Data mining

Regression

Data scientist

Big data

Deep Learning – neuronal networks

Data Scientist Team – various – roles: Technical skills, the communication skills or presentation skills = story telling skills

In terms of platforms, let's you want to be in the traditional predictive analytics environment, and you're not working with big data, then R or Stata, or Python would be your tools. If you're working mostly with unstructured data, then Python is most suitable than R. If you're working with big data, then Hadoop and Spark are the environments that you will be working with.

Data science work (importing data, cleaning data, analyzing data, visualizing data, machine learning)

If the question is to determine probabilities of an action, then a predictive model might be used. If the question is to show relationships, a descriptive approach maybe be required. This would be one that would look at clusters of similar activities based on events and preferences. Statistical analysis applies to problems that require counts. For example if the question requires a yes/ no answer, then a classification approach to predicting a response would be suitable. Machine Learning is a field of study that gives computers the ability to learn without being explicitly programmed. Machine Learning can be used to identify relationships and trends in data that might otherwise not be accessible or identified. In the case where the question is to learn about human behaviour, then an appropriate response would be to use Clustering Association approaches.

In order to **understand the data**, descriptive statistics needed to be run against the data columns that would become variables in the model. First, these statistics included Hearst, univariates, and statistics on each variable, such as mean, median, minimum, maximum, and standard deviation. Second, pairwise correlations were used, to see how closely certain variables were related, and which ones, if any, were very highly correlated, meaning that they would be essentially redundant, thus making only one relevant for modeling. Third, histograms of the variables were examined to understand their distributions. Histograms are a good way to understand how values or a variable are distributed, and which sorts of data preparation may be needed to make the variable more useful in a model. For example, for a categorical variable that has too many distinct values to be informative in a model, the histogram would help them decide how to consolidate those values. The univariates, statistics, and histograms are also used to assess data quality. From the information provided, certain values can be re-coded or perhaps even dropped if necessary, such as when a certain variable has many missing values. The question then becomes, does "missing" mean anything? Sometimes a missing value might mean "no", or "0" (zero), or at other times it simply means "we don't know". Or, if a variable contains invalid or misleading values, such as a numeric variable called "age" that contains 0 to 100 and also 999, where that "triple-9" actually means "missing", but would be treated as a valid value unless we corrected it.

When a true, non-readmission is misclassified, and action is taken to reduce that patient's risk, the cost of that error is the wasted intervention. A statistician calls this a type I error, or a **false-positive**. But when a true readmission is misclassified, and no action is taken to reduce that risk, then the cost of that error is the readmission and all its attended costs, plus the trauma to the patient. This is a type II error, or a **false-negative**.

This time 68% accuracy was obtained on only yes, called **sensitivity** by statisticians, and 85% accuracy on the no, called **specificity**.

How do we determine which model was optimal? As you can see on this slide, the optimal model is the one giving the maximum separation between the blue **ROC curve** relative to the red base line. Today it is commonly used in machine learning and data mining. The ROC curve is a useful diagnostic tool in determining the optimal classification model. This curve quantifies how well a binary classification model performs, declassifying the yes and no outcomes when some discrimination criterion is varied. In this case, the criterion is a relative misclassification cost. By plotting the true-positive rate against the false-positive rate for different values of the relative misclassification cost, the ROC curve helped in selecting the optimal model.

The value of the model will be dependent on successfully incorporating **feedback** and making adjustments for as long as the solution is required. Throughout the Data Science Methodology, each step sets the stage for the next. Making the methodology cyclical, ensures refinement at each stage in the game.

The plan for the feedback stage included these steps: **First**, the review process would be defined and put into place, with overall responsibility for measuring the results of a "flying to risk" model of the congestive heart failure risk population. Clinical management executives would have overall responsibility for the review process. **Second**, congestive heart failure patients receiving intervention would be tracked and their re-admission outcomes recorded. **Third**, the intervention would then be measured to determine how effective it was in reducing re-admissions. For ethical reasons, congestive heart failure patients would not be split into controlled and treatment groups. Instead, readmission rates would be compared before and after the implementation of the model to measure its impact. *After the deployment and feedback stages, the impact of the intervention program on re-admission rates would be reviewed after the first year of its implementation*. Then the model would be refined, based on all of the data compiled after model implementation and the knowledge gained throughout these stages. **Other refinements included:** Incorporating information about participation in the intervention program, and possibly refining the model to incorporate detailed pharmaceutical data. If you recall, data collection was initially deferred because the pharmaceutical data was not readily available at the time. But after feedback and practical experience with the model, it might be determined that adding that data could be worth the investment of effort and time. We also have to allow for the possibility that other refinements might present themselves during the feedback stage. Also, the intervention actions and processes would be reviewed and very likely refined as well, based on the experience and knowledge gained through initial deployment and feedback. Finally, the refined model and intervention actions would be redeployed, with the feedback process continued throughout the life of the Intervention program.

What is SQL? SQL is a language used for relational databases to query or get data out of a database. And is short for its original name Structured English Query Language. So SQL is a language used for a database to query data.

When data is stored in tabular form, the data is organized in tables like in a spreadsheet, which is columns and rows. That's a **relational database**. The columns contain properties about the item such as last name, first name, email address, city. A table is a collection of related things like a list of employees or a list of book authors. In a relational database, you can form relationships between tables. So a database is a repository of data. A set of software tools for the data in the database is called a **database management system or DBMS** for short. The terms database, database server, database system, data server and database management systems are often used interchangeably. **For relational databases, it's called a relational database management system or RDBMS**. RDBMS is a set of software tools that controls the data such as access, organization and storage. And RDBMS serves as the backbone of applications in many industries including banking, transportation, health and so on. *Examples of relational database management systems are, my SQL, Oracle Database, DB2 Warehouse on Cloud and DB2 Express C*.

For the majority of people using a database, there are five simple commands:

- to CREATE a table,
- INSERT data to populate the table,
- SELECT data from the table,
- UPDATE data in the table,
- DELETE data from the table.

Data definition language or DDL statements are used to define, change, or drop data.

Data manipulation language or DML statements are used to read and modify data.

the author_id attribute is assigned as the primary key, so that **no duplicate values can exist. The **primary key** of a relational table uniquely identifies each tuple or row in a table*.

Char is a character string of a fixed length. **Varchar** is a character string of a variable length.

A **CREATE** table statement is an example of **DDL** because it is used for defining structure of the database objects.

The SELECT statement is a Data Manipulation Language statements or DML. The **SELECT** statement is called a query, and the output we get from executing this query is called a result set or a result table.

The **WHERE** clause always requires a predicate. A predicate is conditioned evaluates to true, false or unknown.

Operators supported by a relational database management system:

equal to	=
greater than	>
less than	<
greater than or equal to	≥
less than or equal to	≤
not equal to	≠
Between # and #	≥ y ≤

Expressions that are used with select statements:

- **COUNT** is a built-in database function that retrieves the number of rows that match the query criteria. For example, get the total number of rows in a given table, select COUNT(*) from tablename. Let's say you create a table called MEDALS which has a column called COUNTRY, and you want to retrieve the number of rows where the medal recipient is from Canada. You can issue a query like this. select COUNT(COUNTRY) from MEDALS where COUNTRY where COUNTRY='CANADA'.
- **DISTINCT** is used to remove duplicate values from a result set. Example, to retrieve unique values in a column, select DISTINCT columnname from tablename. In the MEDALS table mentioned earlier, a country may have received a gold medal multiple times. Example, retrieve the list of unique countries that received gold medals. That is, removing all duplicate values of the same country. select DISTINCT COUNTRY from MEDALS where MEDALTYPE = 'GOLD'.
- **LIMIT** is used for restricting the number of rows retrieved from the database. Example, retrieve just the first 10 rows in a table. select * from tablename LIMIT 10. This can be very useful to examine the results set by looking at just a few rows instead of retrieving the entire result set which may be very large. Example, retrieve just a few rows in the MEDALS table for a particular year. select * from MEDALS where YEAR = 2018 LIMIT 5.

To insert data into a table, we use the **INSERT** statement. The INSERT statement is used to add new rows to a table.

Example:

INSERT INTO AUTHOR

(AUTHOR_ID,LASTNAME,FIRSTNAME,EMAIL,CITY,COUNTRY)

VALUES

(‘A1’,’CHONG’,’RAUL’,’RFC@IBM.COM’,TORONTO’,’CANADA’,’CA’),

(‘A2’,’AHUJA’,’RAV’,’RA@IBM.COM’,TORONTO’,’CANADA’,’CA’)

After a table is created and populated with data, the data in a table can be altered with the **UPDATE** statement.

The syntax of the UPDATE statement looks like this

UPDATE [TableName] SET [ColumnName] = [Value]

WHERE [Condition]

In the statement,

TableName identifies the table.

The ColumnName identifies the column value to be changed, as specified in the <WHERE [Condition] >.

Example, you want to update the FIRSTNAME and LASTNAME of the author with AUTHOR_ID A2 from Rav Ahuja to Lakshmi Katta.

To change the first name and last name to Lakshmi Katta where the AUTHOR_ID = A2, enter the UPDATE statement as follows:

UPDATE AUTHOR

SET LAST NAME = KATTA

FIRST NAME = LAKSHMI

WHERE AUTHOR_ID = A2.

Note that if you do not specify the WHERE clause, all the rows in the table will be updated. In this example, without specifying the WHERE clause all rows in the table would have changed the first and last names to Lakshmi Katta.

The rows are removed with the **DELETE** statement.

The syntax of the DELETE statement looks like this

DELETE FROM [TABLEName] <WHERE [Condition] >.

The rows to be removed are specified in the WHERE condition.

Example, we want to delete the rows for AUTHOR_ID A2 and A3.

DELETE FROM AUTHOR

WHERE AUTHOR_ID IN ('A2','A3')

**Note that if you do not specify the WHERE clause, all the rows in the table will be removed.*

Retrieving Rows – using a Range:

a) Db2 => select title, pages from book

where pages >= 290 AND pages <= 300

b) Db2 => select title, pages from book

where pages BETWEEN 290 AND 300

Retrieving Rows – using a Set of Values:

a) Db2 => select firstname, lastname, country from author

where country=’AU’ OR country=’BR’

b) Db2 => select firstname, lastname, country from author

where country IN (’AU’,’BR’)

Sorting Result Sets – ORDER BY clause:

a) Db2 => select title from book

order by title

**ascending order by default*

b) Db2 => select title from book

order by title **desc**

***Desc**ending order with keyword*

Specifying Column Sequence Number:

Db2 => select title, pages from book

order by 2

Eliminating Duplicates - DISTINCT clause:

select country from author order by 1

select distinct(country) from author

GROUPS BY clause:

select country from author order by 1

select country, count(country) from author group by country

select country, count(country) as count from author group by country

Restricting the Result Set - HAVING clause:

select country, count(country) as count from author group by country

select country, count(country) as count from author group by country having count(country) > 4

Built-in Database Functions

PETSALE TABLE

ID	ANIMAL	QUANTITY	SALEPRICE	SALESDATE
INTEGER	VARCHAR(20)	INTEGER	DECIMAL(6,2)	DATE
1	Dog	9	450.09	29/05/2019

Aggregate or Column Functions

INPUT: Collection of values (e.g. entire column)

Output: Single value

Examples: SUM(), MIN(), MAX(), AVG(), etc.

SUM: Add up all the values in a column

SUM (COLUMN_NAME)

Example 1: Add all values in the SALEPRICE column:

Select SUM(SALEPRICE) from PETSALE

Example 2: Explicitly name the output column SUM_OF_SALEPRICE:

```
Select SUM(SALEPRICE) as SUM_OF_SALEPRICE
    from PETSALE
```

MIN/MAX

Example 3A. Get the maximum QUANTITY of any ANIMAL:

```
Select MAX(QUANTITY) from PETSALE
```

Example 3B. Get the minimum value of ID column for Dogs:

```
Select MIN(ID) from PETSALE where ANIMAL='Dog'
```

AVG

Example 4. Specify the Average value of SALEPRICE:

```
Select AVG(SALEPRICE) from PETSALE
```

#Mathematical operations can be performed between columns.

Example 5. Caulculate the average SALEPRICE per 'Dog':

```
select AVG(SALEPRICE/QUANTITY) from PETSALE
    where ANIMAL='Dog'
```

SCALAR: Perform operations on every input value.

Examples: ROUND (redondear), LENGTH, UCASE (mayuscula), LCASE (minuscula).

Example 6: Round UP or DOWN every value in SALEPRICE:

```
Select ROUND(SALEPRICE) from PETSALE
```

Example 7: Retrieve the length of each value in ANIMAL (obtener la langitud de cada valor en ANIMAL):

```
Select LENGTH(ANIMAL) from PETSALE
```

UCASE, LCASE

Example 8: Retrieve ANIMAL values in UPPERCASE:

```
Select UCASE(ANIMAL) from PETSALE
```

Example 9: Use the function in a WHERE clause:

```
select * from PETSALE
where LCASE(ANIMAL)='cat'
```

Example 10: Use the DISTINCT() function to get unique values:

```
Select DISTINCT(UCASE(ANIMAL)) from PETSALE
```

Date, Time Functions

DATE: YYYYMMDD

TIME: HHMMSS (horas, minutos y segundos)

TIMESTAMP: YYYYXXDDHHMMSSZZZZZZ (where double X represents the month and six Z or Z represents microseconds)

Date/Time functions:

YEAR(), MONTH(), DAY(), DAYOFMONTH(), DAYOFWEEK(), DAYOFYEAR(), WEEK(), HOUR(), MINUTE(), SECOND()

Example 11: Extract the DAY portion from a date:

```
Select DAY(SALEDATE) from PETSALE
        where ANIMAL='Cat'
```

Example 12: Get the number of sales during the month of May:

```
Select COUNT(*) from PETSALE
        where MONTH(SALEDATE)='5'
```

Date or Time Arithmetic

Example 13: What date is it 3 days after each sale date?

```
Select (SALEDATE+3 DAYS) from PETSALE
```

Special Registers: CURRENT_DATE, CURRENT_TIME

Example 14: Find how many days have passed since each SALEDATE till now:

```
Select (CURRENT_DATE – SALESDATE) from PPETSALE
```

Sample result: 10921 (YMMDD)

Sub-query: A query inside another query

```
Select COLUMN1 from TABLE
        Where COLUMN2=(select MAX(COLUMN2) from TABLE)
```

To retrieve the list of employees who earn more than the average salary:

Sub-queries to evaluate Aggregate functions

```
Select EMP_ID, F_NAME, L_NAME, SALARY
        From employees
        Where SALARY <
        (select AVG(SALARY) from employees);
```

Sub-queries in list of columns

```
Select EMP_ID, SALARY,
        (select AVG(SALARY) from employees)
        As AVG_SALARY
        From employees;
```

Sub-queries in FROM clause

Sub-queries like these are sometimes called **derived tables or **table expressions**. Because the outer query uses the results of the sub-query as a data source.*

```
Select * from
        (select EMP_ID, F_NAME, L_NAME, DEP_ID
                from employees) AS EMP4ALL;
```

Accessing Multiple Tables with sub-queries

To retrieve only the employee records that correspond to departments in the DEPARTMENTS table:

```
Select * from employees
        Where DEP_ID_IN
        (select DEPT_ID_DEP from DEPARTMENTS);
```

To retrieve only the list of employees from a specific location:

```
Select * from employees
        Where DEP_ID_IN
        (select DEPT_ID_DEP from departments
                Where LOC_ID='L0002');
```

To retrieve the department ID and name for employee who earn more than $70,000:

```
Select DEPT_ID_DEP, DEP_NAME from departments
        Where DEPT_ID_DEP IN
                (select DEP_id from employees
                        Where SALARY > 70000);
```

Accessing multiple tables with Implicit Joint

Specify 2 tables in the FROM clause:

```
Select * from employees, departments;
```

**The result is a full joint (or Cartesian join): every row in the table is jointed with every row in the second table. The result set will have more rows than in both tables.*

Use additional operands to limit the result set:

```
Select * from employees, departments
        Where employees.DEP_ID=
                Departments.DEPT_ID_DEP;
```

Use shorter aliases for table names:

```
Select * from employees E, departments D
        Where E.DEP_ID = D.DEPT_ID_DEP;
```

To see the department name for each employee:

```
Select employees.EMP_ID, departments.DEPT_NAME
        From employees E, departments D
        Where E.DEP_ID = D.DEPT_ID_DEP;
```

Column names in the select clause can be pre-fixed by aliases:

```
Select E.EMP_ID, D.DEP_ID from
        Employees E, Departments D
        Where E.DEP_ID = D.DEPT_ID_DEP
```

A **Primary Key** of a relational table uniquely identifies each row in a table. Notice when some attributes have FK in brackets next to them. This identifies the **Foreign Key**. A foreign key is a set of columns referring to a primary key of another table. These entities are part of the relationship set between the entities above them. All of the entities have a one to many relationship established between them. A table containing a primary key that is related to at least one foreign key is called a **Parent Table**. A table containing one or more foreign keys is called a **Dependent Table** or **Child Table**.

Relational Model Constraints

1. **Entity Integrity Constraint** = Primary Key
2. **Referential Integrity Constraint.** Defines relationships between tables and ensures that these relationships remain valid. The validity of the data is enforced using a combination of primary keys and foreign keys. i.e. for a book to exist, it has to be written by at least one author.
3. **Semantic Integrity Constraint.** Refers to the correctness of the meaning of the data. For example, in the relation author, if the attribute or column city contains a garbage value instead of Toronto, the garbage value does not have any meaning.
4. **Domain Constraint.** Specifies the permissible values for a given attribute. For example, in the relation author, the attribute country must contain a two letter country code such as CA for Canada or IN for India. If a number value of 34 is entered for the country attribute instead of a two let her country code, the value 34 does not have any meaning.
5. **Null Constraint.** Specifies that attribute values cannot be null.
6. **Check Constraint.** enforces domain integrity by limiting the values that are accepted by an attribute.

PRIMARY KEYS

If a relation schema has more than one key, then each key is called a candidate key. One of the candidate keys is designated as the primary key, and the others are called secondary keys.

In a practical relational database, each relation schema must have a primary key.

Rules for primary keys:

- The value of the Primary Key must be unique for each instance of the entity.
- There can be no missing values (i.e. Not Null) for Primary Keys. If the Primary Key is composed of multiple attributes, each of those attributes must have a value for each instance.
- The Primary Key is immutable, that is, once created the value of the Primary Key cannot be changed.
- If the Primary Key consists of multiple attributes, none of these values can be updated.

SEMANTIC INTEGRITY

Semantic integrity ensures that data entered into a row reflects an allowable value for that row. The value must be within the domain, or allowable set of values, for that column. For example, the quantity column of the items table permits only numbers. If a value outside the domain can be entered into a column, the semantic integrity of the data is violated.

SEMANTIC CONSTRAINTS

Semantic Constraints are constraints that cannot be directly expressed in the schemas of the data model. Semantic constraints are also called application-based rules or business rules. They are additional rules specified by users or database administrators. For example, a class can have a maximum of 30 students; salary of an employee cannot exceed the salary of the employee's manager.

Domain constraints specify that within a tuple the value of each attribute must be an element from the domain of that attribute. The data types associated with the domains include:

- Integers (short integer, integer, long integer)
- Real numbers (float and double precision float)
- Characters
- Booleans
- Fixed-length strings and variable length strings
- Date, time, timestamp
- Money
- Other special data types
- Other possible domain values may be a sub-range of values from a data type or as an enumerated data type in which values are explicitly listed.