for the above command is also 22%, data was checked and the test results were also evaluated as can be seen below, the results for each row are different than that of other models, I'm not sure if this is a positive or negative sign that all models have the same accuracy other than maybe because data wasn't curated to cater to each specific model.

Models were dropped and rebuild, settings were checked etc, command used was

```
BEGIN
DBMS_DATA_MINING.DROP_MODEL('SVM_REGRESSION_BANK', true);
end;
```

	ACTUAL	PREDICTION	PROBABILITY	COST
1	0	0	1.0	0.0562
2	0	1	0.0	0.9438
3	0	0	1.0	0.0711
4	0	1	0.0	0.9289
5	0	0	1.0	0.0491
6	0	1	0.0	0.9509
7	1	0	1.0	0.1089
8	1	1	0.0	0.8911
9	0	0	1.0	0.053
10	0	1	0.0	0.947
11	0	0	1.0	0.0506
12	0	1	0.0	0.9494
13	0	0	1.0	0.0904
14	0	1	0.0	0.9096
15	0	0	1.0	0.2099

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]

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