DUBLIN INSTITUTE OF TECHNOLOGY

WORKING WITH DATA ASSIGNMENT 2

Name: Latisha Panwar
Student No. - D17124379
Class - Data Analytics
Course - DT228A
Year - 1 (2017-2018)
Lecturer - Brendan Tierney
Username - lpanwar
Password - d17124379

PART A – ER Diagram

ER Model consists of six attributes namely, Manager, Depot, Vehicles, Stock, Products and Supplier. These entities consists of a number of attributes as per the guideline. Entities and their attributes are mentioned below in detail.

Manager:

This entity consists information regarding manager staff employed at Dublin Logistics.

- Staff_No - Unique Primary key
- Name - Name of the manager
- Office_Contact_No - Official contact number of the manager
- Home_Contact_No - Unique constraint is applied to this attribute so that every manager has unique phone number

Depot

This entity gives details of all types of depots at Dublin Logistics.

- Depot_Code - Unique Primary key
- Address - Location of the depot
- Type_of_Depot - Whether depot is a local(LOC) or national(NAT) depot
- Telephone_No - Unique constraint applied so that every depot has unique telephone numbers to contact depot
- Additional_Telephone_No - Unique constraint, only for national(NAT) depots
- Manager_Staff_No - Foreign key for this table

Vehicles

Vehicle entity contains information about all the vehicles responsible for the distribution of products.

- Registration_No - Unique Primary key
- Model - Model of the makel
- Make - Brand of the vehicle
- Operated_From - can be local(LOC) or national(NAT)
- Maintained_At - can be maintained only at national(NAT) depots

Supplier

This entity gives details of the suppliers related to Dublin Logistics.

- Supplier_Code - Unique Primary key
- Address - Location of the supplier
- Telephone_No - Unique constraint applied as no two or more suppliers has the same contact numbers

Products

Details of all the products supplied by Dublin Logistics.

- Product_Code - Unique Primary key
- Product_Name - Name of the product

- Product_Description - Details of the product
- Supplier_Supplier_Code - Foreign Key

Stock

It consists of details of all the products, depots they are supplied from and their quantity.

- Number_of_items - Total number of items supplied from the depot
- Depot_Depot_Code - Unique Primary Foreign key
- Products_Product_Code - Unique Primary Foreign key

Depot shares a **one to one relation** with the table Manager. As a manger is never responsible for more than one depot.

Depot entity shares **one to many relation** with Vehicles entity. This is so because one depot should have at least 10 or more vehicles, however one vehicle is operated or maintained at a unique depot and cannot be operated or maintained at any other depot.

Depot entity also shares a **one to many** relation with Stock entity. As each of the products is distributed by at least one or more depot, and every depot holds stock of one or more depot.

Supplier and Products entities share a **one to many** relationship, because each supplier supplies one or more product.

SQL QUERIES

1. MANAGER RESPONSIBLE FOR EACH DEPOT

```
SELECT STAFF_NO, NAME, DEPOT_CODE, TYPE_OF_DEPOT
FROM DEPOT
INNER JOIN MANAGER
ON DEPOT.MANAGER_STAFF_NO = MANAGER.STAFF_NO
ORDER BY STAFF_NO;
```

RESULTS:

STAFF_NO	NAME	DEPOT_CODE	TYPE_OF_DEPOT
S001	John Kinsky	DC102	LOC
S002	Hannah Baker	DC105	NAT
S003	Roy McMuller	DC101	LOC
S004	Clay Johnson	DC104	LOC
S005	Sia Montgomery	DC103	NAT
S006	Tony Tomson	DC106	LOC

2. NUMBER OF LOCAL OR NATIONAL DEPOTS UNDER DUBLIN LOGISTICS

```
SELECT TYPE_OF_DEPOT, COUNT(TYPE_OF_DEPOT) AS NUMBER_OF_DEPOTS
FROM DEPOT
GROUP BY TYPE_OF_DEPOT;
```

RESULTS:

TYPE_OF_DEPOT	NUMBER_OF_DEPOTS
NAT	2
LOC	4

3. NUMBER OF PRODUCTS SUPPLIED BY NATIONAL DEPOT VS LOCAL DEPOT

SELECT DEPOT.TYPE_OF_DEPOT, SUM(NUMBER_OF_ITEMS) AS TOTAL_ITEMS_DELIVERED
FROM STOCK, DEPOT
WHERE DEPOT.DEPOT_CODE = STOCK.DEPOT_DEPOT_CODE
GROUP BY TYPE_OF_DEPOT;

RESULTS:

TYPE_OF_DEPOT	TOTAL_ITEMS_DELIVERED
NAT	48200
LOC	112800

4. NUMBER OF VEHICLES MAINTAINED AT DIFFERENT NATIONAL DEPOTS

SELECT MAINTAINED_AT, COUNT(REGISTRATION_NO) AS
NUMBER_OF_VEHICLES_MAINTAINED
FROM VEHICLES
GROUP BY MAINTAINED_AT;

RESULTS:

MAINTAINED_AT	NUMBER_OF_VEHICLES_MAINTAINED
DC103	6
DC105	6

5. TYPE AND QUANTITY OF PRODUCTS SUPPLIED BY EACH SUPPLIER

SELECT SUPPLIER_CODE, PRODUCTS.PRODUCT_CODE, PRODUCT_NAME,
PRODUCT_DESCRIPTION, STOCK.NUMBER_OF_ITEMS
FROM SUPPLIER
INNER JOIN PRODUCTS
ON SUPPLIER.SUPPLIER_CODE = PRODUCTS.SUPPLIER_SUPPLIER_CODE
INNER JOIN STOCK
ON PRODUCTS.PRODUCT_CODE = STOCK.PRODUCTS_PRODUCT_CODE
ORDER BY SUPPLIER_CODE;

RESULTS:

SUPPLIER_CODE	PRODUCT_CODE	PRODUCT_NAME	PRODUCT_DESCRIPTION	NUMBER_OF_ITEMS
SC01	P012	3D Shampoos	Hair shampoo for all hair types	28000
SC02	P011	XY Sugars	10kg white sugar bags	18500
SC03	P014	RT Jasmine Rice	Jasmine flavoured 20 kgs white rice bags	15000
SC04	P015	Irish Apples	Varieties of Apples grown in Ireland	29700
SC05	P013	T detergents	10kg detergent bags for clothes	31500
SC06	P016	Irish Veggies	Handpicked varieties of tomatoes, potatoes, beetroots, carrots, raddish	38300

6. DETAILS OF PRODUCTS SUPPLIED FROM EACH DEPOT

```
SELECT DEPOT_CODE, TYPE_OF_DEPOT, PRODUCTS.PRODUCT_CODE, PRODUCT_NAME,
PRODUCT_DESCRIPTION, STOCK.NUMBER_OF_ITEMS
FROM DEPOT
INNER JOIN STOCK
ON DEPOT.DEPOT_CODE = STOCK.DEPOT_DEPOT_CODE
INNER JOIN PRODUCTS
ON PRODUCTS.PRODUCT_CODE = STOCK.PRODUCTS_PRODUCT_CODE
ORDER BY DEPOT_CODE;
```

RESULS:

DEPOT_CODE	TYPE_OF_DEPOT	PRODUCT_CODE	PRODUCT_NAME	PRODUCT_DESCRIPTION	NUMBER_OF_ITEMS
DC101	LOC	P014	RT Jasmine Rice	Jasmine flavoured 20 kgs white rice bags	15000
DC102	LOC	P013	T detergents	10kg detergent bags for clothes	31500
DC103	NAT	P015	Irish Apples	Varieties of Apples grown in Ireland	29700
DC104	LOC	P016	Irish Veggies	Handpicked varieties of tomatoes, potatoes, beetroots, carrots, raddish	38300
DC105	NAT	P011	XY Sugars	10kg white sugar bags	18500
DC106	LOC	P012	3D Shampoos	Hair shampoo for all hair types	28000

7. STOCK DETAILS SUPPLIED BY DUBLIN LOGISTICS

```
SELECT SUM(NUMBER_OF_ITEMS) AS TOTAL_QUANTITY_SUPPLIED,
ROUND(AVG(NUMBER_OF_ITEMS), 3) AS AVERAGE_ITEMS_SUPPLIED,
MAX(NUMBER_OF_ITEMS) AS MAXIMUM_QUANTITY_SUPPLIED,
    MIN(NUMBER_OF_ITEMS) AS MINIMUM_QUANTITY_SUPPLIED
    FROM STOCK;
```

RESULTS:

TOTAL_QUANTITY_SUPPLIED	AVERAGE_ITEMS_SUPPLIED	MAXIMUM_QUANTITY_SUPPLIED	MINIMUM_QUANTITY_SUPPLIED
161000	26833.333	38300	15000

Part B – SQL Statistical and Analytical Functions

Dublin Bike data set contains real-time information regarding status of bikes and bike stands at various bike stations all over Dublin. Most important attributes used for the 8 statistical and analytical functions are the following:

a) BANKING- It is the only categorical variable in the data set. It provides information about the baking facility at the bike station. If the value is 'TRUE', the station provides banking facility and if the value is 'FALSE' then there is no banking facility at the station. This information can be useful for the biker to avail the facility in case of need.
b) BIKE_STANDS -This interval variable tells about total number of bike stands at a bike station.
c) AVAILABLE_BIKE_STANDS-It is an interval variable which gives details about the real time status of bike stands available to park bikes.
d) AVAILABLE_BIKES –This attribute is an interval variable which tells the real time status of bikes available at a bike station.

From a biker's point of view, above mentioned attributes holds more importance than other variables in the data set. Hence, following statistical functions have been applied on two or more of the above attributes:

1. STATS_MODE
   This function returns the most occurring value from a set of values. In the following queries this function is applied to 'BIKE_STANDS' to find the frequency of most number of bike stands at a station but based on banking facility.
   SQL QUERY:

```
SELECT BANKING, STATS_MODE(BIKE_STANDS)
    FROM DUBLIN_BIKES
    GROUP BY BANKING
    ORDER BY stats_mode(BIKE_STANDS);
```

   RESULTS:

	BANKING	STATS_MODE(BIKE_STANDS)
1	TRUE	40
2	FALSE	40

   As can be seen from the output, most number of bike stands is 40 for the station if it has banking facility or not.

2. COVARIANCE (COVAR_POP & COVAR_SAMP)
   COVAR_POP AND COVAR_SAMP functions returns population covariance and sample covariance respectively, that is strength of correlation between two or more sets of random variable (WolframMathWorld)

   SQL QUERY:

```
SELECT
    COVAR_POP(AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES) AS CUM_COVP,
    ROUND(COVAR_SAMP(AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES), 5) AS CUM_COVS
 FROM DUBLIN_BIKES
 ORDER BY AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES;
```

   RESULTS:

	CUM_COVP	CUM_COVS
1	-123.099	-124.34242

   Negative covariance tells us that as available bikes tends to decrease as available bike stands increases. This makes sense as the more number the of bike stands available for parking at a bike station the more number of bikes are being used by the people from that station and hence less bikes will be available at that station. Large values means that there is stronger covariance between the two variables.

3. MANN WHITNEY TEST (STATS_MW_TEST)
   This function is a non-parametric test and hence suitable for the attributes considered for this test 'BANKING' and 'BIKE_STANDS' as these data sets are not normally distributed. This test will compare differences between the ordinal variable 'BANKING' and the continuous dependent variable 'BIKE_STANDS'.

   SQL QUERY:

   ```
   SELECT
   ROUND(STATS_MW_TEST(BANKING, BIKE_STANDS, 'STATISTIC'), 5) z_statistic,
   ROUND(STATS_MW_TEST(BANKING, BIKE_STANDS, 'ONE_SIDED_SIG', 'FALSE'), 5)
   one_sided_p_value
   FROM DUBLIN_BIKES;
   ```

   RESULTS:

	Z_STATISTIC	ONE_SIDED_P_VALUE
1	-0.46557	0.67924

   p_value greater than 0.05 indicates no significant relation between total number of bike stands and banking facility at various bike stations.

4. T-TEST (STATS_T_TEST_INDEP)
   This test is a non-parametric test which measure the significance of the difference of means. Here t-TEST of pooled variance is used to compare if means of the two distributions, 'BANKING' and 'BIKE_STANDS' are same or different.

   SQL QUERY:

   ```
   SELECT ROUND(STATS_T_TEST_INDEP(BANKING, BIKE_STANDS, 'STATISTIC', 'FALSE'), 5)
   t_observed_for_false,
       ROUND(STATS_T_TEST_INDEP(BANKING, BIKE_STANDS, 'STATISTIC', 'TRUE'), 5)
   t_observed_for_true,
       ROUND(STATS_T_TEST_INDEP(BANKING, BIKE_STANDS), 5) two_sided_p_value
       FROM DUBLIN_BIKES;
   ```

   RESULTS:

	T_OBSERVED_FOR_FALSE	T_OBSERVED_FOR_TRUE	TWO_SIDED_P_VALUE
1	-0.63889	0.63889	0.52439

   t-Test indicates the same results as Mann Whitney Test, that there is no significant relations between number of bike stands at a station based on banking facility ($p > 0.05$)

5. ONE-WAY ANOVA (STATS_ONE_WAY_ANOVA)

As proved with the covariance test, that bikes available at a bike stations shares an indirect relationship with available bike stands, hence the independent variable considered for this test is 'AVAILABLE_BIKE_STANDS' and dependent variable chosen is 'AVAILABLE_BIKES'.

SQL QUERY:

```
SELECT BANKING,
    ROUND(STATS_ONE_WAY_ANOVA(AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES,
'F_RATIO'), 5) f_ratio,
    ROUND(STATS_ONE_WAY_ANOVA(AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES, 'SIG'),
10) p_value
    FROM DUBLIN_BIKES
    GROUP BY BANKING;
```

RESULTS:

	BANKING	F_RATIO	P_VALUE
1	FALSE	8.76128	0.0000000134
2	TRUE	5.43294	0.0011281488

P_value results close to zero shows that for both type of stations with or without banking facility the difference in available bike stands and available bikes is significant.

6. f-TEST (STATS_F_TEST)
   This test is used to find out whether number of bike stands are significantly different stations with or without banking facility.

SQL QUERY:

```
SELECT ROUND(VARIANCE(DECODE(BANKING, 'TRUE', BIKE_STANDS, null)), 5)
var_BANKING_TRUE,
    ROUND(VARIANCE(DECODE(BANKING, 'FALSE', BIKE_STANDS, null)), 5)
var_BANKING_FALSE,
    ROUND(STATS_F_TEST(BANKING, BIKE_STANDS, 'STATISTIC', 'TRUE'), 5) f_statistic_T,
    ROUND(STATS_F_TEST(BANKING, BIKE_STANDS, 'STATISTIC', 'FALSE'), 5) f_statistic_F,
    ROUND(STATS_F_TEST(BANKING, BIKE_STANDS), 5) two_sided_p_value
    FROM DUBLIN_BIKES;
```

RESULTS:

	VAR_BANKING_TRUE	VAR_BANKING_FALSE	F_STATISTIC_T	F_STATISTIC_F	TWO_SIDED_P_VALUE
1	58.67563	57.20096	1.02578	0.97487	0.90829

From the results it can be interpreted that the difference between bike stands for stations with banking facility as 'TRUE' and for stations with banking facility as 'FALSE' is not significant, as the p_value is not close to zero and f_statistic is close to 1.

7. Kolmogorov-Smirnov TEST (STATS_KS_TEST)

This is a non-parametric test used to check whether the available bike stands or available bikes at the stations has any relation with banking facility available or not at the stations.

SQL QUERY:

```
SELECT ROUND(STATS_KS_TEST(BANKING, AVAILABLE_BIKE_STANDS, 'STATISTIC'), 5)
KS_STATISTIC_AVL_BK_STNDS,
    ROUND(STATS_KS_TEST(BANKING, AVAILABLE_BIKE_STANDS), 5)
P_VALUE_AVL_BK_STNDS,
    ROUND(STATS_KS_TEST(BANKING, AVAILABLE_BIKES, 'STATISTIC'), 5)
KS_STATISTIC_AVL_BKS,
    ROUND(STATS_KS_TEST(BANKING, AVAILABLE_BIKES), 5) P_VALUE_AVL_BKS
    FROM DUBLIN_BIKES;
```

RESULTS:

	KS_STATISTIC_AVL_BK_STNDS	P_VALUE_AVL_BK_STNDS	KS_STATISTIC_AVL_BKS	P_VALUE_AVL_BKS
1	0.11429	0.91085	0.11648	0.89875

The results, p_value not close to zero indicates that the relationship is not significant.

8. Wilcoxon Signed Ranks TEST (STATS_WSR_TEST)
This non-parametric test is performed on two related variables 'BIKE_STANDS' and 'AVAILABLE_BIKE_STANDS' for stations with or without banking facility.

SQL QUERY:

```
SELECT BANKING,
    ROUND(STATS_WSR_TEST(BIKE_STANDS, AVAILABLE_BIKE_STANDS, 'STATISTIC'), 5)
W_STATISTIC
    FROM DUBLIN_BIKES
    GROUP BY BANKING;
```

RESULTS:

	BANKING	W_STATISTIC
1	FALSE	-6.15788
2	TRUE	-4.46112

The results, with values not closer to zero indicate that there are is no significant relationship between total number of bike stands and available bike stands for both stations with or without banking facility available.

Part C – Machine Learning SQL

STEP 1
Confirmed the existence of tables MINING_DATA_BUILD_V AND MINING_DATA_APPLY_V in Oracle schema.

STEP 2
A database view is created after combining both the tables mentioned above.

```
CREATE OR REPLACE VIEW ANALYTICAL_VIEW AS
SELECT * FROM MINING_DATA_BUILD_V
UNION ALL SELECT * FROM MINING_DATA_APPLY_V;
```

Checked number of rows in the ANALYTICAL_VIEW table using following query, as it can be needed in next step to confirm data partition is correct.

```
SELECT COUNT(*) AS NO_OF_ROWS FROM ANALYTICAL_VIEW;
```

NO_OF_ROWS
3000

STEP 3
Two additional database views namely, TRAINING_DATA and TEST_DATA are created. As per the guidelines TRAINING_DATA contains 60% of the data contained in database view ANALYTICAL_VIEW. TEST_DATA contains the remaining 40% of the data.
For sampling of the data ORA_HASH function is used as this function is useful in examining a fragment of data and developing a random sample. Also, this function works well views.

```
CREATE OR REPLACE VIEW TRAINING_DATA AS
WITH ROW_COUNT AS (SELECT COUNT(*) COUNT FROM ANALYTICAL_VIEW )
SELECT AVIEW.* FROM ANALYTICAL_VIEW AVIEW
WHERE ORA_HASH(CUST_ID, (SELECT COUNT FROM ROW_COUNT)-1, 12345) <= (SELECT
COUNT FROM ROW_COUNT)*60/100;
```

```
CREATE OR REPLACE VIEW TEST_DATA AS
WITH ROW_COUNT AS (SELECT COUNT(*) COUNT FROM ANALYTICAL_VIEW )
SELECT AVIEW.* FROM ANALYTICAL_VIEW AVIEW
WHERE ORA_HASH(CUST_ID, (SELECT COUNT FROM ROW_COUNT)-1, 12345) > (SELECT
COUNT FROM ROW_COUNT)*60/100;
```

```
SELECT COUNT(*) FROM TRAINING_DATA;
```

NO_OF_ROWS
1801

```
SELECT COUNT(*) AS NO_OF_ROWS FROM TEST_DATA;
```

NO_OF_ROWS
1199

As can be compared from the number of rows of the ANALYTICAL_VIEW view, the data set is now partitioned into the ratio of 60:40 for TRAINING_DATA and TEST_DATA views.

STEP 4
Machine Learning models are created using Decision Tree and Naïve Bayes algorithms from the data sets generated above.

DECISION TREES

```
CREATE TABLE DTREE_SETTINGS
( SETTING_NAME VARCHAR2(30),
 SETTING_VALUE VARCHAR2(4000));
```

After creating Decision Tree settings, settings are then inserted

```
BEGIN
 INSERT INTO DTREE_SETTINGS (SETTING_NAME, SETTING_VALUE)
 VALUES (DBMS_DATA_MINING.ALGO_NAME,
DBMS_DATA_MINING.ALGO_DECISION_TREE);

 INSERT INTO DTREE_SETTINGS (SETTING_NAME, SETTING_VALUE)
 VALUES (DBMS_DATA_MINING.PREP_AUTO,DBMS_DATA_MINING.PREP_AUTO_ON);
END;
/

SELECT * FROM DTREE_SETTINGS;
```

	SETTING_NAME	SETTING_VALUE
1	ALGO_NAME	ALGO_DECISION_TREE
2	PREP_AUTO	ON

```
--- FOR AFFINITY CARD
BEGIN
   DBMS_DATA_MINING.CREATE_MODEL(
      model_name          => 'DECISION_TREE_AFFINITY_2',
      mining_function     => dbms_data_mining.classification,
      data_table_name     => 'TRAINING_DATA',
      case_id_column_name => 'cust_id',
      target_column_name  => 'affinity_card',
      settings_table_name => 'DTREE_SETTINGS');
END;
/

describe user_mining_model_settings;
```

```
SELECT model_name,
       mining_function,
       algorithm,
       ROUND(build_duration,5) AS BUILD_DURATION,
       model_size
FROM user_MINING_MODELS;
```

```
SELECT setting_name,
         setting_value,
         setting_type
FROM user_mining_model_settings
WHERE model_name in 'DECISION_TREE_AFFINITY_2';
```

	SETTING_NAME	SETTING_VALUE	SETTING_TYPE
1	PREP_AUTO	ON	INPUT
2	TREE_TERM_MINPCT_NODE	.05	DEFAULT
3	TREE_TERM_MINREC_SPLIT	20	DEFAULT
4	TREE_IMPURITY_METRIC	TREE_IMPURITY_GINI	DEFAULT
5	TREE_TERM_MINPCT_SPLIT	.1	DEFAULT
6	TREE_TERM_MAX_DEPTH	7	DEFAULT
7	TREE_TERM_MINREC_NODE	10	DEFAULT
8	ALGO_NAME	ALGO_DECISION_TREE	INPUT

```
SELECT attribute_name,
       attribute_type,
       usage_type,
       target
FROM  all_mining_model_attributes
WHERE model_name = 'DECISION_TREE_AFFINITY_2';
```

	ATTRIBUTE_NAME	ATTRIBUTE_TYPE	USAGE_TYPE	TARGET
1	AGE	NUMERICAL	ACTIVE	NO
2	CUST_MARITAL_STATUS	CATEGORICAL	ACTIVE	NO
3	EDUCATION	CATEGORICAL	ACTIVE	NO
4	HOUSEHOLD_SIZE	CATEGORICAL	ACTIVE	NO
5	OCCUPATION	CATEGORICAL	ACTIVE	NO
6	YRS_RESIDENCE	NUMERICAL	ACTIVE	NO
7	AFFINITY_CARD	CATEGORICAL	ACTIVE	YES

NAÏVE BAYES ALGORITHM

Table for NAÏVE BAYES settings is created and settings are inserted.

```
CREATE TABLE NBAYES_SETTINGS
( SETTING_NAME VARCHAR2(30),
 SETTING_VALUE VARCHAR2(4000));

BEGIN
 INSERT INTO NBAYES_SETTINGS (SETTING_NAME, SETTING_VALUE)
 VALUES (DBMS_DATA_MINING.ALGO_NAME, DBMS_DATA_MINING.ALGO_NAIVE_BAYES);

 INSERT INTO NBAYES_SETTINGS (SETTING_NAME, SETTING_VALUE)
 VALUES (DBMS_DATA_MINING.PREP_AUTO,DBMS_DATA_MINING.PREP_AUTO_ON);
END;
/

SELECT * FROM NBAYES_SETTINGS;
```

	SETTING_NAME	SETTING_VALUE
1	ALGO_NAME	ALGO_NAIVE_BAYES
2	PREP_AUTO	ON

```
--- FOR AFFINITY CARD
BEGIN
   DBMS_DATA_MINING.CREATE_MODEL(
      model_name          => 'NAIVE_BAYES_AFFINITY',
      mining_function     => dbms_data_mining.classification,
      data_table_name     => 'TRAINING_DATA',
      case_id_column_name => 'cust_id',
      target_column_name  => 'affinity_card',
      settings_table_name => 'NBAYES_SETTINGS');
END;
/

describe user_mining_model_settings;

SELECT model_name,
      mining_function,
      algorithm,
      ROUND(build_duration,5) AS BUILD_DURATION,
      model_size
FROM user_MINING_MODELS;
```

	MODEL_NAME	MINING_FUNCTION	ALGORITHM	BUILD_DURATION	MODEL_SIZE
1	DECISION_TREE_AFFINITY_2	CLASSIFICATION	DECISION_TREE	1	0.0664
2	NAIVE_BAYES_AFFINITY	CLASSIFICATION	NAIVE_BAYES	1	0.0602

```
SELECT setting_name,
        setting_value,
        setting_type
```

FROM user_mining_model_settings
WHERE model_name in 'NAIVE_BAYES_AFFINITY';

	SETTING_NAME	SETTING_VALUE	SETTING_TYPE
1	ALGO_NAME	ALGO_NAIVE_BAYES	INPUT
2	PREP_AUTO	ON	INPUT
3	NABS_SINGLETON_THRESHOLD	0	DEFAULT
4	NABS_PAIRWISE_THRESHOLD	0	DEFAULT

SELECT attribute_name,
    attribute_type,
    usage_type,
    target
from all_mining_model_attributes
where model_name = 'NAIVE_BAYES_AFFINITY';

	ATTRIBUTE_NAME	ATTRIBUTE_TYPE	USAGE_TYPE	TARGET
1	AGE	NUMERICAL	ACTIVE	NO
2	HOME_THEATER_PACKAGE	NUMERICAL	ACTIVE	NO
3	CUST_GENDER	CATEGORICAL	ACTIVE	NO
4	CUST_MARITAL_STATUS	CATEGORICAL	ACTIVE	NO
5	BOOKKEEPING_APPLICATION	NUMERICAL	ACTIVE	NO
6	EDUCATION	CATEGORICAL	ACTIVE	NO
7	HOUSEHOLD_SIZE	CATEGORICAL	ACTIVE	NO
8	OCCUPATION	CATEGORICAL	ACTIVE	NO
9	Y_BOX_GAMES	NUMERICAL	ACTIVE	NO
10	YRS_RESIDENCE	NUMERICAL	ACTIVE	NO
11	AFFINITY_CARD	CATEGORICAL	ACTIVE	YES

STEP 5
In this step, models created above are evaluated using TEST_DATA view.

Applying Decision Tree model to TEST_DATA view:

CREATE OR REPLACE VIEW DTREE_TEST_RESULTS
AS
SELECT cust_id,
    ROUND(prediction(DECISION_TREE_AFFINITY_2 USING *), 5) predicted_value,
    ROUND(prediction_probability(DECISION_TREE_AFFINITY_2 USING *), 5) probability
FROM   TEST_DATA;

Applying Naïve Bayes model to TEST_DATA view:

CREATE OR REPLACE VIEW NBAYES_TEST_RESULTS
AS

```
SELECT cust_id,
     ROUND(prediction(NAIVE_BAYES_AFFINITY USING *), 5)  predicted_value,
     ROUND(prediction_probability(NAIVE_BAYES_AFFINITY USING *), 5) probability
FROM   TEST_DATA;

--- DTREE CONFUSION MATRIX
DECLARE
v_accuracy NUMBER;
BEGIN
DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
accuracy => v_accuracy,
apply_result_table_name => 'dtree_test_results',
target_table_name => 'test_data',
case_id_column_name => 'cust_id',
target_column_name => 'affinity_card',
confusion_matrix_table_name => 'dtree_confusion_matrix',
score_column_name => 'PREDICTED_VALUE',
score_criterion_column_name => 'PROBABILITY',
cost_matrix_table_name => null,
apply_result_schema_name => null,
target_schema_name => null,
cost_matrix_schema_name => null,
score_criterion_type => 'PROBABILITY');
DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;

select * from dtree_confusion_matrix;
```

The confusion matrix generated for decision tree model is as follows:

	ACTUAL_TARGET_VALUE	PREDICTED_TARGET_VALUE	VALUE
1	1	0	184
2	0	0	828
3	1	1	134
4	0	1	53

```
--- NAIVE BAYES Confusion Matrix
DECLARE
v_accuracy NUMBER;
BEGIN
DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
accuracy => v_accuracy,
apply_result_table_name => 'NBAYES_test_results',
target_table_name => 'test_data',
case_id_column_name => 'cust_id',
target_column_name => 'affinity_card',
confusion_matrix_table_name => 'NBAYES_confusion_matrix',
score_column_name => 'PREDICTED_VALUE',
```

```
score_criterion_column_name => 'PROBABILITY',
cost_matrix_table_name => null,
apply_result_schema_name => null,
target_schema_name => null,
cost_matrix_schema_name => null,
score_criterion_type => 'PROBABILITY');
DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;
```

```
select * from NBAYES_confusion_matrix;
```

The confusion matrix generated for Naïve Bayes model is as follows:

	ACTUAL_TARGET_VALUE	PREDICTED_TARGET_VALUE	VALUE
1	1	0	73
2	0	0	701
3	1	1	245
4	0	1	180

STEP 6 & 7

Now, a new table LABELLED_DATA is created that contains random sample of 20 records created from ANALYTICAL_VIEW, two additional attributes that contains predicted value and predicted probability value.

```
CREATE TABLE LABELLED_DATA AS
SELECT FULL.* FROM
(SELECT AVIEW.*, PROB.DTREE_PREDICTED_VALUE, PROB.DTREE_PROBABILITY,
PROB.NBAYES_PREDICTED_VALUE, PROB.NBAYES_PROBABILITY FROM ANALYTICAL_VIEW
AVIEW,
(SELECT DTREE.CUST_ID, DTREE.PREDICTED_VALUE AS DTREE_PREDICTED_VALUE,
DTREE.PROBABILITY AS DTREE_PROBABILITY, NBAYES.PREDICTED_VALUE AS
NBAYES_PREDICTED_VALUE, NBAYES.PROBABILITY AS NBAYES_PROBABILITY
FROM DTREE_TEST_RESULTS DTREE
JOIN NBAYES_TEST_RESULTS NBAYES
ON DTREE.CUST_ID = NBAYES.CUST_ID) PROB
WHERE AVIEW.CUST_ID = PROB.CUST_ID
ORDER BY dbms_random.value) FULL
WHERE rownum <= 20;
```

```
SELECT * FROM LABELLED_DATA;
```

	CUST_ID	CUST_GENDER	AGE	CUST_MARITAL_STATUS	COUNTRY_NAME	CUST_INCOME_LEVEL	EDUCATION	OCCUPATION	HOUSEHOLD_SIZE	YRS_RESIDENCE	AFFINITY_CARD	BULK_PACK_DISKETTES	FLAT_PANEL_MONITOR
1	101765	M	71	Married	United States of America	I: 170,000 - 189,999	HS-grad	?	3	7	0	1	1
2	100471	M	49	Married	United States of America	I: 170,000 - 189,999	HS-grad	Crafts	3	5	1	1	1
3	100649	F	31	NeverM	United States of America	C: 50,000 - 69,999	< Bach.	Cleric.	1	2	0	0	0
4	100080	M	27	Married	United States of America	G: 130,000 - 149,999	9th	Crafts	3	3	0	0	0
5	100161	M	56	NeverM	United States of America	C: 50,000 - 69,999	Bach.	Transp.	2	10	0	0	0
6	100594	F	26	NeverM	United States of America	J: 190,000 - 249,999	< Bach.	Sales	6-8	1	0	1	1
7	100400	M	59	Divorc.	United States of America	C: 50,000 - 69,999	12th	Transp.	6-8	6	0	0	0
8	102473	M	38	Married	United States of America	B: 30,000 - 49,999	Bach.	Prof.	3	6	1	0	0
9	100931	M	48	Married	United States of America	K: 250,000 - 299,999	< Bach.	Crafts	3	6	1	1	1
10	100484	F	24	NeverM	United States of America	I: 170,000 - 189,999	12th	Other	1	2	0	1	1
11	102085	M	47	Separ.	United States of America	L: 300,000 and above	HS-grad	Crafts	9+	5	0	1	1
12	102460	M	49	NeverM	United States of America	E: 90,000 - 109,999	Bach.	House-s	2	4	0	0	0
13	100043	M	24	NeverM	United States of America	G: 130,000 - 149,999	HS-grad	Other	1	2	0	0	0
14	101399	M	43	Mabsent	United States of America	J: 190,000 - 249,999	Profsc	Prof.	2	4	1	1	1
15	100155	M	26	Married	United States of America	I: 170,000 - 189,999	Assoc-V	Cleric.	3	3	0	1	1
16	102223	M	33	Divorc.	United States of America	I: 170,000 - 189,999	12th	Crafts	2	3	0	1	1
17	101897	M	34	Married	United States of America	J: 190,000 - 249,999	HS-grad	Protec.	3	3	0	1	1
18	102055	F	43	Divorc.	United States of America	K: 250,000 - 299,999	HS-grad	Cleric.	9+	5	0	1	1
19	100999	M	52	Married	United States of America	J: 190,000 - 249,999	7th-8th	Crafts	3	5	0	1	1
20	101549	M	38	Married	United States of America	L: 300,000 and above	< Bach.	Exec.	3	6	1	1	1

Continued from above table..

HOME_THEATER_PACKAGE	BOOKKEEPING_APPLICATION	PRINTER_SUPPLIES	Y_BOX_GAMES	OS_DOC_SET_KANJI	DTREE_PREDICTED_VALUE	DTREE_PROBABILITY	NBAYES_PREDICTED_VALUE	NBAYES_PROBABILITY
1	1	1	0	0	0	0.63684	1	0.74968
1	1	1	0	0	0	0.63684	1	0.90509
0	1	1	0	0	0	0.9901	0	0.99996
0	0	1	1	0	0	0.88591	0	0.99615
1	1	1	0	0	0	0.91898	0	0.88342
0	1	1	1	0	0	0.9901	0	1
1	0	1	0	0	0	0.9901	0	0.99922
1	1	1	0	0	1	0.68621	1	0.99096
1	1	1	0	0	0	0.63684	1	0.90509
0	0	1	1	0	0	0.9901	0	1
1	1	1	0	0	0	0.9901	0	0.98666
1	1	1	0	0	0	0.91898	0	0.88342
0	1	1	1	0	0	0.9901	0	1
1	1	1	0	0	0	0.68889	1	0.77275
0	1	1	1	0	0	0.88591	0	0.97944
0	0	1	0	0	0	0.91898	0	0.99849
0	1	1	0	0	0	0.88591	1	0.68226
1	1	1	0	0	0	0.9901	0	0.99685
1	0	1	0	0	0	0.63684	1	0.63695
1	1	1	0	0	0	0.63684	1	0.96681

Part D – PL/SQL Code

```
SET SERVEROUTPUT ON

CREATE TABLE LABELLED_ALLDATA AS
SELECT FULL.* FROM
(SELECT AVIEW.*, PROB.DTREE_PREDICTED_VALUE, PROB.DTREE_PROBABILITY,
PROB.NBAYES_PREDICTED_VALUE, PROB.NBAYES_PROBABILITY FROM ANALYTICAL_VIEW
AVIEW,
(SELECT DTREE.CUST_ID, DTREE.PREDICTED_VALUE AS DTREE_PREDICTED_VALUE,
DTREE.PROBABILITY AS DTREE_PROBABILITY, NBAYES.PREDICTED_VALUE AS
NBAYES_PREDICTED_VALUE, NBAYES.PROBABILITY AS NBAYES_PROBABILITY
FROM DTREE_TEST_RESULTS DTREE
JOIN NBAYES_TEST_RESULTS NBAYES
ON DTREE.CUST_ID = NBAYES.CUST_ID) PROB
WHERE AVIEW.CUST_ID = PROB.CUST_ID
ORDER BY DBMS_RANDOM.VALUE) FULL;

SELECT * FROM LABELLED_ALLDATA;

declare
```

```
True_Negative number;
True_Positive number;
False_Positive number;
False_Negative number;
BEGIN
   select count(*) into True_Negative from LABELLED_NEWDATA WHERE affinity_card=0 and
dtree_predicted_value=0;
   select count(*) into True_Positive from LABELLED_NEWDATA WHERE affinity_card=1 and
dtree_predicted_value=1;
   select count(*) into False_Positive from LABELLED_NEWDATA WHERE affinity_card=0 and
dtree_predicted_value=1;
   select count(*) into False_Negative from LABELLED_NEWDATA WHERE affinity_card=1 and
dtree_predicted_value=0;

DBMS_OUTPUT.PUT_LINE('Number of false Positives ' ||  False_Positive);
DBMS_OUTPUT.PUT_LINE('Number of true Negatives ' ||  True_Negative);
DBMS_OUTPUT.PUT_LINE('Number of true Positives ' ||  True_Positive);
DBMS_OUTPUT.PUT_LINE('Number of false Negatives ' ||  True_Positive);

End;
/
```

```
Number of false Positives 53
Number of true Negatives 828
Number of true Positives 134
Number of false Negatives 134
```

```
declare
True_Negative number;
True_Positive number;
False_Positive number;
False_Negative number;
BEGIN
select count(*) into True_Negative from LABELLED_NEWDATA WHERE affinity_card=0 and
nbayes_predicted_value=0;
   select count(*) into True_Positive from LABELLED_NEWDATA WHERE affinity_card=1 and
nbayes_predicted_value=1;
   select count(*) into False_Positive from LABELLED_NEWDATA WHERE affinity_card=0 and
nbayes_predicted_value=1;
   select count(*) into False_Negative from LABELLED_NEWDATA WHERE affinity_card=1 and
nbayes_predicted_value=0;

DBMS_OUTPUT.PUT_LINE('Number of false Positives ' ||  False_Positive);
DBMS_OUTPUT.PUT_LINE('Number of true Negatives ' ||  True_Negative);
DBMS_OUTPUT.PUT_LINE('Number of true Positives ' ||  True_Positive);
DBMS_OUTPUT.PUT_LINE('Number of false Negatives ' ||  True_Positive);
```

End;
/

```
Number of false Positives 180
Number of true Negatives 701
Number of true Positives 245
Number of false Negatives 245
```

As can be seen from the results, confusion matrices created by the PL/SQL code is same as the one generated in the PART C of the report.