PART A

DATA MODELLING

Data model is a structure which represents the data and the relationships between each of the tables stored in a database. Data modelling is a technique where a set of tools and techniques are used to prepare a data model and it is an essential skill for any data analyst involved in analysing an organisation's data. Later this diagram is implemented to create the ***Schema*** of database model comprising of tables, views, columns and consytraints.

In this project, Oracle SQL Developer Data modeller and Oracle SQL Developer are used to design the data model and implement the same in developer to create relevant tables and data. Here, an example scenario of **Dublin Logistics distribution company** has been considered for understanding of the data modelling.

- **DATA MODEL**

There are three main phases of data modelling included in this project:

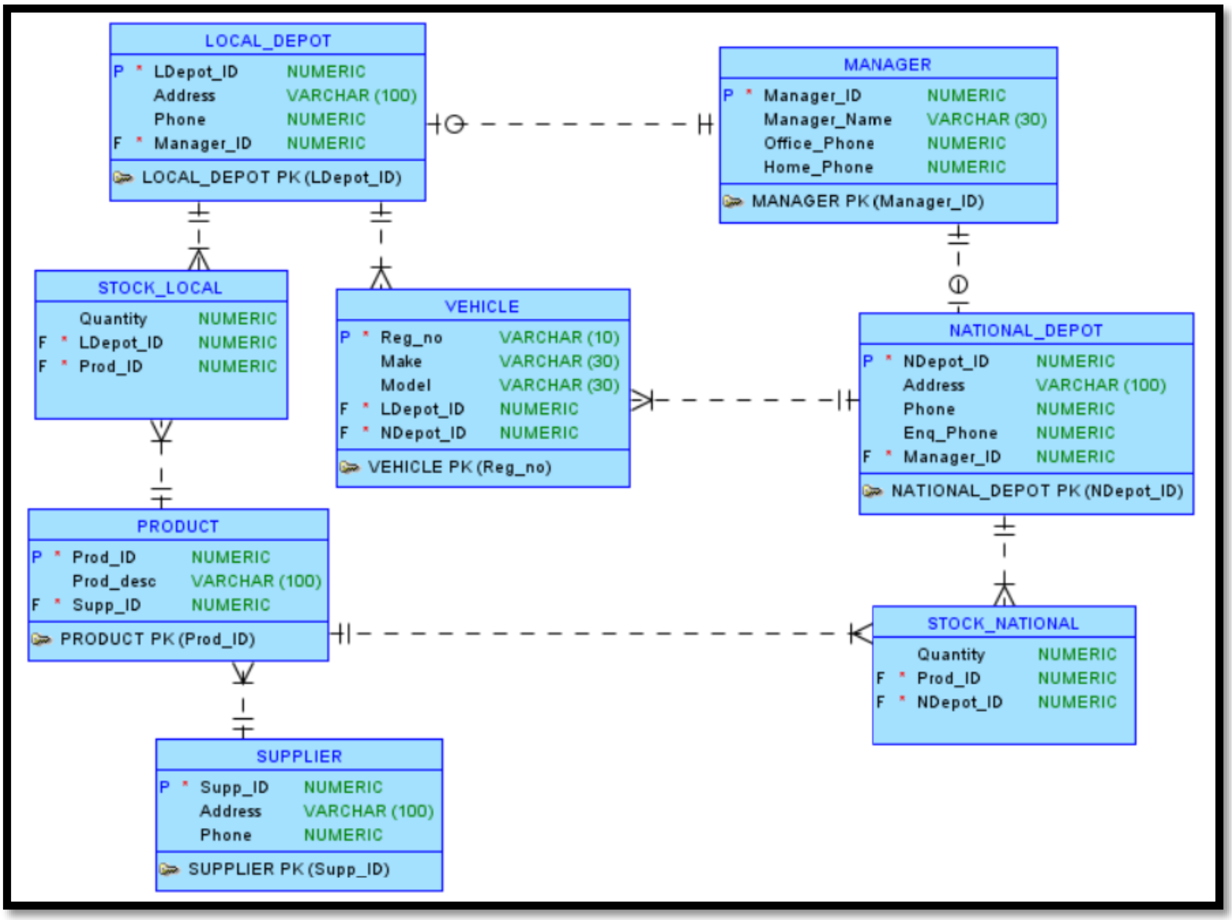
1. Designing a rough sketch of entities and attributes linked to the scenario as explained.

MANAGER	LOCAL_DEPOT	NATIONAL_DEPOT	VEHICLE
Manager_ID	LDepot_ID	NDepot_ID	Reg_No
Manager_Name	Address	Address	Make
Office_Phone	Phone	Enq_Phone	Model
Home_Phone	Manager_ID	Phone	Operated by
	Prod_ID	Manager_ID	Maintained by
		Prod_ID	

STOCK_LOCAL	STOCK_NATIONAL	SUPPLIER	PRODUCT
Quantity(>=0)	Quantity(>=0)	Supp_ID	Prod_ID
LDepot_ID	NDepot_ID	Address	Prod_desc
Prod_ID	Prod_ID	Phone	Supp_ID

- In this phase we roughly design our data model with selected number of entities and its relevant attributes.
- Here we have got 8 *entities* namely ***Manager***, ***Local_depot***, ***National_depot***, ***Vehicle***, ***Stock_local***, ***Stock_national***, ***Supplier***, ***Product***.

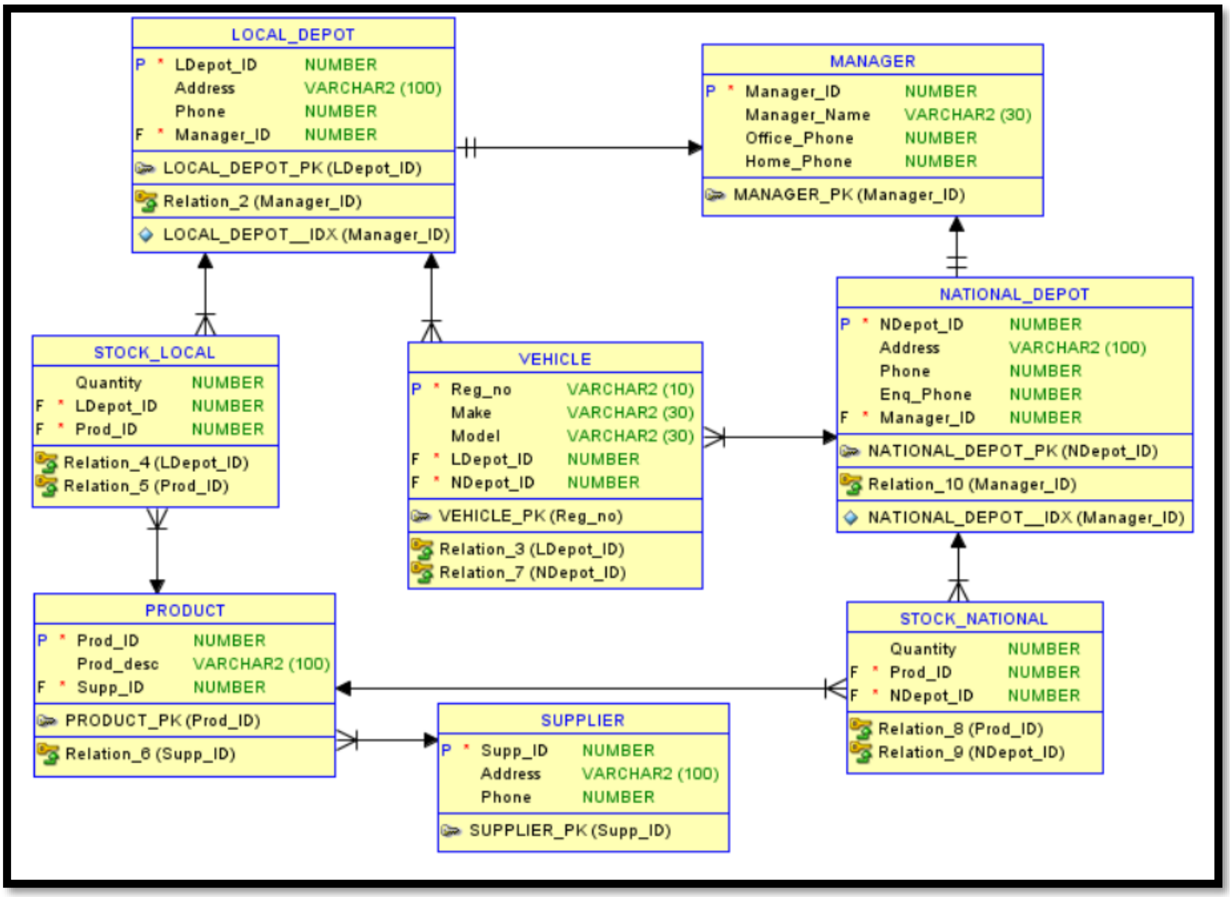
2. Creating a physical data model using Oracle SQL Developer Data modeller, based on the rough sketch made above.



- A physical data model basically creates ***Entity-Relationship(ER)*** diagram which represents key ***relationships*** between the concepts in a data. Each of these concepts are called ***entities*** with relevant ***attributes*** mapped to it.It explains the data as much as possible, each attributes consists of primary keys and foreign keys. ***Primary keys*** identify each of the entities specified. It is always not mandatory fo an attribute to have primary key but it should necessarily have an unique identifier. ***Foreign keys*** identify the relationship between different entities which are specified.
  Relationship between each of the entities is represented by **cardinality** which connects between the entities to explain how each of them link together.
  It is also important to see that our data model is **normalised** as much as possible without including large number of entities. End of the day we want to create a database with minimum number of tables expalining maximum relationships between them.

3. Creating a relational data model using Oracle SQL Developer Data modeller by engineering the physical model to relational model.

- In this data model entities are considered as tables and the attributes are the columns of that table. Naming conventions are taken care to be compatible with the database.



- The above diagram contains **8 entities** as explained below.

1. **LOCAL_DEPOT**: The depot from where vehicles get operated by, with *LDepot_ID as* primary key along with other attributes like address and phone number of depot. *Manager_ID* is the foreign key.
2. **NATIONAL_DEPOT**: The depot from where vehicles get maintained by, with *NDepot_ID* as primary key along with other attributes like address, enquiry phone number and depot phone number and *Manager_ID* as foreign key.
3. **VEHICLE**: Vehicles operated in company with primary key *Reg_no* and other attributes like make and model of vehicle with *LDepot_ID* and *NDepot_ID* as foreign keys.
4. **MANAGER**: Managers employed by Dublin Logistics Company with primary key *Manager_ID* and other attributes like name,office phone and home phone.
5. **STOCK_LOCAL**: This entity represents stock maintained at local depot which contains *Quantity* of products. This entity does not contain primary key but has 2 foreign keys *LDepot_ID* and *Prod_ID*.
6. **STOCK_NATIONAL**: This entity represents stock maintained at national depot which contains *Quantity* of products. This entity does not contain primary key but has 2 foreign keys *NDepot_ID* and *Prod_ID*.

7. **SUPPLIER**: Represents supplier who supplies products to depots, with primary key *Supp_ID* and other attributes like address and phone number of supplier.
8. **PRODUCT**: The entity represents the products with primary key *Prod_ID* and another attribute product description. *Supp_ID* serves as foreign key here.

- Below are the relationships between the entities as explained:

➢ Each depot has a manager and some managers also have other managerial responsibilities instead of managing depots.

  - *relationship represented by 1:1 from LOCAL_DEPOT to MANAGER with source optional and by 1:1 from NATIONAL_DEPOT to MANAGER with source optional.*

➢ Each depot operates ten or more vehicles.

  - *relationship represented by 1:N from LOCAL_DEPOT to VEHICLE and 1:N from NATIONAL_DEPOT to VEHICLE.*

➢ Stocks at local and national depots contain number of products which may be zero sometimes.

  - *relationship represented by 1:N from STOCK_LOCAL to PRODUCT and 1:N from STOCK_NATIONAL to PRODUCT. Additional constraint is added to tables of stock, that Quantity>=0.*

➢ Each product is supplied by a single supplier.

  - *relationship represented by 1:N from SUPPLIER to PRODUCT.*

➢ Every depot holds stock of one or more products.

  - *relationship represented by 1:N from LOCAL_DEPOT to STOCK_LOCAL AND NATIONAL_DEPOT to STOCK_NATIONAL.*

- **TABLE CREATION**

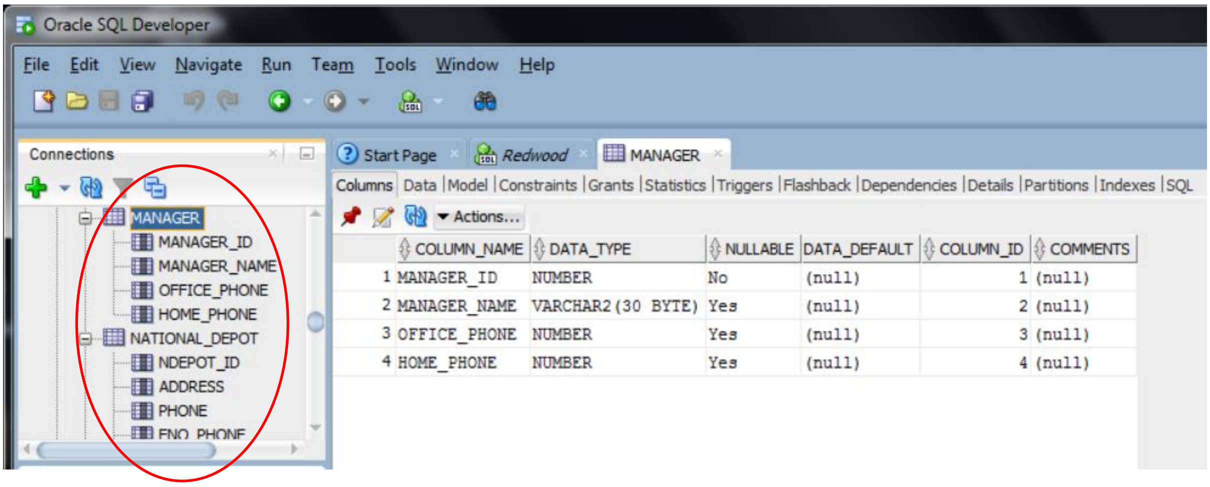
From the relational model obtained above, DDL script is generated which can be used in Oracle SQL Data Developer to create the relevant tables for database. Below is the DDL script obtained by the relational model designed in previous section.

```sql
CREATE TABLE local_depot (
    ldepot_id    NUMBER NOT NULL,
    address      VARCHAR2(100),
    phone        NUMBER,
    manager_id   NUMBER NOT NULL
);

CREATE UNIQUE INDEX local_depot__idx ON
    local_depot ( manager_id ASC );

ALTER TABLE local_depot ADD CONSTRAINT local_depot_pk PRIMARY
KEY ( ldepot_id );

CREATE TABLE manager (
    manager_id     NUMBER NOT NULL,
    manager_name   VARCHAR2(30),
    office_phone   NUMBER,
    home_phone     NUMBER
);

ALTER TABLE manager ADD CONSTRAINT manager_pk PRIMARY KEY (
manager_id );

CREATE TABLE national_depot (
    ndepot_id    NUMBER NOT NULL,
    address      VARCHAR2(100),
    phone        NUMBER,
    enq_phone    NUMBER,
    manager_id   NUMBER NOT NULL
);

CREATE UNIQUE INDEX national_depot__idx ON
    national_depot ( manager_id ASC );

ALTER TABLE national_depot ADD CONSTRAINT national_depot_pk
PRIMARY KEY ( ndepot_id );

CREATE TABLE product (
    prod_id     NUMBER NOT NULL,
    prod_desc   VARCHAR2(100),
    supp_id     NUMBER NOT NULL
);

ALTER TABLE product ADD CONSTRAINT product_pk PRIMARY KEY (
prod_id );
```

```sql
CREATE TABLE stock_local (
    quantity    NUMBER,
    ldepot_id   NUMBER NOT NULL,
    prod_id     NUMBER NOT NULL
);

CREATE TABLE stock_national (
    quantity    NUMBER,
    prod_id     NUMBER NOT NULL,
    ndepot_id   NUMBER NOT NULL
);

CREATE TABLE supplier (
    supp_id   NUMBER NOT NULL,
    address   VARCHAR2(100),
    phone     NUMBER
);

ALTER TABLE supplier ADD CONSTRAINT supplier_pk PRIMARY KEY (
supp_id );

CREATE TABLE vehicle (
    reg_no      NUMBER NOT NULL,
    make        VARCHAR2(30),
    model       VARCHAR2(30),
    ldepot_id   NUMBER NOT NULL,
    ndepot_id   NUMBER NOT NULL
);

ALTER TABLE vehicle ADD CONSTRAINT vehicle_pk PRIMARY KEY (
reg_no );

ALTER TABLE national_depot
    ADD CONSTRAINT relation_10 FOREIGN KEY ( manager_id )
        REFERENCES manager ( manager_id );

ALTER TABLE local_depot
    ADD CONSTRAINT relation_2 FOREIGN KEY ( manager_id )
        REFERENCES manager ( manager_id );

ALTER TABLE vehicle
    ADD CONSTRAINT relation_3 FOREIGN KEY ( ldepot_id )
  REFERENCES local_depot ( ldepot_id );
```

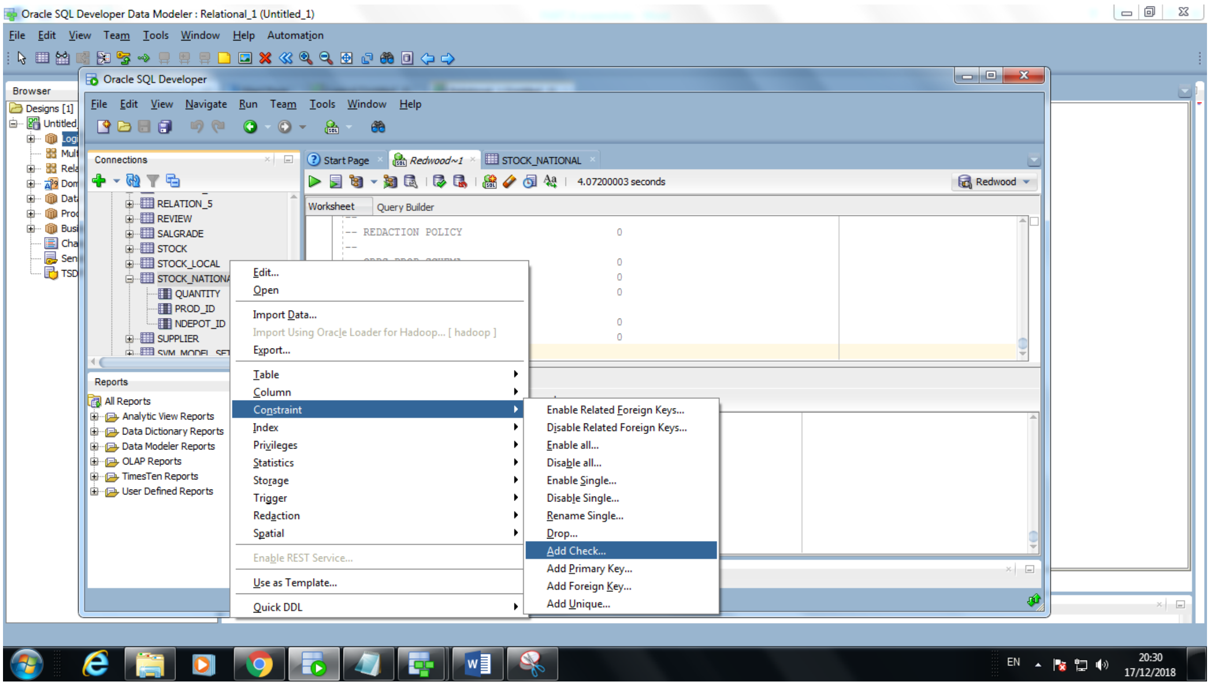
```
ALTER TABLE stock_local
    ADD CONSTRAINT relation_4 FOREIGN KEY ( ldepot_id )
        REFERENCES local_depot ( ldepot_id );

ALTER TABLE stock_local
    ADD CONSTRAINT relation_5 FOREIGN KEY ( prod_id )
        REFERENCES product ( prod_id );

ALTER TABLE product
    ADD CONSTRAINT relation_6 FOREIGN KEY ( supp_id )
        REFERENCES supplier ( supp_id );

ALTER TABLE vehicle
    ADD CONSTRAINT relation_7 FOREIGN KEY ( ndepot_id )
        REFERENCES national_depot ( ndepot_id );

ALTER TABLE stock_national
    ADD CONSTRAINT relation_8 FOREIGN KEY ( prod_id )
        REFERENCES product ( prod_id );

ALTER TABLE stock_national
    ADD CONSTRAINT relation_9 FOREIGN KEY ( ndepot_id )
        REFERENCES national_depot ( ndepot_id );
```

As a result 8 tables are created in the Oracle schema, for example as shown below:

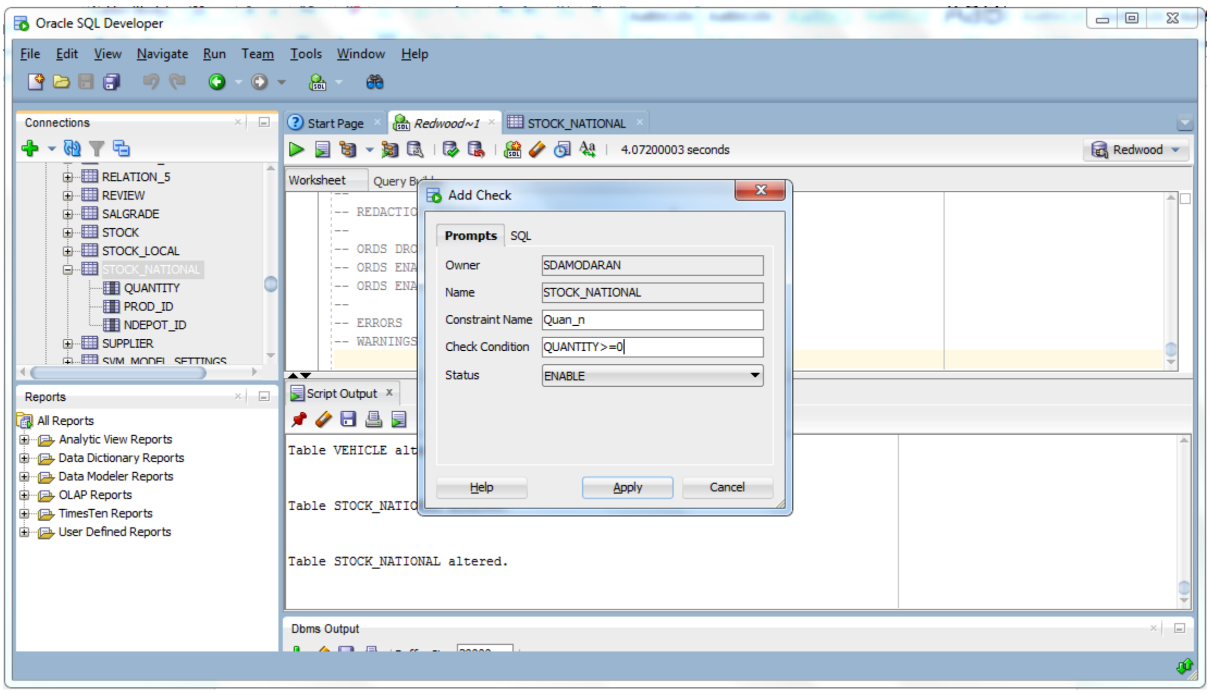


After creating the tables we need to add a constraint for tables STOCK_LOCAL and STOCK_NATIONAL, since the Quantity of stocks recorded can be zero sometimes. Below are the steps followed for adding constraint QUANTITY>=0.

By right clicking on the corresponding table we get **Constraint** option. **Add check** under Constraint has been selected.



The condition QUANTITY>=0 has been added as shown in the window below:



A confirmation window will pop up saying the constraint has been added.

- **DATA INSERTION INTO TABLES THAT ARE CREATED**

Once the tables are created, now we will test the tables by adding some records to each of the tables. Below is the code used for inserting data into tables.

```
INSERT INTO MANAGER
(MANAGER_ID,MANAGER_NAME,OFFICE_PHONE,HOME_PHONE)
VALUES (1,'sowmya',7777777777,88888888);
INSERT INTO MANAGER
(MANAGER_ID,MANAGER_NAME,OFFICE_PHONE,HOME_PHONE)
VALUES (2,'Devraj',6666666666,44444444);
INSERT INTO MANAGER
(MANAGER_ID,MANAGER_NAME,OFFICE_PHONE,HOME_PHONE)
VALUES (3,'Divya',9999999999,34565435);
INSERT INTO MANAGER
(MANAGER_ID,MANAGER_NAME,OFFICE_PHONE,HOME_PHONE)
VALUES (4,'Patil',7865432876,87564678);
INSERT INTO MANAGER
(MANAGER_ID,MANAGER_NAME,OFFICE_PHONE,HOME_PHONE)
VALUES (5,'Rakesh',6754567879,44565678);
INSERT INTO MANAGER
(MANAGER_ID,MANAGER_NAME,OFFICE_PHONE,HOME_PHONE)
VALUES (6,'Charan',6675456787,88766467);
INSERT INTO MANAGER
(MANAGER_ID,MANAGER_NAME,OFFICE_PHONE,HOME_PHONE)
VALUES (7,'Jeevan',6675677878,55896258);
INSERT INTO MANAGER
(MANAGER_ID,MANAGER_NAME,OFFICE_PHONE,HOME_PHONE)
VALUES (8,'Charitha',0012533012,11200369);
INSERT INTO MANAGER
(MANAGER_ID,MANAGER_NAME,OFFICE_PHONE,HOME_PHONE)
VALUES (9,'Rafia',4452566958,10238950);
INSERT INTO MANAGER
(MANAGER_ID,MANAGER_NAME,OFFICE_PHONE,HOME_PHONE)
VALUES (10,'Sudeepti',1236547890,22589632);

INSERT INTO SUPPLIER (SUPP_ID,ADDRESS,PHONE)
VALUES (1,'BANGALORE',7789876756);
INSERT INTO SUPPLIER (SUPP_ID,ADDRESS,PHONE)
VALUES (2,'HUBLI',5586599685);
INSERT INTO SUPPLIER (SUPP_ID,ADDRESS,PHONE)
VALUES (3,'DHARWAD',5521478963);
INSERT INTO SUPPLIER (SUPP_ID,ADDRESS,PHONE)
VALUES (4,'BIDAR',4454788854);
INSERT INTO SUPPLIER (SUPP_ID,ADDRESS,PHONE)
VALUES (5,'MANGALURU',5589636986);
```

```
INSERT INTO PRODUCT (PROD_ID,PROD_DESC,SUPP_ID)
VALUES (1,'RICE',1);
INSERT INTO PRODUCT (PROD_ID,PROD_DESC,SUPP_ID)
VALUES (2,'SUGAR',2);
INSERT INTO PRODUCT (PROD_ID,PROD_DESC,SUPP_ID)
VALUES (3,'PULSES',3);
INSERT INTO PRODUCT (PROD_ID,PROD_DESC,SUPP_ID)
VALUES (4,'WHEAT',4);
INSERT INTO PRODUCT (PROD_ID,PROD_DESC,SUPP_ID)
VALUES (5,'RAGI',5);

INSERT INTO LOCAL_DEPOT (LDEPOT_ID,ADDRESS,PHONE,MANAGER_ID)
VALUES (1,'BRAY A94',1234567891,1);
INSERT INTO LOCAL_DEPOT (LDEPOT_ID,ADDRESS,PHONE,MANAGER_ID)
VALUES (2,'LEESON STREET',7786657786,2);
INSERT INTO LOCAL_DEPOT (LDEPOT_ID,ADDRESS,PHONE,MANAGER_ID)
VALUES (3,'STILLORGAN PARK',4478596582,6);
INSERT INTO LOCAL_DEPOT (LDEPOT_ID,ADDRESS,PHONE,MANAGER_ID)
VALUES (4,'BRAY TOWNHALL',5587956852,7);
INSERT INTO LOCAL_DEPOT (LDEPOT_ID,ADDRESS,PHONE,MANAGER_ID)
VALUES (5,'SIMMONS COURT',5587596585,8);

INSERT INTO NATIONAL_DEPOT
(NDEPOT_ID,ADDRESS,ENQ_PHONE,PHONE,MANAGER_ID)
VALUES (1,'STILLORGAN',22589634,1251250369,4);
INSERT INTO NATIONAL_DEPOT
(NDEPOT_ID,ADDRESS,ENQ_PHONE,PHONE,MANAGER_ID)
VALUES (2,'HEUSTON',22015863,5526801259,5);
INSERT INTO NATIONAL_DEPOT
(NDEPOT_ID,ADDRESS,ENQ_PHONE,PHONE,MANAGER_ID)
VALUES (3,'Dun laoghire',66767677,3425676548,9);
INSERT INTO NATIONAL_DEPOT
(NDEPOT_ID,ADDRESS,ENQ_PHONE,PHONE,MANAGER_ID)
VALUES (4,'Rathmines',55879658,5587589658,10);
INSERT INTO NATIONAL_DEPOT
(NDEPOT_ID,ADDRESS,ENQ_PHONE,PHONE,MANAGER_ID)
VALUES (5,'Woodbrook',66548765,7654215873,3);

INSERT INTO STOCK_LOCAL (QUANTITY,LDEPOT_ID,PROD_ID)
VALUES (3,1,1);
INSERT INTO STOCK_LOCAL (QUANTITY,LDEPOT_ID,PROD_ID)
```

```
VALUES (4,2,2);
INSERT INTO STOCK_LOCAL (QUANTITY,LDEPOT_ID,PROD_ID)
VALUES (5,1,3);
INSERT INTO STOCK_LOCAL (QUANTITY,LDEPOT_ID,PROD_ID)
VALUES (20,2,4);
INSERT INTO STOCK_LOCAL (QUANTITY,LDEPOT_ID,PROD_ID)
VALUES (15,2,3);
INSERT INTO STOCK_LOCAL (QUANTITY,LDEPOT_ID,PROD_ID)
VALUES (0,2,1);

INSERT INTO STOCK_NATIONAL (QUANTITY,NDEPOT_ID,PROD_ID)
VALUES (8,1,1);
INSERT INTO STOCK_NATIONAL (QUANTITY,NDEPOT_ID,PROD_ID)
VALUES (9,1,2);
INSERT INTO STOCK_NATIONAL (QUANTITY,NDEPOT_ID,PROD_ID)
VALUES (10,2,2);
INSERT INTO STOCK_NATIONAL (QUANTITY,NDEPOT_ID,PROD_ID)
VALUES (30,1,3);
INSERT INTO STOCK_NATIONAL (QUANTITY,NDEPOT_ID,PROD_ID)
VALUES (34,2,3);

INSERT INTO VEHICLE (REG_NO,MAKE,MODEL,LDEPOT_ID,NDEPOT_ID)
VALUES ('A123','HONDA','JAZZ',1,2);
INSERT INTO VEHICLE (REG_NO,MAKE,MODEL,LDEPOT_ID,NDEPOT_ID)
VALUES ('A456','MARUTI','SWIFT',2,1);
INSERT INTO VEHICLE (REG_NO,MAKE,MODEL,LDEPOT_ID,NDEPOT_ID)
VALUES ('A678','HONDA','CIVIC',2,2);
INSERT INTO VEHICLE (REG_NO,MAKE,MODEL,LDEPOT_ID,NDEPOT_ID)
VALUES ('A675','HYUNDAI','I10',1,2);
INSERT INTO VEHICLE (REG_NO,MAKE,MODEL,LDEPOT_ID,NDEPOT_ID)
VALUES ('A876','HYUNDAI','I20',1,1);
```

- **QUERYING TABLES**

Below are some queries written to fetch data from the tables created in Oracle schema.

/*Query1*/

```
SELECT LOCAL_DEPOT.LDEPOT_ID,VEHICLE.REG_NO,VEHICLE.MAKE,
VEHICLE.MODEL,MANAGER.MANAGER_ID FROM LOCAL_DEPOT
INNER JOIN VEHICLE ON LOCAL_DEPOT.LDEPOT_ID=VEHICLE.LDEPOT_ID
INNER JOIN MANAGER ON MANAGER.MANAGER_ID=LOCAL_DEPOT.MANAGER_ID;
```

OUTPUT:

Script Output × | Query Result ×

SQL | All Rows Fetched: 5 in 0.016 seconds

	LDEPOT_ID	REG_NO	MAKE	MODEL	MANAGER_ID
1	1	A123	HONDA	JAZZ	1
2	2	A456	MARUTI	SWIFT	2
3	2	A678	HONDA	CIVIC	2
4	1	A675	HYUNDAI	I10	1
5	1	A876	HYUNDAI	I20	1

/*Query2*/

```
SELECT LDEPOT_ID,AVG(QUANTITY) FROM STOCK_LOCAL
GROUP BY LDEPOT_ID;
```

OUTPUT:

Script Output × | Query... ×

SQL | All Rows Fetched: 2 in 0 seconds

	LDEPOT_ID	AVG(QUANTITY)
1	1	4
2	2	9.75

/*Query3*/

```
SELECT PRODUCT.PROD_ID,STOCK_LOCAL.QUANTITY,PRODUCT.PROD_DESC
FROM PRODUCT
INNER JOIN STOCK_LOCAL ON PRODUCT.PROD_ID=STOCK_LOCAL.PROD_ID;
```

OUTPUT:

Script Output × | Query Result ×

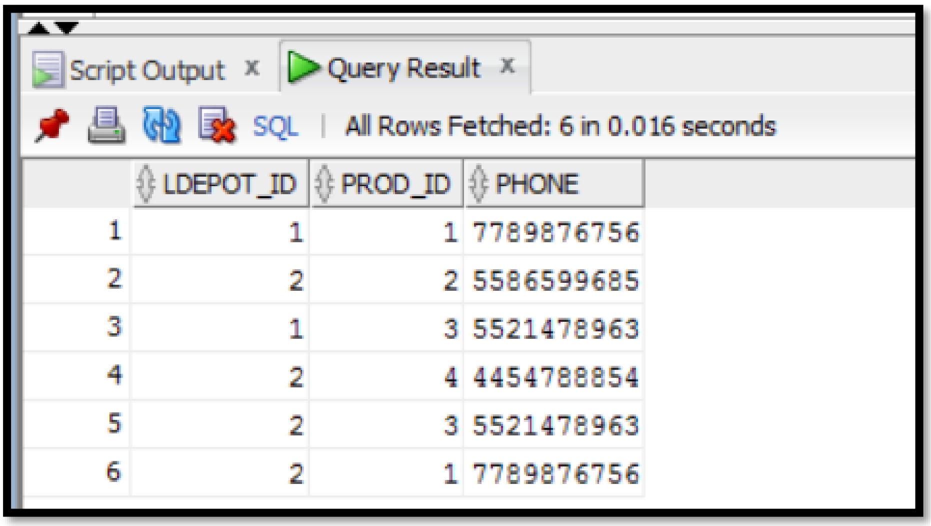
SQL | All Rows Fetched: 6 in 0 seconds

	PROD_ID	QUANTITY	PROD_DESC
1	1	3	RICE
2	2	4	SUGAR
3	3	5	PULSES
4	4	20	WHEAT
5	3	15	PULSES
6	1	0	RICE

/*Query4*/

```sql
SELECT STOCK_LOCAL.LDEPOT_ID,PRODUCT.PROD_ID,SUPPLIER.PHONE
FROM SUPPLIER
INNER JOIN PRODUCT ON PRODUCT.SUPP_ID=SUPPLIER.SUPP_ID
INNER JOIN STOCK_LOCAL ON STOCK_LOCAL.PROD_ID=PRODUCT.PROD_ID;
```

OUTPUT:

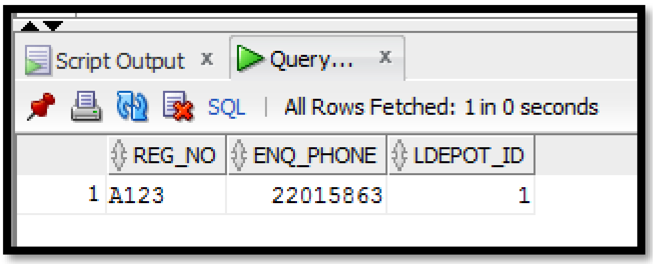


	LDEPOT_ID	PROD_ID	PHONE
1	1	1	7789876756
2	2	2	5586599685
3	1	3	5521478963
4	2	4	4454788854
5	2	3	5521478963
6	2	1	7789876756

/*Query5*/

```sql
SELECT VEHICLE.REG_NO,NATIONAL_DEPOT.ENQ_PHONE,LOCAL_DEPOT.LDEPOT_ID
FROM NATIONAL_DEPOT
INNER JOIN VEHICLE ON VEHICLE.NDEPOT_ID=NATIONAL_DEPOT.NDEPOT_ID
INNER JOIN LOCAL_DEPOT ON LOCAL_DEPOT.LDEPOT_ID=VEHICLE.LDEPOT_ID
WHERE VEHICLE.REG_NO='A123';
```

OUTPUT:



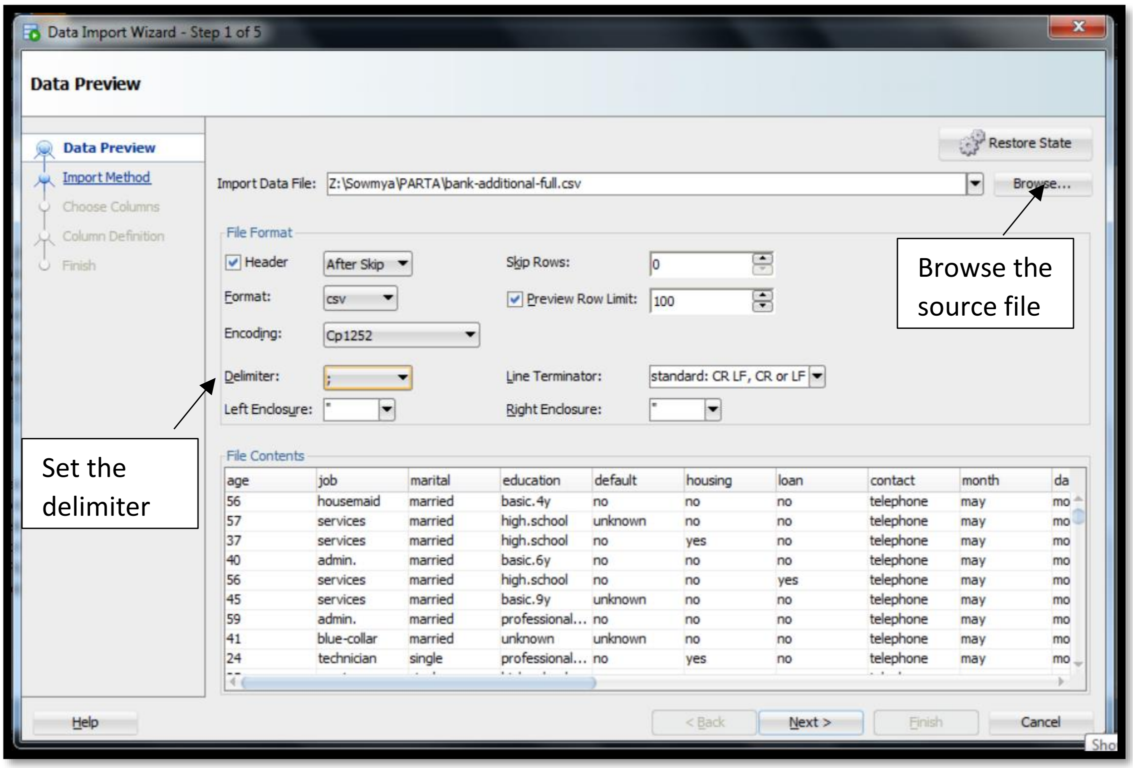
	REG_NO	ENQ_PHONE	LDEPOT_ID
1	A123	22015863	1

PART B

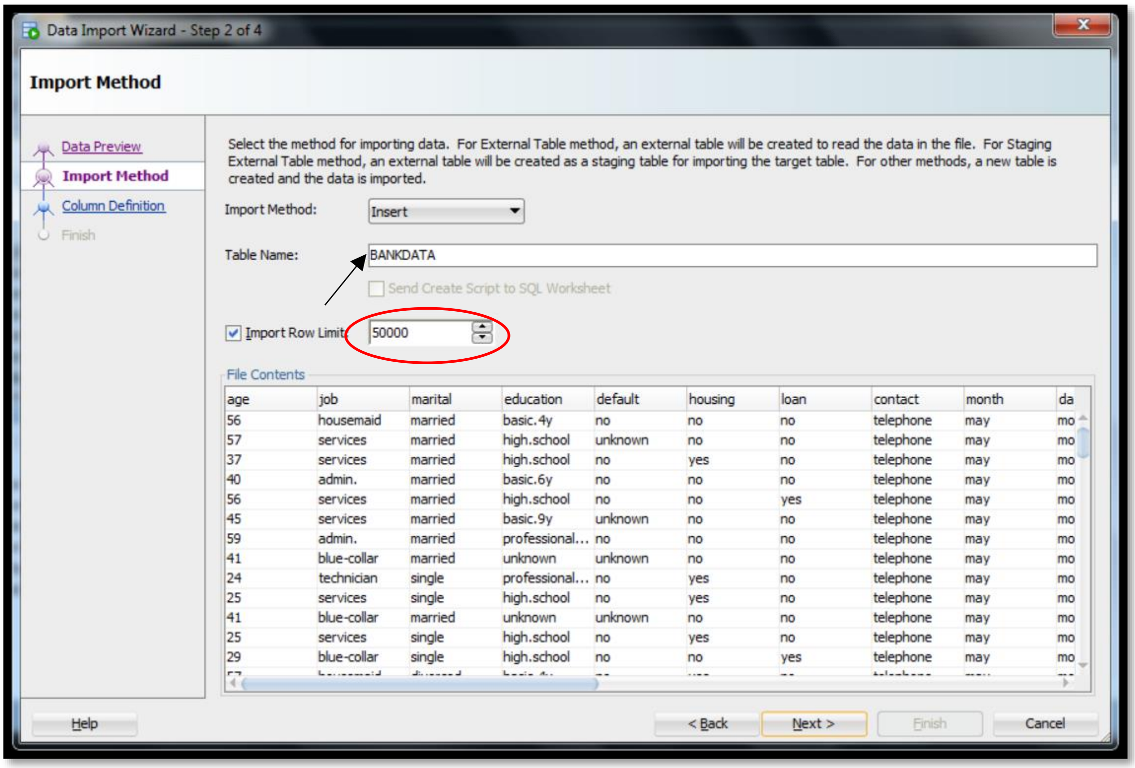
This part of the project aims at statistical analysis of a well-known dataset **Portuguese bank data** by using different available statistical functions. The dataset contains 20 different columns with 1 target variable 'y'.

The dataset is downloaded from https://archive.ics.uci.edu/ml/datasets/Bank+Marketing. and imported to Oracle SQL Developer as documented below:

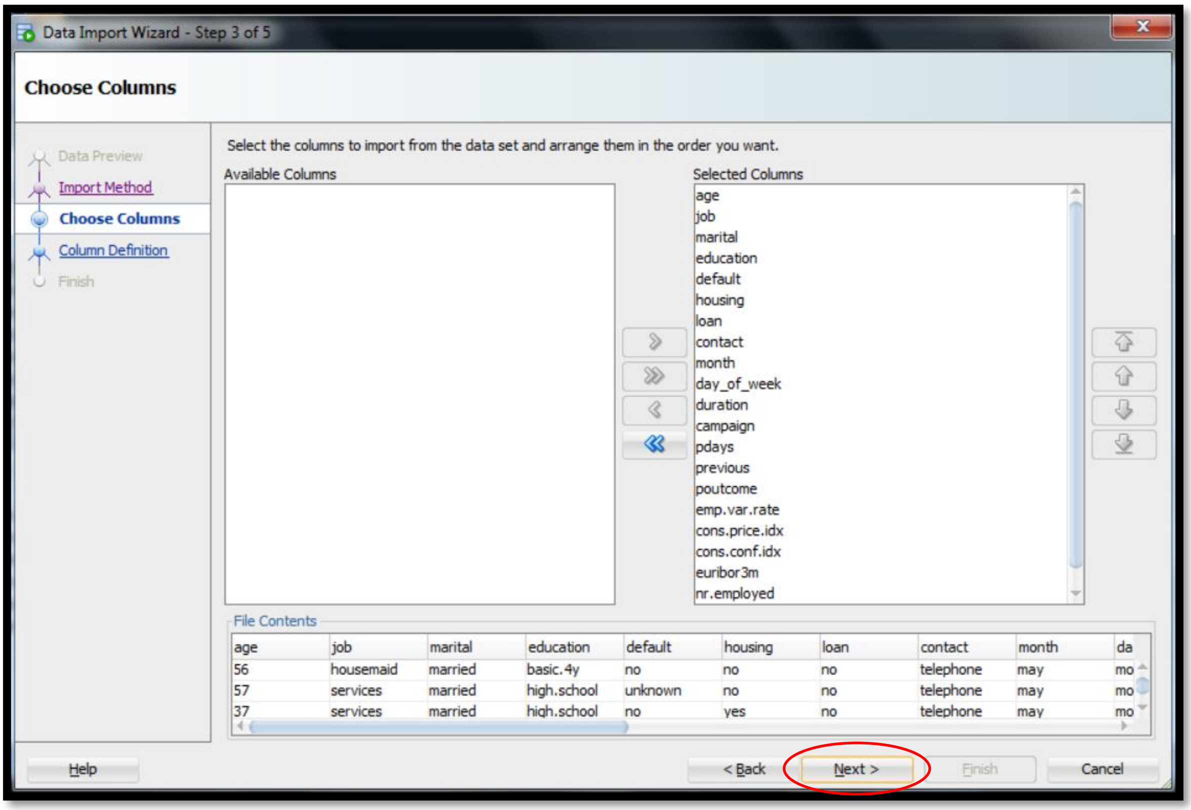
1. By right clicking on Tables we get ***import*** option to import the data into SQL Developer.



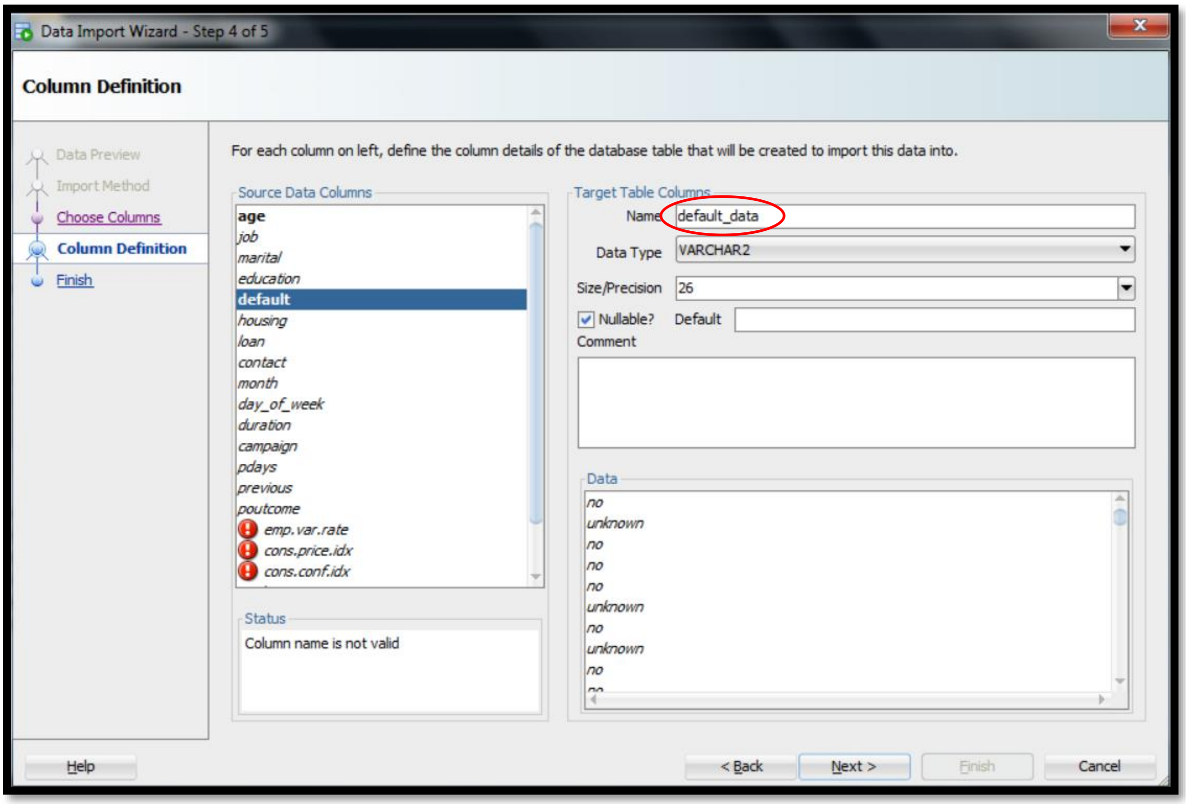
2. In next step, we can optionally set row limit and table name & proceed to next step.



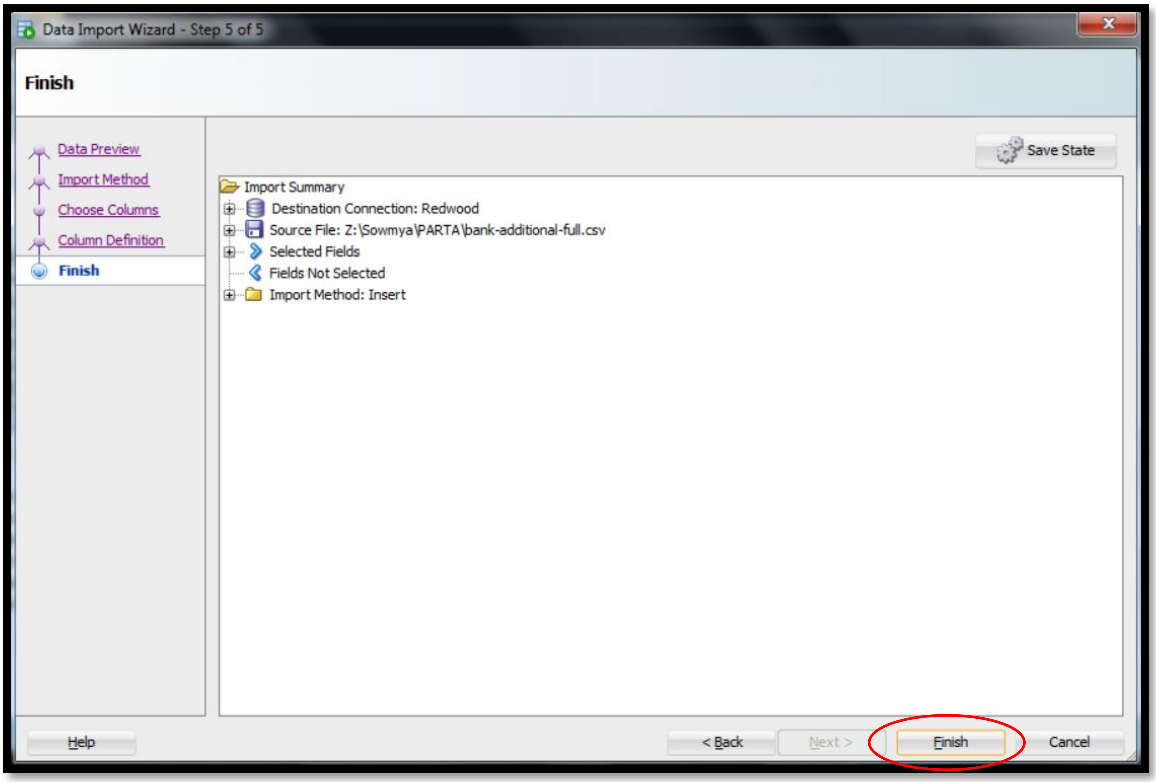
3. Out of 21 columns available from the dataset, we can choose the columns of our interest. Here, all columns have been selected.



4. When we import the data, some column names will not be compatable with the application (for example: some column names might be keywords). Such column names are replaced with new name.



5. Click finish in the last step.

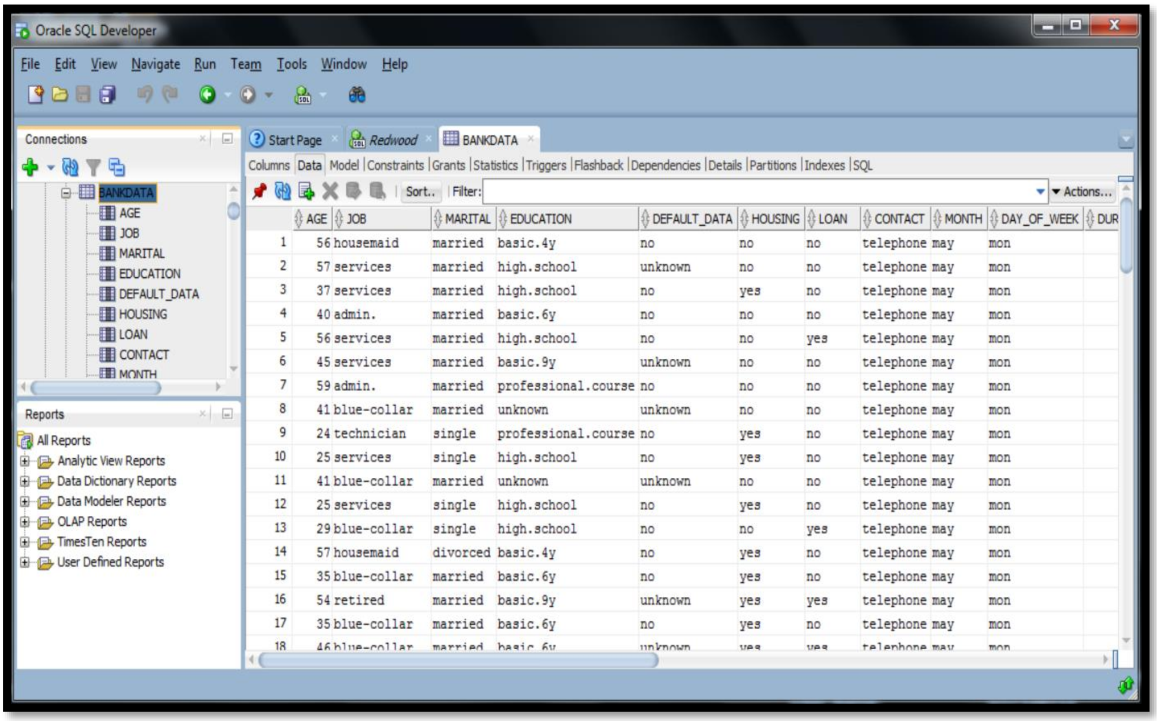


Once the process is completed, we will get a script for importing the data. This script is exeucted in the SQL Developer workspace. *(Script could not be attached since the code is very lengthy, sample of it has been recorded below)*

```
SET DEFINE OFF

CREATE TABLE BANKDATA ( age NUMBER(4),
job VARCHAR2(13),
marital VARCHAR2(8),
education VARCHAR2(19),
default_crdt VARCHAR2(26),
housing VARCHAR2(7),
loan VARCHAR2(7),
contact VARCHAR2(9),
month VARCHAR2(3),
day_of_week VARCHAR2(3),
duration NUMBER(4),
campaign NUMBER(1),
pdays NUMBER(3),
previous NUMBER(1),
poutcome VARCHAR2(11),
emp_var_rate1 NUMBER(4, 1),
cons_price_idx1 NUMBER(7, 3),
cons_conf_idx1 NUMBER(5, 1),
euribor3m NUMBER(4, 3),
```

```
nr_employed1 NUMBER(6),
y VARCHAR2(3));


INSERT INTO BANKDATA (age, job, marital, education, default_crdt,
housing, loan, contact, month, day_of_week, duration, campaign,
pdays, previous, poutcome, emp_var_rate1, cons_price_idx1,
cons_conf_idx1, euribor3m, nr_employed1, y)

VALUES (56.0, 'housemaid', 'married', 'basic.4y', 'no', 'no', 'no',
'telephone', 'may', 'mon', 261.0, 1.0, 999.0, 0.0, 'nonexistent',
1.1, 93.994, -36.4, 4.857, 5191.0, 'no');

INSERT INTO BANKDATA (age, job, marital, education, default_crdt,
housing, loan, contact, month, day_of_week, duration, campaign,
pdays, previous, poutcome, emp_var_rate1, cons_price_idx1,
cons_conf_idx1, euribor3m, nr_employed1, y)

VALUES (57.0, 'services', 'married', 'high.school', 'unknown', 'no',
'no', 'telephone', 'may', 'mon', 149.0, 1.0, 999.0, 0.0,
'nonexistent', 1.1, 93.994, -36.4, 4.857, 5191.0, 'no');

INSERT INTO BANKDATA (age, job, marital, education, default_crdt,
housing, loan, contact, month, day_of_week, duration, campaign,
pdays, previous, poutcome, emp_var_rate1, cons_price_idx1,
cons_conf_idx1, euribor3m, nr_employed1, y)

VALUES (37.0, 'services', 'married', 'high.school', 'no', 'yes',
'no', 'telephone', 'may', 'mon', 226.0, 1.0, 999.0, 0.0,
'nonexistent', 1.1, 93.994, -36.4, 4.857, 5191.0, 'no');

INSERT INTO BANKDATA (age, job, marital, education, default_crdt,
housing, loan, contact, month, day_of_week, duration, campaign,
pdays, previous, poutcome, emp_var_rate1, cons_price_idx1,
cons_conf_idx1, euribor3m, nr_employed1, y)

VALUES (40.0, 'admin.', 'married', 'basic.6y', 'no', 'no', 'no',
'telephone', 'may', 'mon', 151.0, 1.0, 999.0, 0.0, 'nonexistent',
1.1, 93.994, -36.4, 4.857, 5191.0, 'no');

INSERT INTO BANKDATA (age, job, marital, education, default_crdt,
housing, loan, contact, month, day_of_week, duration, campaign,
pdays, previous, poutcome, emp_var_rate1, cons_price_idx1,
cons_conf_idx1, euribor3m, nr_employed1, y)

VALUES (56.0, 'services', 'married', 'high.school', 'no', 'no',
'yes', 'telephone', 'may', 'mon', 307.0, 1.0, 999.0, 0.0,
'nonexistent', 1.1, 93.994, -36.4, 4.857, 5191.0, 'no');
```

Once we run the script, we can see the data imported into Oracle schema as shown below:

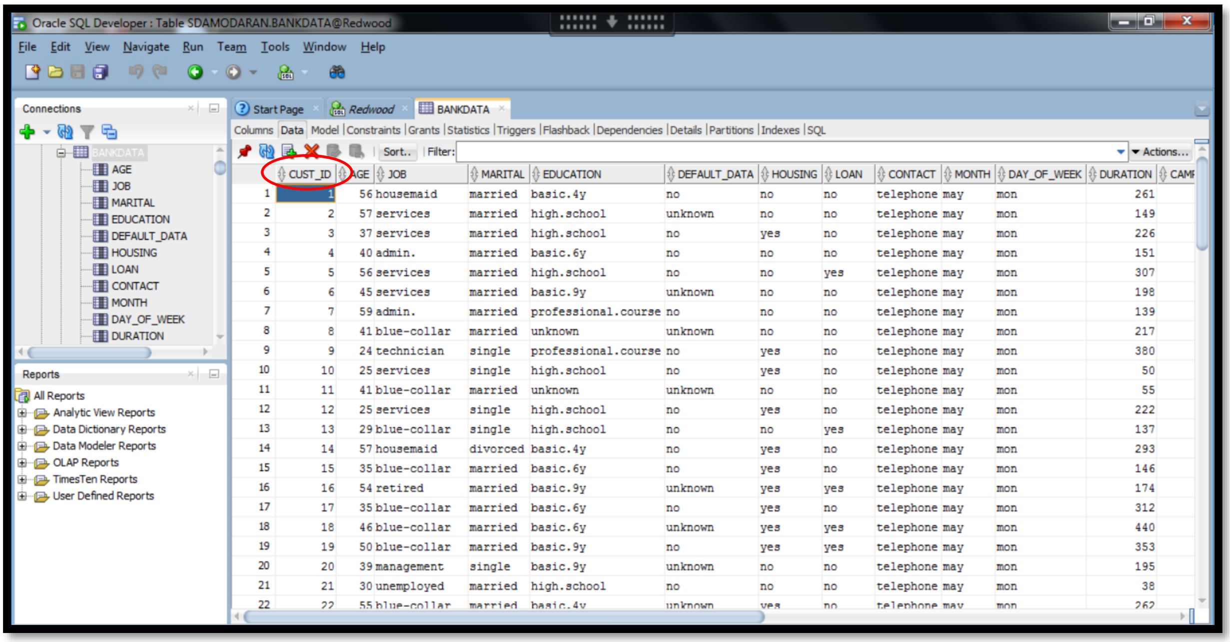


- **ANALYSIS OF IMPORTED DATA USING SQL STATISTICAL FUNCTIONS**

- Before analysing the data we have to prepare the same for making it easy to be analysed.
  Since the imported table BANKDATA does not contain any primary key, we will add a column to the table which uniquely identifies each of the observations in the table.
  Below is the SQL query used for adding primary key to BANKDATA:

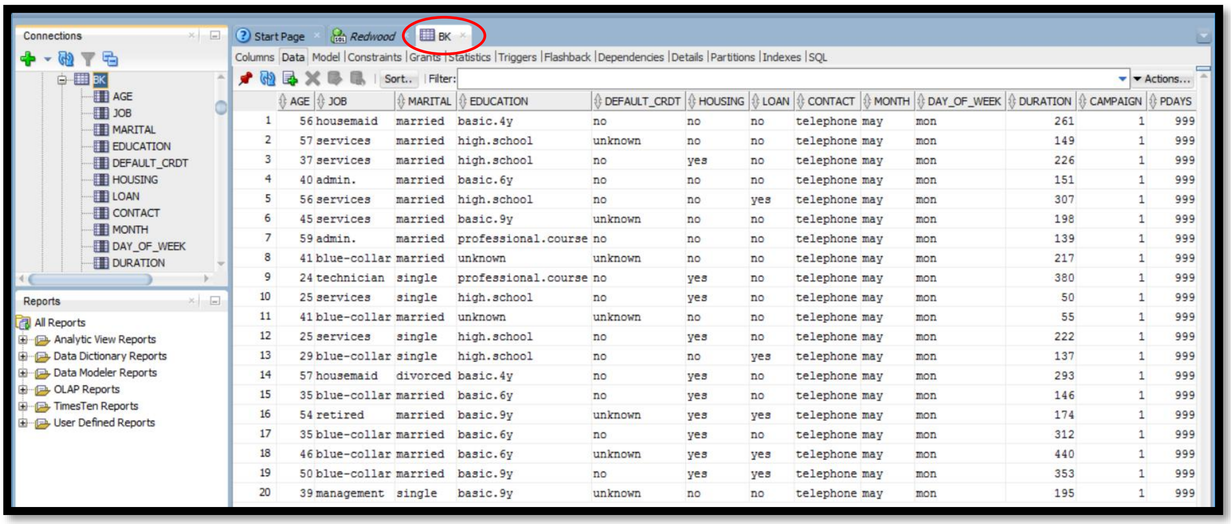
```
ALTER TABLE BANKDATA ADD cust_id integer generated by default on
null as identity;
update BANKDATA set cust_id=rownum;
alter table BANKDATA modify cust_id generated always as identity
start with limit value ;
alter table BANKDATA add constraint cust_id primary key (cust_id);
```

Now we can see the primary key column **'cust_id'** added to table BANKDATA.



- And also, for the demonstration purpose we query only top 20 records in BANKDATA table. Hence a separate table **BK** has been created which holds the top 20 records in BANKDATA table.

```
CREATE TABLE BK AS SELECT *  FROM PORTUGUESE_BANK WHERE ROWNUM<=20;

SELECT  * FROM BK;
```

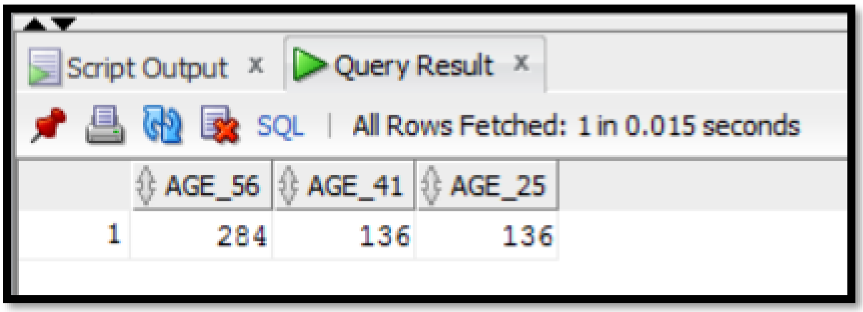


- Below are some SQL statistical and analytical queries used to analyse the sample table BK.

/*Query1*/

```
SELECT * FROM (SELECT age,duration from BK)
PIVOT (AVG(duration) for (age)
IN (56 as age_56, 41 as age_41, 25 as age_25));
```

OUTPUT:

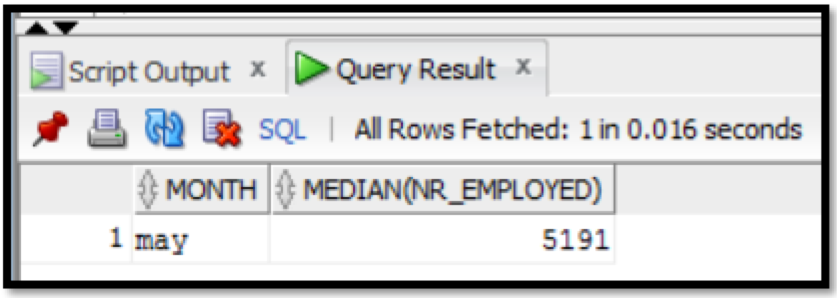


PIVOT FUNCTION:
The pivot function is an analytical function in SQL which when used turns the unique values in a single column of a table into multiple columns in the output and performs necessary aggregations on remaining, selected column values that are wanted in the output. Here we are fetching **avergae *duration* for 3 unique values in *age* column**. In the output we can see that, 3 unique values of *age* column has been converted into **3 separate columns with corresponding average values of *duration* .**

/*Query2*/

```
SELECT month,MEDIAN(nr_employed) FROM BK
GROUP BY month;
```

OUTPUT:



MEDIAN FUNCTION:
Median is a statitstical SQL function which calculates the median (value seperating higher half from lower half of a sorted data sample) of series of numbers.
Here we are calculating the **median of *nr_employed*** column by using GROUP BY function for column *month.* Since we have only 1 value for month in first 20 records of BANKDATA, we get a single group for output.

/*Query3*/

```
SELECT age, job,
LEAD (age, 1) OVER (ORDER BY age) AS "Nextage"
FROM BK
ORDER BY age;
```

OUTPUT:

Script Output × Query Result ×

SQL | All Rows Fetched: 20 in 0.016 seconds

	AGE	JOB	Nextage
1	24	technician	25
2	25	services	25
3	25	services	29
4	29	blue-collar	35
5	35	blue-collar	35
6	35	blue-collar	37
7	37	services	39
8	39	management	40
9	40	admin.	41
10	41	blue-collar	41
11	41	blue-collar	45
12	45	services	46
13	46	blue-collar	50
14	50	blue-collar	54
15	54	retired	56
16	56	services	56
17	56	housemaid	57
18	57	services	57
19	57	housemaid	59
20	59	admin.	(null)

LEAD FUNCTION:

Lead is an analytical function of SQL which returns the value from the next row of a selected column in the table. Here, **Lead function has been used for column *age***. The third column is added to the output which provides lead function values of *age* column. The OVER keyword specifies the column on which lead function has to be applied. In this scenario, the **last value** in the lead output column will be **NULL in default,** this value can be set manually. We can also specify the physical offset for getting the lead value , from the current row in the table. If nothing is specified the defualt is considered as **'1'**.

/*Query4*/

```
SELECT education,BK_ID,
LAG(education,1) OVER (ORDER BY education) As "Previouseducation"
FROM BK
ORDER BY education;
```

OUTPUT:

	CUST_ID	EDUCATION	Previouseducation
1	1	basic.4y	(null)
2	14	basic.4y	basic.4y
3	17	basic.6y	basic.4y
4	18	basic.6y	basic.6y
5	15	basic.6y	basic.6y
6	4	basic.6y	basic.6y
7	19	basic.9y	basic.6y
8	6	basic.9y	basic.9y
9	20	basic.9y	basic.9y
10	16	basic.9y	basic.9y
11	13	high.school	basic.9y
12	12	high.school	high.school
13	10	high.school	high.school
14	5	high.school	high.school
15	3	high.school	high.school
16	2	high.school	high.school
17	9	professional.course	high.school
18	7	professional.course	professional.course
19	8	unknown	professional.course
20	11	unknown	unknown

LAG FUNCTION:

Lag is an analytical function of SQL which returns the value from the previous row of a selected column in the table. Here, **Lag function has been used for column *education***. The third column is added to the output which provides lag function values of *education* column. The OVER keyword specifies the column on which lag function has to be applied. In this scenario, the **first value** in the lag output column will be **NULL in default,** this value can be set manually. We can also specify the physical offset for getting the lag value , from the current row in the table. If nothing is specified the defualt is considered as **'1'**.

/*Query5*/

```
SELECT job,cust_id, duration, CUME_DIST()
OVER (PARTITION BY job ORDER BY duration) AS cume_dist
FROM BK
WHERE job='services';
```

OUTPUT:

Script Output × | Query Result ×

SQL | All Rows Fetched: 6 in 0 seconds

	JOB	CUST_ID	DURATION	CUME_DIST
1	services	10	50	0.1666666666666666666666666666666666666667
2	services	2	149	0.3333333333333333333333333333333333333333
3	services	6	198	0.5
4	services	12	222	0.6666666666666666666666666666666666666667
5	services	3	226	0.8333333333333333333333333333333333333333
6	services	5	307	1

CUME_DIST FUNCTION:

Cume_dist analytical function of SQL returns the cumulative distribution of a variable. Here the query calculated the *duration* percentile for each customer who are employees with *job* = **'services'**. For example: 50% of employees with services job have duration value less than or equal to that of customer with *cust_id* '6'.

/*Query6*/

```
SELECT job, cust_id, housing, loan,
RANK() OVER (PARTITION BY job
ORDER BY housing, loan) "Rank"
FROM BK WHERE job = 'blue-collar';
```

OUTPUT:

Script Output × | Query Result ×

SQL | All Rows Fetched: 7 in 0.015 seconds

	JOB	CUST_ID	HOUSING	LOAN	Rank
1	blue-collar	8	no	no	1
2	blue-collar	11	no	no	1
3	blue-collar	13	no	yes	3
4	blue-collar	15	yes	no	4
5	blue-collar	17	yes	no	4
6	blue-collar	18	yes	yes	6
7	blue-collar	19	yes	yes	6

RANK FUNCTION:

Rank analytical function of SQL calculates the rank of a variable with respect to one more defined columns. It returns a number which ranks the specified variable. Here the query ranks the *cust_id* working in a **'blue-collar'** *job* with respect to their *housing* and *personal* loans taken from the Protuguese bank. Customers with identical housing and personal loans receive same ranks.

/*Query7*/

```
SELECT job, MIN(age) as MIN_AGE,MAX(age)as MAX_AGE
FROM BK
GROUP BY job
ORDER BY job;
```

OUTPUT

Script Output | Query Result

SQL | All Rows Fetched: 7 in 0.016 seconds

	JOB	MIN_AGE	MAX_AGE
1	admin.	40	59
2	blue-collar	29	50
3	housemaid	56	57
4	management	39	39
5	retired	54	54
6	services	25	57
7	technician	24	24

MIN and MAX FUNCTIONS:

MIN and MAX are statistical SQL functions which returns the minimum and maximum values of a particular variable. Here, the query returns the **minimum and maximum *age*** of bank customers employed in each of the *job* service type and the same has been ordered by job.

/*Query8*/

```
SELECT cust_id,duration, RATIO_TO_REPORT(duration)
OVER () AS Ratio_to_Report FROM BK
WHERE marital = 'married'
ORDER BY Ratio_to_Report;
```

OUTPUT:

Script Output × | Query Result ×

SQL | All Rows Fetched: 14 in 0.016 seconds

	CUST_ID	DURATION	RATIO_TO_REPORT
1	11	55	0.0175831202046035805626598465473145780051
2	7	139	0.0444373401534526854219948849104859335038
3	15	146	0.0466751918158567774936061381074168797954
4	2	149	0.0476342710997442455242966751918158567775
5	4	151	0.0482736572890025575447570332480818414322
6	16	174	0.0556265984654731457800511508951406649616
7	6	198	0.0632992327365728900255754475703324808184
8	8	217	0.0693734015345268542199488491048593350384
9	3	226	0.0722506393861892583120204603580562659847
10	1	261	0.0834398976982097186700767263427109974425
11	5	307	0.0981457800511508951406649616368286445013
12	17	312	0.0997442455242966751918158567774936061381
13	19	353	0.1128516624040920716112531969309462915601
14	18	440	0.1406649616368286445012787723785166240409

RATIO_TO_REPORT FUNCTION:

Ratio_To_Report is an analytical SQL function which calculates ratio of a value to the sum of values available in the particular column. Here, the query returns the ratio-to-report value of each **married** customer's *duration* to the total of all married customer's duration.

PART C

In this part of the project, machine learning prediction models have been built to predict the target customers for Portuguese Bank term deposits, by using the in-database machine learning functions of the Oracle SQL.

The goal here is to predict the customers who will get subscribed to the new service product of the Portuguese bank i.e. '*term deposits*'. By predicting these customers the bank can concentrate on marketing the term deposits to only those customers who are predicted to buy the term deposits, instead of contacting each and every customer. At the same time the bank can reduce the investment in marketing and avoid interupting each and every valued customer of the bank. The predicting variable here in the dataset is represented as **'y'** which contains a binary value 'yes' representing the customer might subscribe to term deposit and 'no' representing the customer might not subscribe to term deposit.

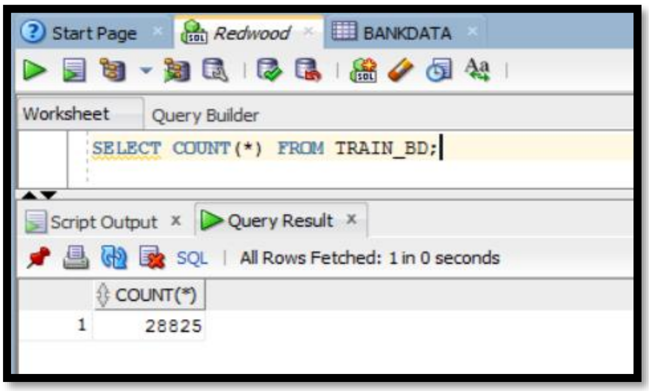
- ## DATA PARTITON

Before starting to create machine learning models, it is aways important to divide the dataset into **TRAIN** and **TEST** data. The data splitting is usually done in machine learning for **cross-validation purposes**. One portion(TRAIN) of data is used to train the model and another portion(TEST) is used to test the model, so that we will make sure that the model we have built is not fitting just to the train data which we have used. It is a way of evaluating the built model. Typically when we split a data, a larger portion is used for training **(70% here)** and smaller portion is used for testing **(30% here)**. The SQL function SAMPLE which is used in the query, splits the data randomly in 70:30 proportion. After the model has been built we will evaluate its performance by predicting the target variable **'y'** against the test data. Below is the SQL query used for data partition.

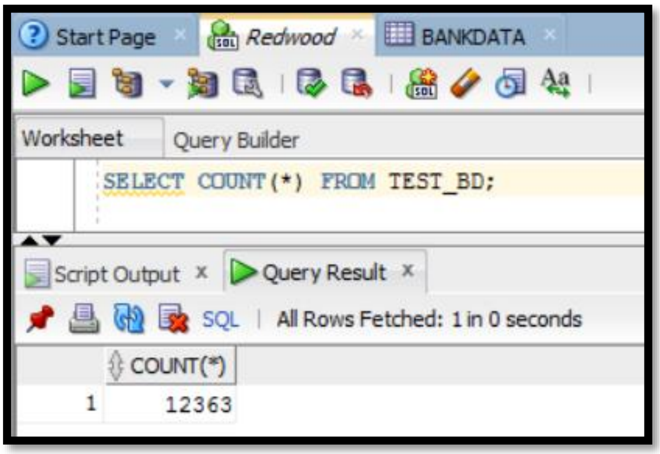
```
CREATE TABLE TRAIN_BD AS SELECT * FROM BANKDATA
SAMPLE (70) SEED (1);
```

We can check the Train (TRAIN_BD) and Test (Test_BD) data tables now.

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




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



- **BUILDING MACHINE LEARNING MODELS**

For creating data mining models in Oracle PL/SQL we are provided with a package called **DBMS_DATA_MINING** which is an API for creating and evaluating data mining models.It provides functions for both supervised and unsupervised learning models.
Since the scenario we are dealing with is a classification problem, **classification** data mining function has been used here, which uses historical data to predict the target variable.

Different classification algorithms are available in Oracle namely ***Decision tree***, ***Support Vector Machine,Logistic Regression*** and ***Naïve Bayes*** as default method. Here we are using 3 of these algorithms to build our predictive model.

1. **Decision Tree**

Decision tree is a supervised learning algorithm which uses tree like structures to analyse between the columns of the dataset and predict the target value 'y'. It is used very often to solve classification problems. The variables which predict the target variable 'y' is selected automatically by the algorithm based on different parameters like *gain* for example. Decision tree uses particular rules while splitting the trees at nodes and keeps on splitting the same until the target variable belongs to one of its branches.

For creating a decision tree in Oracle SQL, below mentioned functions have been used.

- Initially a setting table has been created which holds the settings for decision tree model as shown below.

```
CREATE TABLE decision_tree_model_settings (
setting_name VARCHAR2(30),
setting_value VARCHAR2(30));
```

```
BEGIN
INSERT INTO decision_tree_model_settings (setting_name,
setting_value)
VALUES
(dbms_data_mining.algo_name,dbms_data_mining.algo_decision_tree);
INSERT INTO decision_tree_model_settings (setting_name,
setting_value)
VALUES (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);
COMMIT;
END;
```

- **setting_name** contains the name of setting parameter.
- **setting_value** contains the value of setting parameter.

Here we are specifying the algorithm that we are using as **algo_decision_tree** and enabling the function ADP(Automatic Data Preparation) .

- The second step is to create the model using CREATE_MODEL procedure.

```
BEGIN
 DBMS_DATA_MINING.CREATE_MODEL(
 model_name => 'BANKDATA_DT_MODEL1',
 mining_function => dbms_data_mining.classification,
 data_table_name => 'TRAIN_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 settings_table_name => 'decision_tree_model_settings');
END;
```

The above query intakes all the details like the data **model_name**, **mining_function**, **data_table_name** which is used to train the model, primary key of the table as **case_id_column_name**, target variable as **target_column_name** and **settings_table_name** which was created in initial step. As a result, a decision tree model **BANKDATA_DT_MODEL1** has been built.

In addition we can also see which attribiutes were used as best predictors in building the model, as shown below:

```
SELECT attribute_name,
attribute_type,
usage_type,
target
FROM all_mining_model_attributes
WHERE model_name = 'BANKDATA_DT_MODEL1';
```

OUTPUT:

Script Output × Query... ×

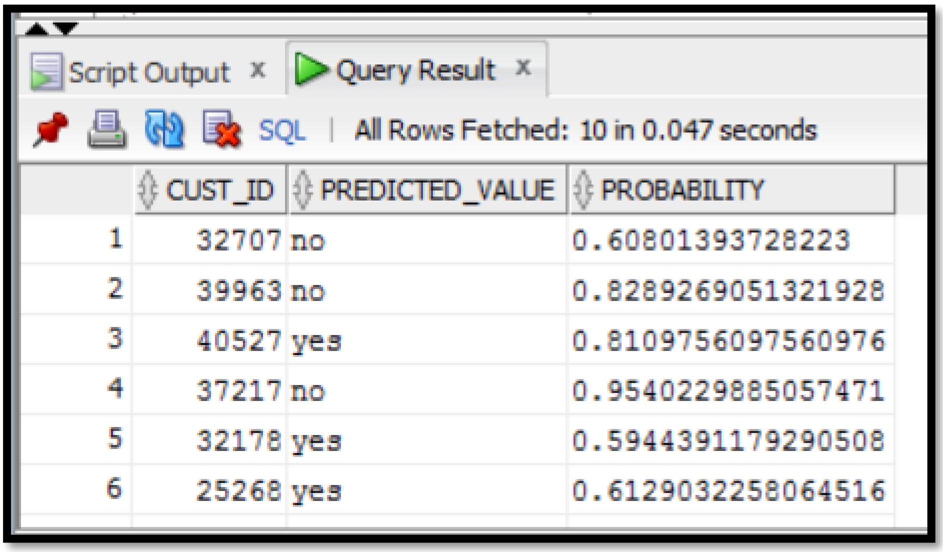
SQL | All Rows Fetched: 12 in 0.046 seconds

	ATTRIBUTE_NAME	ATTRIBUTE_TYPE	USAGE_TYPE	TARGET
1	PDAYS	NUMERICAL	ACTIVE	NO
2	POUTCOME	CATEGORICAL	ACTIVE	NO
3	JOB	CATEGORICAL	ACTIVE	NO
4	DURATION	NUMERICAL	ACTIVE	NO
5	AGE	NUMERICAL	ACTIVE	NO
6	EURIBOR3M	NUMERICAL	ACTIVE	NO
7	CONS_PRICE_IDX	NUMERICAL	ACTIVE	NO
8	NR_EMPLOYED	NUMERICAL	ACTIVE	NO
9	CONTACT	CATEGORICAL	ACTIVE	NO
10	CONS_CONF_IDX	NUMERICAL	ACTIVE	NO
11	MONTH	CATEGORICAL	ACTIVE	NO
12	Y	CATEGORICAL	ACTIVE	YES

- The next step includes evaluating the built model with test data which is accomplished as shown below:

```
CREATE VIEW BANKDATA_dt_test_results
AS
SELECT CUST_ID,
prediction(BANKDATA_DT_MODEL1 USING *) predicted_value,
prediction_probability(BANKDATA_DT_MODEL1 USING *) probability
FROM TEST_BD;
SELECT *
FROM BANKDATA_dt_test_results
WHERE rownum <=10;
```

The test results are as below:

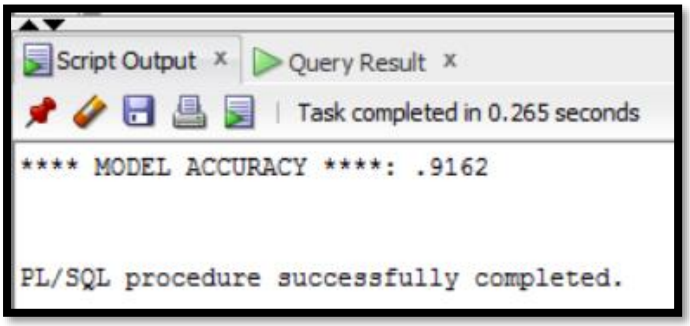


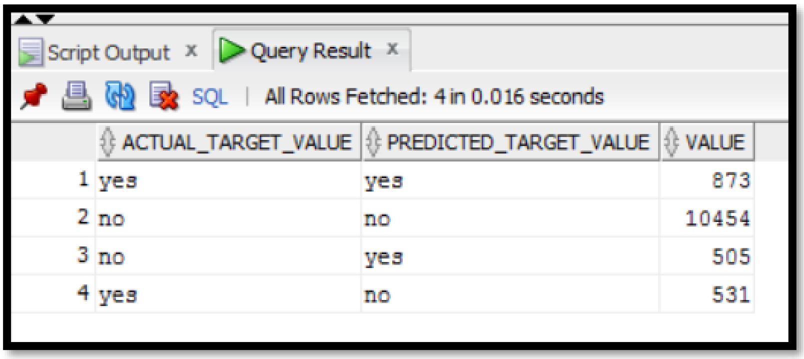
	CUST_ID	PREDICTED_VALUE	PROBABILITY
1	32707	no	0.60801393728223
2	39963	no	0.828926905132192
3	40527	yes	0.810975609756097
4	37217	no	0.954022988505747
5	32178	yes	0.594439117929050
6	25268	yes	0.612903225806451

- Once we get the test results, we can create a confusion matrix from the results obtained. COMPUTE_CONFUSION matrix procedure has been used for the same.

```
set serveroutput on
DECLARE
 v_accuracy NUMBER;
BEGIN
 DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
 accuracy => v_accuracy,
 apply_result_table_name => 'BANKDATA_dt_test_results',
 target_table_name => 'TEST_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 confusion_matrix_table_name => 'BANKDATA_DT_confusion_matrix1',
 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 BANKDATA_DT_confusion_matrix1;
```

OUTPUT:





	ACTUAL_TARGET_VALUE	PREDICTED_TARGET_VALUE	VALUE
1	yes	yes	873
2	no	no	10454
3	no	yes	505
4	yes	no	531

From the output, we can make out that the decision tree which we have created has an accuracy of **0.916** in predicting the customers who will get subscribed to term deposits of Portuguese bank.

- Calculating LIFT for the decision tree model.

LIFT is a measure of evaluation of a data model which explains the performance of a data model at predicting the target value against a random choice of model. LIFT value in SQL is as calculated below. COMPUT_LIFT procedure has been used for the same.

```
BEGIN
 DBMS_DATA_MINING.COMPUTE_LIFT (
 apply_result_table_name => 'BANKDATA_dt_test_results',
 target_table_name => 'TEST_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 lift_table_name => 'BANKDATA_DT_LIFT',
 positive_target_value => 'yes',
 score_column_name => 'PREDICTED_VALUE',
 score_criterion_column_name => 'PROBABILITY',
 num_quantiles => 10,
 cost_matrix_table_name => null,
 apply_result_schema_name => null,
 target_schema_name => null,
 cost_matrix_schema_name => null,
 score_criterion_type => 'PROBABILITY');
END;

SELECT quantile_number,probability_threshold,gain_cumulative,
quantile_total_count
FROM BANKDATA_DT_LIFT;
```

OUTPUT:

Script Output × Query Result ×

SQL | All Rows Fetched: 10 in 0.016 seconds

	QUANTILE_NUMBER	PROBABILITY_THRESHOLD	GAIN_CUMULATIVE	QUANTILE_TOTAL_COUNT
1	1	0.8109756097560976	0.1306834357437535738831615120274914089347	138
2	2	0.8109756097560976	0.2613668714875071592210767468499427262314	138
3	3	0.5944391179290508	0.3615089232978951890034364261168384879725	138
4	4	0.5944391179290508	0.4523487418787585910652920962199312714777	138
5	5	0.5944391179290508	0.5431885604596219931271477663230240549828	138
6	6	0.5944391179290508	0.634028379040485395189003436426116838488	138

- Calculating ROC(Reciever Operating Characteristics) Table and Area Under Curve(AUC) for decision tree.

COMPUTE_ROC procedure has been used here to compute the ROC table and AUC measure of the model.

```
set serveroutput on
DECLARE
 v_area_under_curve NUMBER;
BEGIN
 DBMS_DATA_MINING.COMPUTE_ROC (
 roc_area_under_curve => v_area_under_curve,
 apply_result_table_name => 'BANKDATA_dt_test_results',
 target_table_name => 'TEST_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 roc_table_name => 'BANKDATA_DT_ROC',
 positive_target_value => 'yes',
 score_column_name => 'PREDICTED_VALUE',
 score_criterion_column_name => 'PROBABILITY');
 DBMS_OUTPUT.PUT_LINE('**** AREA UNDER ROC CURVE ****: ' ||
 ROUND(v_area_under_curve,4));
END;

SELECT  probability,true_positive_fraction,false_positive_fraction
FROM BANKDATA_DT_ROC;
```

OUTPUT:

Script Output × | Query Result ×

Task completed in 0.296 seconds

```
Table BANKDATA_DT_ROC dropped.

**** AREA UNDER ROC CURVE ****: .5943


PL/SQL procedure successfully completed.
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 6 in 0.031 seconds

	PROBABILITY	TRUE_POSITIVE_FRACTION	FALSE_POSITIVE_FRACTION
1	0.8109756097560976	0.2623138602520045819014891179839633447881	0.0950495049504950495049504950495049504950495
2	0.66	0.2920962199312714776632302405498281786942	0.1207920792079207920792079207920792079208
3	0.6129032258064516	0.3562428407789232531500572737686139747995	0.1881188118811881188118811881188118811881
4	0.5944391179290508	0.6471935853379152348224513172966781214204	0.5603960396039603960396039603960396039604
5	0.5617529880478087	0.9404352806414662084765177548682703321879	0.9029702970297029702970297029702970297029703
6	0.5546558704453441	1	1

2. **Support Vector Machine**

Support Vector Machine is a supersived machine learning algorithm that can be used to solve classification problems. The basic idea of SVM is it forms a hyper plane between the classes of a dataset.

The SVM model in Oracle is built as expalined below:

- Initially a setting table has been created which holds the settings for decision tree model as shown below.

```
CREATE TABLE svm_model_settings
( setting_name VARCHAR2(30),
 setting_value VARCHAR2(4000));

BEGIN
 INSERT INTO svm_model_settings (setting_name, setting_value)
 values (dbms_data_mining.algo_name,
dbms_data_mining.algo_support_vector_machines);
 INSERT INTO svm_model_settings (setting_name, setting_value)
 VALUES (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);
END;
```

- The second step is to create the model using CREATE_MODEL procedure.

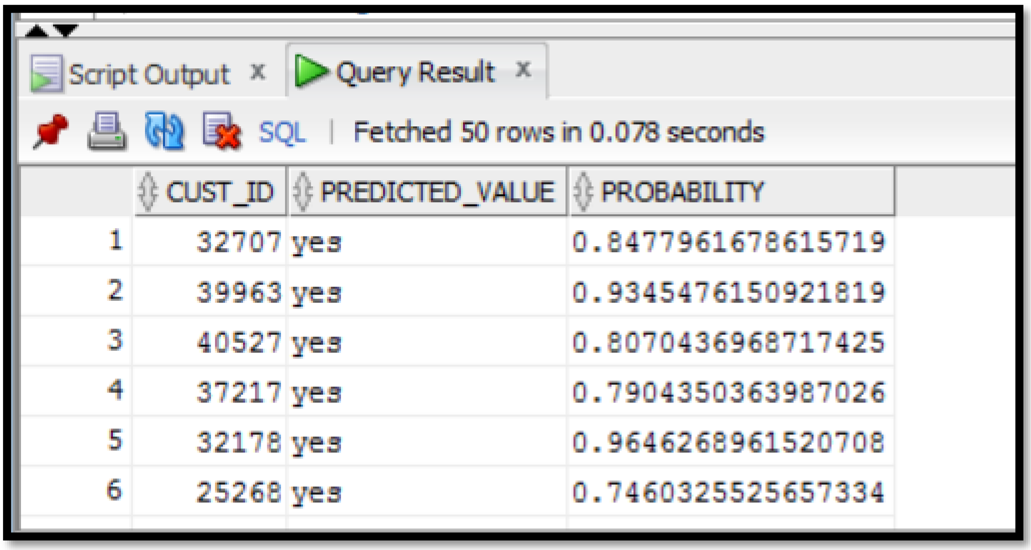
```
BEGIN
 DBMS_DATA_MINING.CREATE_MODEL(
 model_name => 'BANKDATA_SVM_MODEL1',
 mining_function => dbms_data_mining.classification,
 data_table_name => 'TRAIN_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 settings_table_name => 'svm_model_settings');
END;
```

As a result, a SVM model **BANKDATA_SVM_MODEL1** has been built.

- The next step includes evaluating the built model with test data which is accomplished as shown below:

```
CREATE VIEW BANKDATA_svm_test_results
AS
SELECT CUST_ID,
 prediction(BANKDATA_SVM_MODEL1 USING *) predicted_value,
 prediction_probability(BANKDATA_SVM_MODEL1 USING *) probability
FROM TEST_BD;

SELECT *
FROM BANKDATA_svm_test_results
WHERE rownum <=10;
```

The test results are shown as below:



	CUST_ID	PREDICTED_VALUE	PROBABILITY
1	32707	yes	0.8477961678615719
2	39963	yes	0.9345476150921819
3	40527	yes	0.8070436968717425
4	37217	yes	0.7904350363987026
5	32178	yes	0.9646268961520708
6	25268	yes	0.7460325525657334

- Once we get the test results, we can create a confusion matrix from the results obtained. COMPUTE_CONFUSION matrix procedure has been used for the same.

```
set serveroutput on
DECLARE
 v_accuracy NUMBER;
BEGIN
 DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
 accuracy => v_accuracy,
 apply_result_table_name => 'BANKDATA_svm_test_results',
 target_table_name => 'TEST_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 confusion_matrix_table_name => 'BANKDATA_SVM_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 BANKDATA_SVM_confusion_matrix;
```

OUTPUT:

Script Output × | Query Result ×

Task completed in 4.009 seconds

```
**** MODEL ACCURACY ****: .8582


PL/SQL procedure successfully completed.
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 4 in 0.016 seconds

	ACTUAL_TARGET_VALUE	PREDICTED_TARGET_VALUE	VALUE
1	yes	yes	803
2	no	no	9807
3	yes	no	601
4	no	yes	1152

From the output, the accuracy is calculated as **0.858** for Support Vector Machine.

- Calculating LIFT for the SVM model, COMPUT_LIFT procedure has been used for the same.

```
BEGIN
 DBMS_DATA_MINING.COMPUTE_LIFT (
 apply_result_table_name => 'BANKDATA_svm_test_results',
 target_table_name => 'TEST_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 lift_table_name => 'BANKDATA_SVM_LIFT',
 positive_target_value => 'yes',
 score_column_name => 'PREDICTED_VALUE',
 score_criterion_column_name => 'PROBABILITY',
 num_quantiles => 10,
 cost_matrix_table_name => null,
 apply_result_schema_name => null,
 target_schema_name => null,
 cost_matrix_schema_name => null,
 score_criterion_type => 'PROBABILITY');
END;

SELECT quantile_number,probability_threshold,gain_cumulative,
quantile_total_count
FROM BANKDATA_SVM_LIFT;
```

OUTPUT:

Script Output × Query Result ×

SQL | All Rows Fetched: 10 in 0.015 seconds

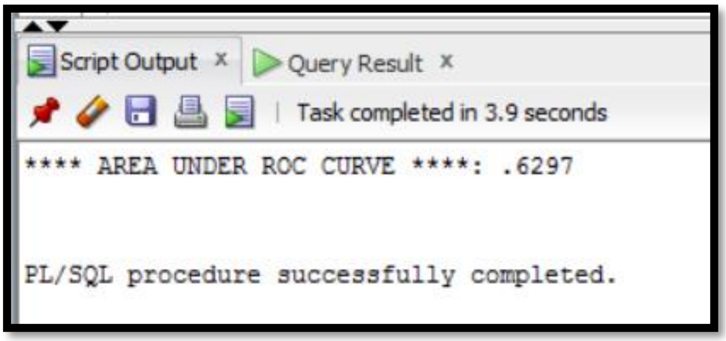
	QUANTILE_NUMBER	PROBABILITY_THRESHOLD	GAIN_CUMULATIVE	QUANTILE_TOTAL_COUNT
1	1	0.9066901752746234	0.1481942714819427148194271481942714819427	196
2	2	0.8710843743646948	0.2901618929016189290161892901618929016189	196
3	3	0.8383313221890845	0.4159402241594022415940224159402241594022	196
4	4	0.8051933102711688	0.5193026151930261519302615193026151930262	196
5	5	0.7629121917244907	0.6052303860523038605230386052303860523039	196
6	6	0.7199793215101162	0.7036114570361145703611457036114570361146	195

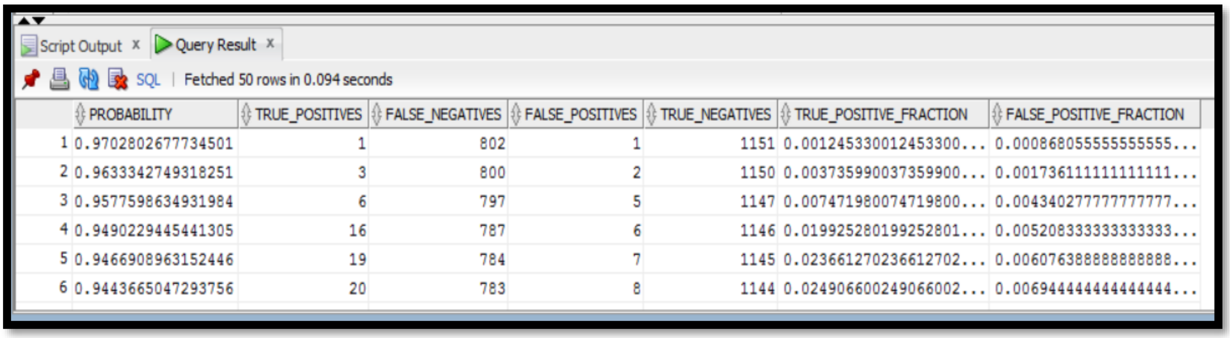
- Calculating ROC(Reciever Operating Characteristics) Table and Area Under Curve(AUC) for decision tree.

COMPUTE_ROC procedure has been used here to compute the ROC table and AUC measure of the model.

```
set serveroutput on
DECLARE
 v_area_under_curve NUMBER;
BEGIN
 DBMS_DATA_MINING.COMPUTE_ROC (
 roc_area_under_curve => v_area_under_curve,
 apply_result_table_name => 'BANKDATA_svm_test_results',
 target_table_name => 'TEST_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 roc_table_name => 'BANKDATA_SVM_ROC',
 positive_target_value => 'yes',
 score_column_name => 'PREDICTED_VALUE',
 score_criterion_column_name => 'PROBABILITY');
 DBMS_OUTPUT.PUT_LINE('**** AREA UNDER ROC CURVE ****: ' ||
 ROUND(v_area_under_curve,4));
END;

SELECT * FROM BANKDATA_SVM_ROC;
```

OUTPUT:





	PROBABILITY	TRUE_POSITIVES	FALSE_NEGATIVES	FALSE_POSITIVES	TRUE_NEGATIVES	TRUE_POSITIVE_FRACTION	FALSE_POSITIVE_FRACTION
1	0.970280267734501	1	802	1	1151	0.001245330012453300...	0.000868055555555555...
2	0.9633342749318251	3	800	2	1150	0.003735990037359900...	0.001736111111111111...
3	0.9577598634931984	6	797	5	1147	0.007471980074719800...	0.004340277777777777...
4	0.9490229445441305	16	787	6	1146	0.019925280199252801...	0.005208333333333333...
5	0.9466908963152446	19	784	7	1145	0.023661270236612702...	0.006076388888888888...
6	0.9443665047293756	20	783	8	1144	0.024906600249066002...	0.006944444444444444...

3. NAIVE BAYES

Naive Bayes is one of the classification algorithms which works collectively as a family of algorithms that uses Bayes' Theorem as basic principle. In Oracle, the Naive Bayes data mining model is built as explained below:

- Initially a setting table has been created which holds the settings for decision tree model as shown below.

```
CREATE TABLE nb_model_settings
( setting_name VARCHAR2(30),
 setting_value VARCHAR2(4000));

BEGIN
 INSERT INTO nb_model_settings (setting_name, setting_value)
 values (dbms_data_mining.algo_name,
dbms_data_mining.algo_naive_bayes);
 INSERT INTO nb_model_settings (setting_name, setting_value)
 VALUES (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);
END;
```

- The second step is to create the model using CREATE_MODEL procedure.

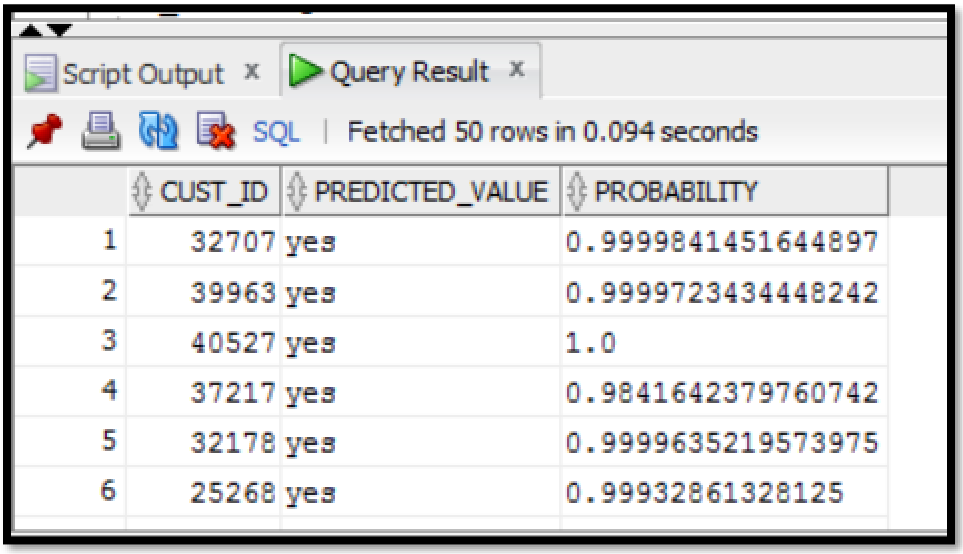
```
BEGIN
 DBMS_DATA_MINING.CREATE_MODEL(
 model_name => 'BANKDATA_NB_MODEL1',
 mining_function => dbms_data_mining.classification,
 data_table_name => 'TRAIN_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 settings_table_name => 'nb_model_settings');
END;
```

As a result a Naïve Bayes model **BANKDATA_NB_MODEL1** has been built.

- The next step includes evaluating the built model with test data which is accomplished as shown below:

```
CREATE OR REPLACE VIEW BANKDATA_nb_test_results
AS
SELECT CUST_ID,
 prediction(BANKDATA_NB_MODEL1 USING *) predicted_value,
 prediction_probability(BANKDATA_NB_MODEL1 USING *) probability
FROM TEST_BD;

SELECT *
FROM BANKDATA_nb_test_results
```

The test results are shown as below:

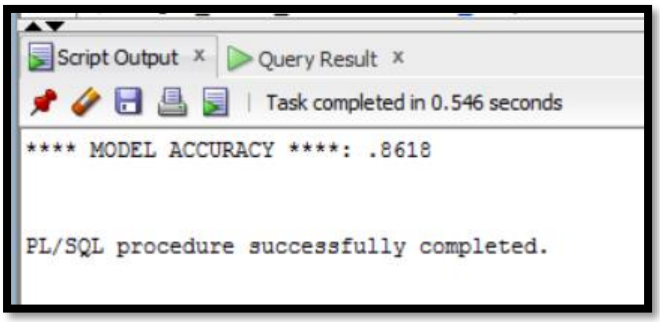


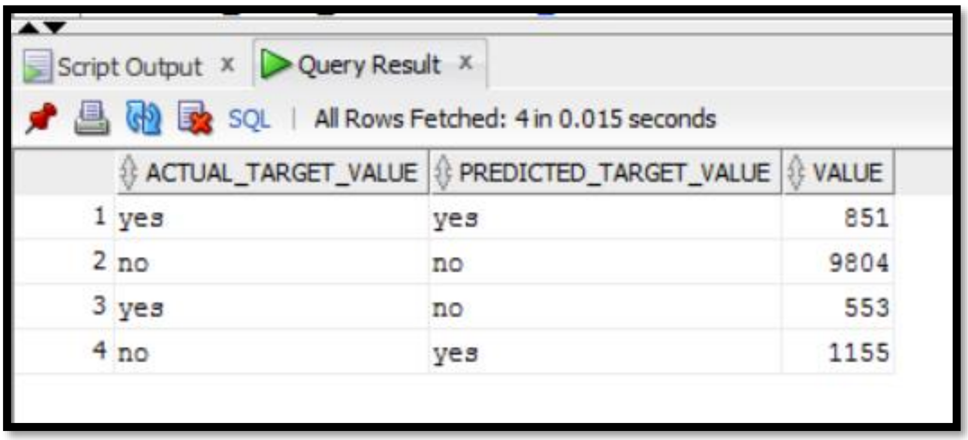
	CUST_ID	PREDICTED_VALUE	PROBABILITY
1	32707	yes	0.9999841451644897
2	39963	yes	0.9999723434448242
3	40527	yes	1.0
4	37217	yes	0.9841642379760742
5	32178	yes	0.9999635219573975
6	25268	yes	0.99932861328125

- Once we get the test results, we can create a confusion matrix from the results obtained. COMPUTE_CONFUSION matrix procedure has been used for the same.

```
set serveroutput on
DECLARE
 v_accuracy NUMBER;
BEGIN
 DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
 accuracy => v_accuracy,
 apply_result_table_name => 'BANKDATA_nb_test_results',
 target_table_name => 'TEST_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 confusion_matrix_table_name => 'BANKDATA_NB_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 BANKDATA_NB_confusion_matrix;
```

OUTPUT:





From the output, we can say that the accuracy of Naive Bayes model is **0.861**.

- Calculating LIFT for the Naïve Bayes model, COMPUT_LIFT procedure has been used for the same.

```
BEGIN
 DBMS_DATA_MINING.COMPUTE_LIFT (
 apply_result_table_name => 'BANKDATA_nb_test_results',
 target_table_name => 'TEST_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 lift_table_name => 'BANKDATA_NB_LIFT',
 positive_target_value => 'yes',
 score_column_name => 'PREDICTED_VALUE',
 score_criterion_column_name => 'PROBABILITY',
 num_quantiles => 10,
 cost_matrix_table_name => null,
 apply_result_schema_name => null,
 target_schema_name => null,
 cost_matrix_schema_name => null,
 score_criterion_type => 'PROBABILITY');
END;

SELECT quantile_number,probability_threshold,gain_cumulative,
quantile_total_count
FROM BANKDATA_NB_LIFT;
```

OUTPUT:

Script Output | Query...

SQL | All Rows Fetched: 10 in 0 seconds

	QUANTILE_NUMBER	PROBABILITY_THRESHOLD	GAIN_CUMULATIVE	QUANTILE_TOTAL_COUNT
1	1	0.9999997615814209	0.1918918954779762749706227967097532314924	201
2	2	0.9999924302101135	0.3403838501693779318448883666274970622797	201
3	3	0.9999494552612305	0.4418331374853113983548766157461809635723	201
4	4	0.9997902512550354	0.5464159811985898942420681551116333725029	201
5	5	0.9991827011108398	0.6509988249118683901292596944770857814336	201
6	6	0.9968986511230469	0.7332549941245593419506462984723854289072	201

- Calculating ROC(Reciever Operating Characteristics) Table and Area Under Curve(AUC) for decision tree.

COMPUTE_ROC procedure has been used here to compute the ROC table and AUC measure of the model.

```
set serveroutput on
DECLARE
 v_area_under_curve NUMBER;
BEGIN
 DBMS_DATA_MINING.COMPUTE_ROC (
 roc_area_under_curve => v_area_under_curve,
 apply_result_table_name => 'BANKDATA_nb_test_results',
 target_table_name => 'TEST_BD',
 case_id_column_name => 'CUST_ID',
 target_column_name => 'Y',
 roc_table_name => 'BANKDATA_NB_ROC',
 positive_target_value => 'yes',
 score_column_name => 'PREDICTED_VALUE',
 score_criterion_column_name => 'PROBABILITY');
 DBMS_OUTPUT.PUT_LINE('**** AREA UNDER ROC CURVE ****: ' ||
 ROUND(v_area_under_curve,4));
END;

SELECT * FROM BANKDATA_NB_ROC;
```

OUTPUT:

Script Output × | Query Result ×

Task completed in 0.686 seconds

```
**** AREA UNDER ROC CURVE ****: .6799


PL/SQL procedure successfully completed.
```

Script Output × | Query Result ×

SQL | Fetched 50 rows in 0.015 seconds

	PROBABILITY	TRUE_POSITIVES	FALSE_NEGATIVES	FALSE_POSITIVES	TRUE_NEGATIVES	TRUE_POSITIVE_FRACTION	FALSE_POSITIVE_FRACTION
1	1.0	58	793	15	1140	0.068155111633372502...	0.0129870129870129870...
2	0.9999999403953552	110	741	26	1129	0.129259694477085781...	0.0225108225108225108...
3	0.9999998807907104	131	720	29	1126	0.153936545240893066...	0.0251082251082251082...
4	0.9999998211860657	150	701	32	1123	0.176263219741480611...	0.0277056277056277056...
5	0.9999997615814209	164	687	38	1117	0.192714453584018801...	0.0329004329004329004...
6	0.9999997019767761	170	681	42	1113	0.199764982373678025...	0.0363636363636363636...

- **CONCLUSION**

In previous section, three data ming models have been created and tested using test data. The accuracy of each of the models are as tabulated below:

MODEL NAME	ACCURACY (%)
Decision tree	**91.62**
Support Vector Machine	**85.82**
Naive Bayes	**86.18**

From the table above, it is clear that Decision tree is evaluated as the best data mining model based on model's accuracy, to predict the customers who might subscribe for the term deposits in Portuguese Bank.

PART D

In this part of project, a confusion matrix of the best data model evaluated is formed by using PL/SQL programming.

- **CONFUSION MATRIX**

Confusion matrix is formed from the four results produced by evaluating the classification model using a test data. Here we are considering Decision tree evaluation results as it is evaluated as the best model for predicting the target variable of Portuguese bank dataset.

A binary classifier model predicts the target variable as either positive('yes' in this case) or negative('no' in this case) . The prediction produces four results as specified above and are explained below:

- **TRUE POSITIVE (TP)** : When actual value is 'yes' and predicted value is also 'yes'.
- **TRUE NEGATIVE (TN)** : When actual value is 'no' and predicted value is also 'no'.
- **FALSE POSITIVE (FP)** : When actual value is 'no' and predicted value is 'yes'.
- **FASLE NEGATIVE (FN)** : When actual value is 'yes' and predicted value is 'no'.

Using the above four results, we can also derive various basic measures of the data mining model which are useful in assessing the same. Some of the measures are explained below:

- **Misclassification rate or Error rate:**
  Misclassification rate is the measure of how often the model's prediction is incorrect. It is calculated as number of incorrect predictions by total number of predictions. The misclassification rate can range between best of 0.0 to worst of 1.0.

  Misclassification rate = (FP+FN)/(TP+TN+FP+FN)

- **Accuracy:**
  Accuracy of a model is the measure of how often the model's prediction is correct. It is calculated as number of correct predictions by total number of predictions.

Accuracy can range between best accuarcy as 1.0 and worst as 0.0.

Accuracy = (TP+TN)/(TP+TN+FP+FN)

- **Sensitivity or Recall or True Positive rate:**
  Sensitivity is the measure of how often the prediction is correct when the actual value is 'yes'. It is calculated as number of correct positive predictions by total number of positives. Sensitivity can range between 1.0 as best and 0.0 as worst.

  Sensitivity = (TP)/(TP+FN)

- **Specificity or True Negative rate:**
  Specificity is the measure of how often the prediction of model is correct when the actual value is 'no'. It is calculated as number of correct negative predictions by total number of negatives. Specificity can range between best as 1.0 and worst as 0.0.

  Specificity = (TP)/(TN+FP)

- **Precision:**
  Precision is measure of how precise the model is when predicting positive instances. It is calculated as number of correct positive predictions by total number of positive predictions. Precision value ranges from worst of 0.0 to best of 1.0.

  Precision = (TP)/(TP+FP)

- **False positive rate:**
  False positive rate is the measure of how often the prediction of model is incorrect when the actual value is 'no'. It is calculated as number of incorrect positive predictions by total number of negatives. False positive rate ranges from worst of 1.0 to best of 0.0.

  False positive rate = (FP)/(TN+FP)

• CONFUSION MATRIX USING PL/SQL PROGRAM

Below is the PL/SQL code used for generating confusion matrix and some useful measures derived from confusion matrix, to assess the model.

```
DECLARE
  TRUE_POSITIVE NUMBER;
  TRUE_NEGATIVE NUMBER;
  FALSE_POSITIVE NUMBER;
  FALSE_NEGATIVE NUMBER;
  TOTAL_RECORDS NUMBER;
  MISCLASSIFICATION_RATE NUMBER;
  ACCURACY NUMBER;
  SENSITIVITY NUMBER;
  SPECIFICITY NUMBER;
  PRECISION NUMBER;
  FALSE_POSITIVE_RATE NUMBER;
BEGIN
  SELECT VALUE INTO TRUE_POSITIVE FROM
BANKDATA_DT_confusion_matrix1
  WHERE ACTUAL_TARGET_VALUE ='yes' AND
PREDICTED_TARGET_VALUE='yes';
  SELECT VALUE INTO TRUE_NEGATIVE FROM
BANKDATA_DT_confusion_matrix1
  WHERE ACTUAL_TARGET_VALUE ='no' AND PREDICTED_TARGET_VALUE='no';
  SELECT VALUE INTO FALSE_POSITIVE FROM
BANKDATA_DT_confusion_matrix1
  WHERE ACTUAL_TARGET_VALUE ='no' AND
PREDICTED_TARGET_VALUE='yes';
  SELECT VALUE INTO FALSE_NEGATIVE FROM
BANKDATA_DT_confusion_matrix1
  WHERE ACTUAL_TARGET_VALUE ='yes' AND
PREDICTED_TARGET_VALUE='no';
  MISCLASSIFICATION_RATE :=
ROUND((FALSE_POSITIVE+FALSE_NEGATIVE)/(TRUE_POSITIVE+TRUE_NEGATIVE
+FALSE_POSITIVE+FALSE_NEGATIVE),2);
  ACCURACY :=
ROUND((TRUE_POSITIVE+TRUE_NEGATIVE)/(TRUE_POSITIVE+TRUE_NEGATIVE+F
ALSE_POSITIVE+FALSE_NEGATIVE),2)*100;
  SENSITIVITY :=
ROUND((TRUE_POSITIVE)/(FALSE_NEGATIVE+TRUE_POSITIVE),2);
  SPECIFICITY :=
ROUND((TRUE_NEGATIVE)/(TRUE_NEGATIVE+FALSE_POSITIVE),2);
  PRECISION :=
ROUND((TRUE_POSITIVE)/(TRUE_POSITIVE+FALSE_POSITIVE),2);
  FALSE_POSITIVE_RATE :=
ROUND((FALSE_POSITIVE)/(TRUE_NEGATIVE+FALSE_POSITIVE),2);
  SELECT COUNT(*) INTO total_records FROM TEST_BD;
  dbms_output.put_line(' |----------------------------------------
---------------------------------------------------------|');
```

```
dbms_output.put_line(' |---------------------------------
CONFUSION MATRIX----------------------------------------------|');
  dbms_output.put_line(' |
|');
  dbms_output.put_line(' | TABLE CONTAINS: '|| TOTAL_RECORDS ||'
records
|');
  dbms_output.put_line(' |
|');
  dbms_output.put_line(' |
PREDICTED                                      |');
  dbms_output.put_line(' |                           ------------
----------|----------------------               |');
  dbms_output.put_line(' |                           |       YES
|        NO           |                       |');
  dbms_output.put_line(' |                  |---------|-----------
----------|--------------------|                |');
  dbms_output.put_line(' |                  |   YES   |
'||TRUE_POSITIVE||' (TP)     |        '||FALSE_NEGATIVE||' (FN)
|                       |');
  dbms_output.put_line(' |  Actual          |---------|-----------
----------|--------------------|                |');
  dbms_output.put_line(' |                  |   NO    |
'||FALSE_POSITIVE||' (FP)     |        '||TRUE_NEGATIVE||' (TN)
|                       |');
  dbms_output.put_line(' |                  |---------|-----------
----------|--------------------|                |');
  dbms_output.put_line(' |
|');
  dbms_output.put_line(' |---------------------------------------
--------------------------------------------------------|');
  dbms_output.put_line(' |---------------------------------------
--------------------------------------------------------|');
  dbms_output.put_line('
');

  dbms_output.put_line(' |---------------------------------------
------------------------------------------------------------------
---------------------|');
  dbms_output.put_line(' |-----------------------------------SOME
BASIC MEASURES CALCULATED FROM CONFUSION MATRIX-------------------
--------------------|');
  dbms_output.put_line(' |
|');
```

```
dbms_output.put_line(' |                            |-------------
------------------------------------------------|
|');
  dbms_output.put_line(' |                            |
MISCLASSIFICATE RATE      ->       '|| MISCLASSIFICATION_RATE ||'
|                                    |');
  dbms_output.put_line(' |                            |-----------
--------------------------------------------------|
|');
  dbms_output.put_line(' |                            |
ACCURACY                  ->        '|| ACCURACY||'  %
|                                    |');
  dbms_output.put_line(' |                            |-----------
--------------------------------------------------|
|');
  dbms_output.put_line(' |                            |
SENSITIVITY               ->       '|| SENSITIVITY||'
|                                    |');
  dbms_output.put_line(' |                            |-----------
--------------------------------------------------|
|');
  dbms_output.put_line(' |                            |
SPECIFICITY               ->       '|| SPECIFICITY||'
|                                    |');
  dbms_output.put_line(' |                            |-----------
--------------------------------------------------|
|');
  dbms_output.put_line(' |                            |
PRECISION                 ->       '||PRECISION||'
|                                    |');
  dbms_output.put_line(' |                            |-----------
--------------------------------------------------|
|');
  dbms_output.put_line(' |                            |
FALSE POSITIVE RATE       ->       '|| FALSE_POSITIVE_RATE ||'
|                                    |');
  dbms_output.put_line(' |                            |-----------
--------------------------------------------------|
|');
dbms_output.put_line(' |
|');
  dbms_output.put_line(' |---------------------------------------
-----------------------------------------------------------------
---------------------|');
  END;
```

OUTPUT:

```
Redwood ×
|----------------------------------------------------------------------------------------------------|
|----------------------------------CONFUSION MATRIX--------------------------------------------------|
|                                                                                                    |
| TABLE CONTAINS: 12363 records                                                                      |
|                                                                                                    |
|                                                  PREDICTED                                         |
|                             ----------------------|----------------------                          |
|                             |        YES          |         NO           |                         |
|                   |---------|---------------------|----------------------|                         |
|                   |   YES   |      873  (TP)      |       531  (FN)      |                         |
|  Actual          |---------|---------------------|----------------------|                         |
|                   |   NO    |      505  (FP)      |     10454  (TN)      |                         |
|                   |---------|---------------------|----------------------|                         |
|                                                                                                    |
|----------------------------------------------------------------------------------------------------|
|----------------------------------------------------------------------------------------------------|

|------------------------------------------------------------------------------------------------------------------------------------|
|------------------------------------SOME BASIC MEASURES CALCULATED FROM CONFUSION MATRIX--------------------------------------------|
|                                                                                                                                    |
|                             |--------------------------------------------------------------|                                     |
|                             |       MISCLASSIFICATE RATE      ->        .08                 |                                     |
|                             |--------------------------------------------------------------|                                     |
|                             |       ACCURACY                  ->         92  %              |                                     |
|                             |--------------------------------------------------------------|                                     |
|                             |       SENSITIVITY               ->        .62                 |                                     |
|                             |--------------------------------------------------------------|                                     |
|                             |       SPECIFICITY               ->        .95                 |                                     |
|                             |--------------------------------------------------------------|                                     |
|                             |       PRECISION                 ->        .63                 |                                     |
|                             |--------------------------------------------------------------|                                     |
|                             |       FALSE POSITIVE RATE       ->        .05                 |                                     |
|                             |--------------------------------------------------------------|                                     |
|                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------|
```

From the above output we can see that the accuracy of decision tree model which is built for predicting customers, is **92 %** which is almost same as that of the accuracy calculated in **PART C** of the project **91.62 %.**