Penn State University
Great Valley Campus

Engineering Division

Data Specification for
NY Property Sales Systems
Version 6.0

27th February 2022

Table of Contents

NY Property Sales Systems
Rudraksh Mishra

INTRODUCTION

New York city being one of the most populated cities in the world has one of the highest real estate sales prices. It provides its annual property sale records to the public every year. Using this data, we will be designing warehouse architecture and also implement Hadoop, to list and compare the pro and cons using individual systems.

PURPOSE

The purpose of this project is to assist the management of the NY real estate department in making better decision using the historical data available within the state by comparing the best approach between data warehouse implementation and Hadoop implementation. The government hired real estate agents might lack the ability to access consistent historical data easily when needed. The problem of data inconsistence can be eliminated by a central data warehouse. For the purpose of data harmonization and consistency this project of data warehouse is built.

PROJECT SUMMARY

Comparison of data warehouse and Hadoop systems on NY property sales data, followed by data visualization to gain insights and answer business questions.

A. Objectives

The main objective of this project is to design, develop and compare between a data warehouse and Hadoop implementation and to find trends in real estate industry using reports based on big data solutions.

B. Scope

The scope of this project is to make consistent data warehouse and Hadoop implementation, business intelligence reports/ tools and validate the systems by using the case study/ data.

C. References

- Data was extracted, cleaned, and separated by course instructor from https://www1.nyc.gov/site/finance/taxes/property-annualized-sales-update.page

D. Outstanding Issues

- For the NY property sales dataset that was extracted, all boroughs did not have data for the years in continuous manner in a particular time frame.
- Hence property sales for some boroughs over few years were missing.

REQUIREMENTS DEFINITION

- **Goals**
  - Extract, transform, and load the NY real estate data
  - Validate the data is consistent and harmonized
  - Make business focused reports.
  - Chose best architecture based on the data being analyzed.
  - List advantages and disadvantages of each system
- **Usability Requirements**
  - User with knowledge of Knime for reporting and data analysis
  - User with knowledge of PostgreSQL for querying
  - Administrative login of authorized users
  - Other requirements will be updated after implementation of architecture
- **System Security Requirements**
  - Only authenticated users can log in to the Postgre database warehouse and Hadoop Hive database using the Hostname, portname, username and password.
  - The data cannot be altered without the permission of the admin.
- **Business Questions**
  - Number of sales over the years for different building categories.
  - To determine the number of commercial units and residential units sold over the years.
  - Number of sales per year according to different binned land areas.
  - To calculate the top 5 neighborhood with the greatest number of sales
  - Number of sales over the years in four boroughs
- **Data Requirements**
  - Data has been preprocessed and provided by the course instructor in the form of excel files that contains New York property sales records for various years.

- **Design Constraints**
  - In data warehousing we are adopting the star schema. One of the main problems in choosing the star schema is that the data is not normalized hence the data can be redundancy. Since there is a risk of redundancy there is a huge risk of data integrity.
  - The data should be altered only with permission of the admin.

DOCUMENT CHANGE LOG

Change Date	Version	CR #	Change Description	Author and Organization
01/23/2022	1.0		Initial creation.	Rudraksh Mishra
02/05/2022	2.0		Warehouse Architecture design	Rudraksh Mishra
02/10/2022	3.0		Data preparation	Rudraksh Mishra
02/13/2022	4.0		Warehouse Reporting System	Rudraksh Mishra
02/17/2022	5.0		Hadoop Architecture design	Rudraksh Mishra
02/21/2022	6.0		Hadoop ETL and reporting	Rudraksh Mishra

2. ARCHITECTURE DESIGN

2.1 Relational Data Warehouse

2.1.1 Determining the Fact Table

In data warehouse design, a fact table is utilized in the dimensional model. A fact table, surrounded by dimension tables, is located in the center of our star schema. A fact table is a list of facts about a specific business process. Metrics or measures are other terms for facts. Since we are building a data warehouse for NY Property Sales we consider the sales price as metric and store the column in fact table. All other columns will be grouped and stored as dimensions and will be accessed by fact table through foreign keys.

2.1.2 Determining Level of Granularity

The granularity refers to the level of detail accessible in a star schema. Granularity is unique to each fact and dimension table. When the fact and dimension tables are connected, the grain of the dimensional model is the finest level of information that is inferred. For our warehouse the level of granularity is property type sold by day in each NY borough.

2.1.3 Attributes considered

Dimensions in a data warehouse are collections of reference information about the facts. To give meaningful, classified, and descriptive responses to business issues, dimensions categorize and describe the facts collected in a data warehouse.
For NY property sales warehouse we will be considering Borough, Neighborhood, Block, Lot, Address, zip code, building class at time of sale, tax class at time of sale, Building class category, residential units, commercial units, land square feet, gross square feet, sale date, Sale price. In the data dictionary below describes all the list of variables considered for developing the data warehouse. Some of the variables like ease-ment, tax class at present, building class at present, apartment number, total units, year built were discarded as they could be either computed with the existing variables or were of low significance for data warehouse.

2.1.4 Indexing

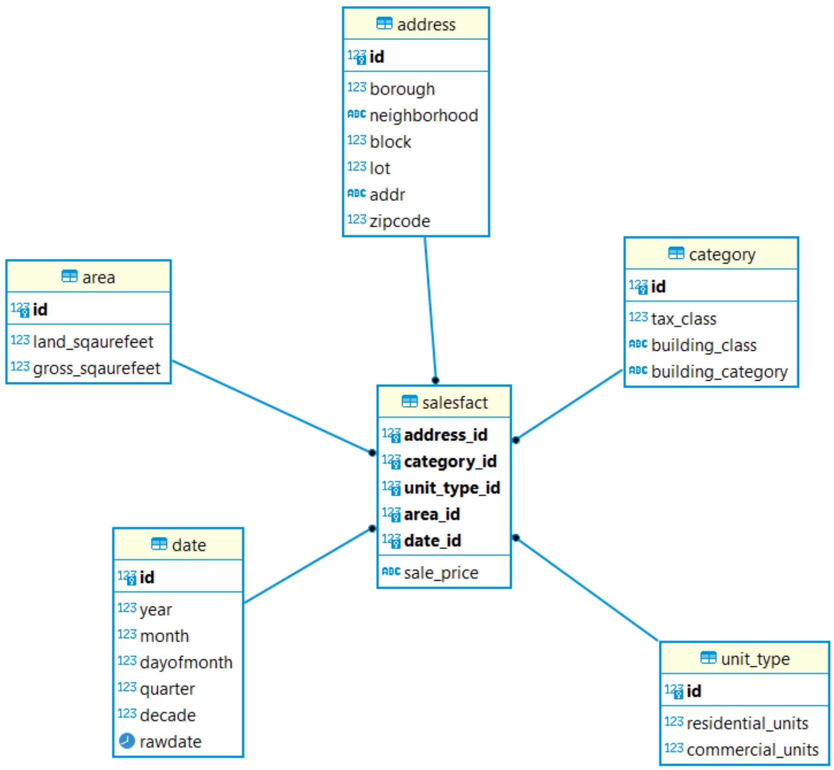
In this warehouse PostGre Database is used. All the rows are indexed serially using the serial4 and hence they increment one after the other.

2.1.5 Data Dictionary

NY Property Sales Data				
Variable	Variable name	Variable type	Values	notes
Borough	Borough	Integer	1,2,3,4,5	Number of the borough
Neighborhood	Neighborhood	varchar	BEDFORD STUYVESANT, BERGEN BEACH,etc	Name of the neighborhood
Block	Block	Integer	8366,8369, etc	Block Number
Lot	Lot	Integer	368,369,etc	Lot number
Address	Address	varchar	7115 BERGEN COURT, 1302 EAST 70STREET,etc	Complete address of the property
zip code	Zipcode	Integer	11234,11217	Zipcode of property
building class at time of sale	Building_category	Varchar	S0,A9,B1,etc.	property's constructive use description
tax class at time of sale	Tax_class	Integer	1,2,3,4	4 different property tax classes
Building class category	Building_class_category	Varchar	02 TWO FAMILY HOMES, 13 CONDOS - ELEVATOR APARTMENTS,etc	Category of the building
residential units	Residential_units	Integer	0,1,2	Number of residential units
commercial units	Commercial_units	Integer	0,1,2	Number of commercial units
land square feet	Land_squarefeet	Integer	4670,2000,etc	Land square feet of property
gross square feet	Gross_squarefeet	Integer	22000,3000,etc	Gross square feet of property
sale date	rawdate	date	3/17/2005, 4/1/2008,etc	Date at which property was sold

Sale price	Sale_price	Integer	136000, 445000,etc	Price at which the property was sold in dollars

2.1.6 Star Schema Reperesentation



2.1.7 Tables schemas

address			
Description	This table describes the address of the property sales		
Attribute	**Description**	Type	Values
Id	Id of addresses	Serial4	Between 1 and 999999999
borough	Number of the borough	Integer	1,2,3,4,5

Neighborhood	Name of the neighborhood	varchar	BEDFORD STUYVESANT, BERGEN BEACH
Block	Block Number	Integer	8366,8369
Lot	Lot number	Integer	368,369,etc
Address	Complete address of property	varchar	7115 BERGEN COURT, 1302 EAST 70STREET
Zipcode	Zipcode of property	Integer	11234,11217
Primary Key	Id		
Foreign Keys			
SQL input	see code below		

```
CREATE TABLE address (
  id serial4 NOT NULL,
  borough int4 NULL,
  neighborhood varchar NULL,
  block int4 NULL,
  lot int4 NULL,
  addr varchar NULL,
  zipcode int4 NULL,
  CONSTRAINT address_pk PRIMARY KEY (id)
);
```

category			
Description	This table describes the category of the property		
Attribute	**Description**	Type	Values
Id	Id for the category type	Serial4	Between 1 and 999999999
Tax_class	4 different property tax classes	Integer	1,2,3,4
Building_class_category	Category of the building	Varchar	02 TWO FAMILY HOMES, 13 CONDOS - ELEVATOR APARTMENTS
Building_category	Property's constructive use description	Varchar	S0,A9,B1,etc.
Primary Key	Id		
Foreign Keys			
SQL input	see code below		

```
CREATE TABLE category (
```

	`id serial4 NOT NULL,`
	`tax_class int4 NULL,`
	`building_class_category varchar NULL,`
	`building_category varchar NULL,`
	`CONSTRAINT category_pk PRIMARY KEY (id));`

unit_type			
Description	This table describes the unit_type and unit count of the property		
Attribute	**Description**	Type	Values
Id	Id for the category type	Serial4	Between 1 and 999999999
Residential_units	Number of residential units	Integer	0,1,2
Commercial_units	Number of commercial units listed	Integer	0,1,2
Primary Key	Id		
Foreign Keys			
SQL input	see below		

```
CREATE TABLE unit_type (
  id serial4 NOT NULL,
  residential_units int4 NULL,
  commercial_units int4 NULL,
  CONSTRAINT unit_type_pk PRIMARY KEY (id)
);
```

area			
Description	This table describes the unit_type of the property		
Attribute	**Description**	Type	Values
Id	Id for the category type	Serial4	Between 1 and 999999999
Land_squarefeet	Land squarefeet of property	Integer	4670,2000,etc
Gross_squarefeet	Gross squarefeet of property	Integer	22000,3000
Primary Key	Id		
Foreign Keys			
SQL input	see below		

```
CREATE TABLE area (
  id serial4 NOT NULL,
  land_sqaurefeet int4 NULL,
  gross_sqaurefeet int4 NULL,
  CONSTRAINT area_pk PRIMARY KEY (id)
);
```

date			
Description	This table describes the sale date of the property		
Attribute	**Description**	Type	Values
Id	Id for the category type	Serial4	Between 1 and 999999999
Year	Year number	Integer	2005,2008
Month	Month number of a year	Integer	1,2,…12
dayofmonth	Day number of a month	Integer	1,2…..31
quarter	Quarter number of a year	Integer	1,2,3,4
rawdate	Original date in mm/dd/yyyy of the property sold	date	3/17/2005, 4/1/2008
Primary Key	Id		
Foreign Keys			
SQL input	See code below		

```
CREATE TABLE "date" (
  id serial4 NOT NULL,
  "year" int4 NULL,
  "month" int4 NULL,
  dayofmonth int4 NULL,
  quarter int4 NULL,
  rawdate date NULL,
  CONSTRAINT date_pk PRIMARY KEY (id)
);
```

salesfact			
Description	This table is the fact table for the property sales dataset. It contains the sales value		
Attribute	**Description**	Type	Values
Address_id	Id of the address from the address table	Serial4	Between 1 and 999999999
Category_id	Id of the category from the category table	Serial4	
Unit_type_id	Id of the units from the unit_type table	Serial4	
Area_id	Id of the area from the area table	Serial4	
Date_id	Id of the date from the date table	Serial4	
Sale_price	Price at which the property was sold in dollars	Integer	136000, 445000
Primary Key	(address_id, category_id, unit_type_id, area_id, date_id)		

Foreign Keys	address_id, category_id, unit_type_id, area_id, date_id

SQL input	CREATE TABLE salesfact (
	  sale_price varchar NULL,
	  address_id int4 NOT NULL DEFAULT
	nextval('fact_address_id_seq'::regclass),
	  category_id int4 NOT NULL DEFAULT
	nextval('fact_category_id_seq'::regclass),
	  unit_type_id int4 NOT NULL DEFAULT
	nextval('fact_unit_type_id_seq'::regclass),
	  area_id int4 NOT NULL DEFAULT
	nextval('fact_area_id_seq'::regclass),
	  date_id int4 NOT NULL DEFAULT
	nextval('fact_date_id_seq'::regclass),
	  CONSTRAINT salesfact_pk PRIMARY KEY (address_id, category_id,
	unit_type_id, area_id, date_id)
	);
	
	ALTER TABLE salesfact ADD CONSTRAINT fact_fk FOREIGN KEY
	(address_id) REFERENCES public.address(id);
	ALTER TABLE salesfact ADD CONSTRAINT fact_fk_1 FOREIGN KEY
	(category_id) REFERENCES category(id);
	ALTER TABLE salesfact ADD CONSTRAINT fact_fk_2 FOREIGN KEY
	(unit_type_id) REFERENCES unit_type(id);
	ALTER TABLE public.salesfact ADD CONSTRAINT fact_fk_3 FOREIGN KEY
	(area_id) REFERENCES public.area(id);
	ALTER TABLE salesfact ADD CONSTRAINT fact_fk_4 FOREIGN KEY
	(date_id) REFERENCES public."date"(id);

2.2 Hadoop Implementation

The same attributes were considered for Hadoop implementation of the NY property sales dataset. Hence, we will be considering Borough, Neighborhood, Block, Lot, Address, zip code, building class at time of sale, tax class at time of sale, Building class category, residential units, commercial units, land square feet, gross square feet, sale date, Sale price. In the data dictionary above (2.1.5) describes all the list of variables considered for developing the Hadoop implementation. Some of the variables like ease-ment, tax class at present, building class at present, apartment number, total units, year built were discarded as they could be either computed with the existing variables or were of low significance for Hadoop implementation. These discarded attributes did not address the current business questions and also were of no significance for the future business questions that might be asked.

Here we make use of Hive to process structured data in Hadoop. Hive is used on top of the Hadoop and has quite similar syntax to that of SQL. Hive changes submitted SQL inquiries into MapReduce jobs and puts them to the clusters. Each time an inquiry is submitted to Hive the Metastore is additionally refreshed.

We will be creating only one table that stores all the attributes as the data stored in Hive in a de-normalized form.

Steps followed to create that table with the above selected attributes are:

- To create a table in Hive, we need to first start create a database. To do so we used:

```
0: jdbc:hive2://localhost:10000> CREATE DATABASE IF NOT EXISTS assignmenthive;
No rows affected (0.595 seconds)
0: jdbc:hive2://localhost:10000> SHOW DATABASES;
+-----------------+
|  database_name  |
+-----------------+
| assignmenthive  |
| default         |
+-----------------+
2 rows selected (0.133 seconds)
0: jdbc:hive2://localhost:10000> USE assignmenthive;
No rows affected (0.05 seconds)
```

- To create the table with our own attributes and their data types we use:

  CREATE TABLE prop_sales(borough INT,neighborhood VARCHAR(100),building_class_category VARCHAR(100),block INT,lot INT,address VARCHAR(100),zipcode INT,residential_units INT,commercial_units INT,land_squarefeet INT,gross_sqaurefeet INT,tax_class INT,building_class VARCHAR(20),sale_price INT,sale_date TIMESTAMP,count VARCHAR(50)) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','STORED AS TEXTFILE;

- To check the created table we can just type DESCRIBE prop_sales;

```
0: jdbc:hive2://localhost:10000> DESCRIBE prop_sales;
+-------------------------+--------------+----------+
|        col_name         |  data_type   | comment  |
+-------------------------+--------------+----------+
| borough                 | int          |          |
| neighborhood            | varchar(100) |          |
| building_class_category | varchar(100) |          |
| block                   | int          |          |
| lot                     | int          |          |
| address                 | varchar(100) |          |
| zipcode                 | int          |          |
| residential_units       | int          |          |
| commercial_units        | int          |          |
| land_squarefeet         | int          |          |
| gross_sqaurefeet        | int          |          |
| tax_class               | int          |          |
| building_class          | varchar(20)  |          |
| sale_price              | int          |          |
| sale_date               | timestamp    |          |
| count                   | varchar(50)  |          |
+-------------------------+--------------+----------+
```

2.2.1 Data Dictionary for Hadoop implementation

NY Property Sales Data				
Variable	Variable name	Variable type	Values	notes
Borough	Borough	Integer	1,2,3,4,5	Number of the borough
Neighborhood	Neighborhood	varchar	BEDFORD STUYVESANT, BERGEN BEACH,etc	Name of the neighborhood
Block	Block	Integer	8366,8369, etc	Block Number
Lot	Lot	Integer	368,369,etc	Lot number
Address	Address	varchar	7115 BERGEN COURT, 1302 EAST 70STREET,etc	Complete address of the property
zip code	Zipcode	Integer	11234,11217	Zipcode of property
building class at time of sale	Building_category	Varchar	S0,A9,B1,etc.	property's constructive

				use description
tax class at time of sale	Tax_class	Integer	1,2,3,4	4 different property tax classes
Building class category	Building_class_category	Varchar	02 TWO FAMILY HOMES, 13 CONDOS - ELEVATOR APARTMENTS,etc	Category of the building
residential units	Residential_units	Integer	0,1,2	Number of residential units
commercial units	Commercial_units	Integer	0,1,2	Number of commercial units
land square feet	Land_squarefeet	Integer	4670,2000,etc	Land square feet of property
gross square feet	Gross_squarefeet	Integer	22000,3000,etc	Gross square feet of property
sale date	rawdate	timestamp	3-17-2005 00:00:00.0	Date at which property was sold
Sale price	Sale_price	Integer	136000, 445000,etc	Price at which the property was sold in dollars
-	count	Varchar	1,2,3,4	Count of each unique line in table

2.3 Reflective analysis of using a data warehouse vs Hadoop.

While developing the architecture design of the data warehouse, the main analysis to be done was which dimensions are to be considered to be kept and which to be discarded. This problem was approached by analyzing which variables will be used in solving the current business problems and also take into account future business problems that might be asked to be solved through the data warehouse. Keeping this in mind, the variables were grouped based on similar characteristics into dimensions and then those dimensions were represented in the fact table, while utilizing the star schema. Another difficulty faced during this process was which variables are to be grouped together in one dimension and on what basis.

Whereas while developing the architecture design of the Hadoop implementation, the only analysis to be done was which attributes are to be discarded based on their current and future use in answering business questions. Since Hadoop implementation did not require building multiple tables and hence no use of primary/foreign keys, the design was much simplified. It only consisted of one table for the entire data stored, as the data stored in Hive in a de-normalized form.

3. Data Preparation

3.1 Relational Data Warehouse Implementation

3.1.1 ETL considerations

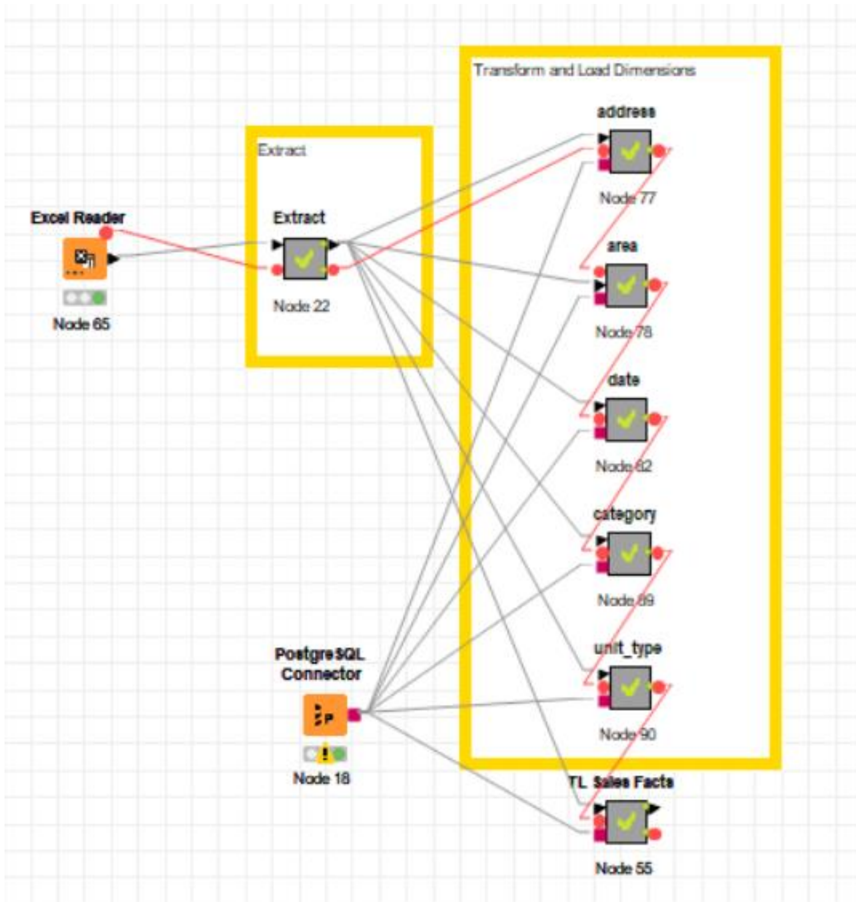
Extract, transform, and load are the three key processes in the ETL process. Let's take a closer look at their consideration:

1) Extract - In layman's terms, this is the process of extracting data from many sources. We're going to use logical extraction. We're going to use online physical extraction. We extract all data from the source system in this step. The fundamental benefit of this extraction is that the rack of modifications is no longer required.

2) Transform: Before loading, data must be transformed into the proper format. Only the necessary data must be selected, and the rest must be discarded. We will join the data in the required scenarios for fact table. Then we convert, which involves shaping the data's format and structure to ensure data compliance with the target system.

3) Load: The process of writing data into the target source is known as loading. It is vital to ensure data freshness while loading the data. Because our data is historical, it will have a high update efficiency. The new-to-historical data ratio should be modest.

3.1.2 ETL Process Flow with description



- First, we use excel reader and load all the 9 files of NY Annualized Property Sales one by one. After all the files are loaded based on same columns, we prune those columns that we chose to discard in the architecture design. This is doing with the help of column filter.

- Using String to Date&Time we change the Date format of the “sale date” variable to yyyy-mm-dd.

- With the help of row filter, we remove the columns with null sale price data. This helps us prune those rows in dataset which do not have any record of the sales price of the property.

- Now we load each dimension one by one. All the dimensions follow the same set of process except the dimension for the sales date.
- The set of procedures followed by address, area, category, unit_type dimension is:
  - We first filter and keep only those set of columns that are required by that particular dimension. This is done using the column filter. For example- for address dimension we require only borough, neighborhood, block, lot, address, zip code.
  - Then we use groupby to group rows which have similar values for all these attributes. For example- if a row had same borough, neighborhood, block, lot, address, zip code- it would be grouped as one row.
  - Then we rename the column names of the dataset to names similar to the variable column names assigned while building the architecture/schema. For example- address column is renamed to addr as "addr" is the name of the column which represents the entire address in our schema design.
  - Next, we make sure we don't load the same data twice. To avoid this, we use reference row filter which avoids the duplicate loading of the rows from reference table I.e excludes the rows from reference table. Here the reference table is loaded from the DB query reader (it loads an empty table initially with the use of SQL statements). Once complete dimension has been loaded then DB query reader outputs the data which was already loaded before. Then the reference row filter filters the repeated or duplicate rows.
  - For date attributes we add another node in between of GroupBy and column rename. This node is the extract date&time fields node, it is used to extract the year, quarter, month(number), dayofmonth in our case.
- This is then passed to the DB writer which loads the data into the warehouse with the help of the postgre sql connector. Postgre Sql connector helps to connect with the database with the correct input port and credentials.

We make sure we connect the flow variable port of the input excel reader to the extract metanode and then connect each dimension's DB writer to the node of the next dimension. By doing so we make sure if there are any changes made to either the input or the intermediatory nodes all the other connected nodes are reset as well.

Now that the dimensions are preprocessed and loaded, it is time to build the fact table. To do so we take the output from the last dimension and join the columns of that with the db query reader. After which we select all the columns except the columns that we matched. For example- we inner join the zip codes from db query reader and the input, and include all other columns except the joined ones (zip code).

Then we group the output by all the dimension IDs using GroupBy node and then rename the columns and load the dataset in the fact table using DB writer. After doing so the complete process of ETL is done.

Here is the statistics of the data loaded. Each screenshot represents a different xls files.

1) 2005_sales_brooklyn_05

Here the total row count loaded into the database is 33,492

Table "default" - Rows: 6 Spec - Columns: 16 Properties Flow Variables

Row ID	S Column	D Min	D Max	D Mean	D Std. de...	D Variance	D Skewness	D Kurtosis	D Overall ...	I No. mis...	I No. NaNs	I No. +∞s	I No. -∞s	D Median	I Row count	Histogram
address_id	address_id	1	31,746	15,808.929	9,146.957	83,666,824....	0.01	-1.193	529,472,634	0	0	0	0	?	33492	1 31.746
area_id	area_id	1	15,254	5,175.654	4,712.983	22,212,208....	0.46	-1.039	173,342,992	0	0	0	0	?	33492	1 15.254
category_id	category_id	1	126	26.237	22.535	507.829	1.649	2.752	878,716	0	0	0	0	?	33492	1 126
date_id	date_id	1	362	178.538	102.445	10,494.955	0.038	-1.146	5,979,592	0	0	0	0	?	33492	1 362
unit_type_id	unit_type_id	1	262	25.645	22.815	520.524	3.822	27.336	858,890	0	0	0	0	?	33492	1 262
sale_price	sale_price	0	240,000,000	439,233.35	1,884,394.293	3,550,941,8...	73.289	8,201.467	14,710,803,...	0	0	0	0	?	33492	0E0 2E8

2) 2008_sales_statenisland

Here the total row count loaded into the database is 2930

Table "default" - Rows: 6 Spec - Columns: 16 Properties Flow Variables

Row ID	S Column	D Min	D Max	D Mean	D Std. de...	D Variance	D Skewness	D Kurtosis	D Overall ...	I No. mis...	I No. NaNs	I No. +∞s	I No. -∞s	D Median	I Row co...	Histogram
address_id	address_id	31,747	38,402	35,079.439	1,881.309	3,539,324.09	-0.028	-1.148	102,782,755	0	0	0	0	?	2930	31.747 38.402
area_id	area_id	1	16,692	10,378.481	7,033.99	49,477,015....	-0.662	-1.4	30,408,949	0	0	0	0	?	2930	1 16.692
category_id	category_id	1	127	18.305	20.356	414.362	2.209	5.647	53,635	0	0	0	0	?	2930	1 127
date_id	date_id	364	696	524.989	94.708	8,969.631	0.036	-1.212	1,538,217	0	0	0	0	?	2930	364 696
unit_type_id	unit_type_id	1	266	21.591	15.516	240.731	10.32	161.41	63,263	0	0	0	0	?	2930	1 266
sale_price	sale_price	0	22,000,000	277,457.217	764,217.945	584,029,06...	18.582	440.626	812,949,646	0	0	0	0	?	2930	0 22.000.000

3) 2009_bronx

Here the total row count loaded into the database is 2529

Table "default" - Rows: 6 Spec - Columns: 16 Properties Flow Variables

Row ID	S Column	D Min	D Max	D Mean	D Std. de...	D Variance	D Skewness	D Kurtosis	D Overall ...	I No. mis...	I No. NaNs	I No. +∞s	I No. -∞s	D Median	I Row co...	Histogram
address_id	address_id	38,403	43,323	40,923.235	1,282.262	1,644,195.445	-0.105	-0.933	103,494,862	0	0	0	0	?	2529	38.403 43.323
area_id	area_id	1	17,227	5,451.502	6,970.504	48,587,926.55	0.745	-1.152	13,786,848	0	0	0	0	?	2529	1 17.227
category_id	category_id	2	126	33.13	21.499	462.216	0.998	1.692	83,786	0	0	0	0	?	2529	2 126
date_id	date_id	697	1,011	862.309	92.175	8,496.297	-0.1	-1.223	2,180,779	0	0	0	0	?	2529	697 1.011
unit_type_id	unit_type_id	1	277	20.252	32.081	1,029.169	4.731	28.534	51,218	0	0	0	0	?	2529	1 277
sale_price	sale_price	0	22,621,380	312,445.371	1,021,914.538	1,044,309,3...	13.015	221.83	790,174,342	0	0	0	0	?	2529	0 22.621.380

4) 2010_queens

Here the total row count loaded into the database is 9039

Table "default" - Rows: 6 Spec - Columns: 16 Properties Flow Variables

Row ID	S Column	D Min	D Max	D Mean	D Std. de...	D Variance	D Skewness	D Kurtosis	D Overall ...	I No. mis...	I No. NaNs	I No. +∞s	I No. -∞s	D Median	I Row co...	Histogram
address_id	address_id	43,324	63,135	52,563.726	5,469.921	29,920,037....	0.177	-0.999	475,123,522	0	0	0	0	?	9039	43.324 63.135
area_id	area_id	1	18,452	5,313.506	6,661.241	44,372,128....	0.857	-0.786	48,028,782	0	0	0	0	?	9039	1 18.452
category_id	category_id	1	126	31.276	23.763	564.69	0.748	0.248	282,704	0	0	0	0	?	9039	1 126
date_id	date_id	1,012	1,367	1,172.45	100.159	10,031.749	0.288	-1.049	10,597,778	0	0	0	0	?	9039	1.012 1.367
unit_type_id	unit_type_id	1	294	18.277	18.338	336.285	6.767	86.769	165,208	0	0	0	0	?	9039	1 294
sale_price	sale_price	0	530,337,853	442,027.859	7,496,775.582	56,201,644,...	63.049	4,141.436	3,995,489,822	0	0	0	0	?	9039	0E0 5E8

5) 2010_statenisland

Here the total row count loaded into the database is 135. As all the duplicates and the null values for few attributes were pruned.

Table "default" - Rows: 6 Spec - Columns: 16 Properties Flow Variables

Row ID	Column	Min	Max	Mean	Std. de...	Variance	Skewness	Kurtosis	Overall ...	No. mis...	No. NaNs	No. +∞s	No. -∞s	Median	Row co...	Histogram
address_id	address_id	31,846	38,371	34,952.556	1,923	3,697,930.771	0.192	-1.253	4,718,595	0	0	0	0	?	135	31.846 38.371
area_id	area_id	1	16,677	11,553.015	6,345.905	40,270,514....	-1.037	-0.605	1,559,657	0	0	0	0	?	135	1 16.677
category_id	category_id	2	127	15.481	20.449	418.147	3.324	13.54	2,090	0	0	0	0	?	135	2 127
date_id	date_id	1,017	1,366	1,184.437	100.143	10,028.531	0.124	-1.076	159,899	0	0	0	0	?	135	1.017 1.366
unit_type_id	unit_type_id	1	265	23.489	22.227	494.043	9.621	105.818	3,171	0	0	0	0	?	135	1 265
sale_price	sale_price	0	5,775,000	267,496.052	554,362.768	307,318,07...	7.778	74.231	36,111,967	0	0	0	0	?	135	0 5.775.000

6) 2011_manhattan

Here the the total row count loaded into the database is 17202

Table "default" - Rows: 6 Spec - Columns: 16 Properties Flow Variables

Row ID	Column	Min	Max	Mean	Std. de...	Variance	Skewness	Kurtosis	Overall ...	No. mis...	No. NaNs	No. +∞s	No. -∞s	Median	Row co...	Histogram
address_id	address_id	63,137	80,095	71,689.925	4,667.371	21,784,356.58	-0.019	-1.023	1,233,210,088	0	0	0	0	?	17202	63.137 80.
area_id	area_id	1	19,360	659.095	3,466.215	12,014,643....	5.122	24.323	11,337,750	0	0	0	0	?	17202	1 19.
category_id	category_id	5	127	49.81	16.628	276.494	2.604	7.457	856,828	0	0	0	0	?	17202	5
date_id	date_id	1,371	1,726	1,547.931	98.578	9,717.569	0.042	-1.095	26,627,503	0	0	0	0	?	17202	1.371 1.
unit_type_id	unit_type_id	1	325	11.108	17.429	303.776	8.877	130.856	191,072	0	0	0	0	?	17202	1
sale_price	sale_price	0	760,000,000	1,408,214.054	10,015,655....	100,313,35...	44.958	2,654.437	24,224,098,...	0	0	0	0	?	17202	0E0

7) 2011_queens

Here the the total row count loaded into the database is 377. As all the duplicates and the null values for few attributes were pruned.

Table "default" - Rows: 6 Spec - Columns: 16 Properties Flow Variables

Row ID	Column	Min	Max	Mean	Std. de...	Variance	Skewness	Kurtosis	Overall ...	No. mis...	No. NaNs	No. +∞s	No. -∞s	Median	Row co...	Histogram
address_id	address_id	43,462	63,069	53,636.682	5,835.353	34,051,346....	0.013	-1.374	20,221,029	0	0	0	0	?	377	43.462 63.0
area_id	area_id	1	18,440	7,537.485	6,868.609	47,177,793....	0.331	-1.353	2,841,632	0	0	0	0	?	377	1 1
category_id	category_id	1	114	26.109	23.818	567.294	1.072	0.465	9,843	0	0	0	0	?	377	1
date_id	date_id	1,373	1,725	1,527.639	98.671	9,735.981	0.297	-1.003	575,920	0	0	0	0	?	377	1.373
unit_type_id	unit_type_id	1	292	22.504	26.217	687.357	8.246	82.405	8,484	0	0	0	0	?	377	1
sale_price	sale_price	0	21,500,000	347,877.515	1,146,350.544	1,314,119,5...	16.86	310.371	131,149,823	0	0	0	0	?	377	0 21.500

8) 2012_statenisland

Here the the total row count loaded into the database is 132.As all the duplicates and the null values for few attributes were pruned.

Table "default" - Rows: 6 Spec - Columns: 16 Properties Flow Variables

Row ID	Column	Min	Max	Mean	Std. de...	Variance	Skewness	Kurtosis	Overall ...	No. mis...	No. NaNs	No. +∞s	No. -∞s	Median	Row co...	Histogram
address_id	address_id	31,812	38,246	35,198.106	1,823.982	3,326,909.699	-0.113	-1.127	4,646,150	0	0	0	0	?	132	31.812 38
area_id	area_id	1	16,661	10,448.659	6,907.436	47,712,672....	-0.72	-1.287	1,379,223	0	0	0	0	?	132	1 16.
category_id	category_id	1	126	19.735	23.399	547.494	2.879	9.433	2,605	0	0	0	0	?	132	1
date_id	date_id	1,742	2,073	1,910.712	97.667	9,538.771	0.13	-1.15	252,214	0	0	0	0	?	132	1.742 2.
unit_type_id	unit_type_id	1	63	19.939	10.79	116.424	-0.245	1.448	2,632	0	0	0	0	?	132	1
sale_price	sale_price	0	3,500,000	259,167.871	462,188.316	213,618,03...	4.068	21.992	34,210,159	0	0	0	0	?	132	0 3.500

9) 2014_manhattan

Here the total row count loaded into the database is 2234.As all the duplicates and the null values for few attributes were pruned.

Table "default" - Rows: 5 | Spec - Columns: 16 | Properties | Flow Variables

Row ID	S Column	D Min	D Max	D Mean	D Std. de...	D Variance	D Skewness	D Kurtosis	D Overall ...	I No. mis...	I No. NaNs	I No. +∞s	I No. -∞s	D Median	I Row co...	Histogram
address_id	address_id	63,180	80,116	71,162.346	2,861.146	8,186,158.332	0.411	2.6	158,976,681	0	0	0	0	?	2234	63.180 80.
area_id	area_id	1	20,680	1,136.736	4,525.992	20,484,600....	3.741	12.008	2,539,469	0	0	0	0	?	2234	1 20.
category_id	category_id	30	156	125.135	45.292	2,051.374	-1.228	-0.429	279,551	0	0	0	0	?	2234	30
date_id	date_id	2,076	2,432	2,208.86	90.078	8,114.085	0.774	-0.264	4,934,593	0	0	0	0	?	2234	2.076 2
unit_type_id	unit_type_id	1	153	4.172	8.535	72.845	5.542	66.501	9,321	0	0	0	0	?	2234	1

The overall statistic of the number of rows loaded in the Postgre Database. This statistics is after pre-processing. Here the the total row count loaded into the database is 80,116 after all the duplicates and the null values for few attributes were pruned.

Table "default" - Rows: 5 | Spec - Columns: 16 | Properties | Flow Variables

Row ID	S Column	D Min	D Max	D Mean	D Std. de...	D Variance	D Skewness	D Kurtosis	D Overall ...	I No. mis...	I No. NaNs	I No. +∞s	I No. -∞s	D Median	I Row co...	Histogram
id	id	1	80,116	40,058.5	23,127.641	534,887,797....	0	-1.2	3,209,326,786	0	0	0	0	?	80116	1 80.116
borough	borough	1	5	2.928	1.218	1.484	-0.29	-0.805	234,595	0	0	0	0	?	80116	1 5
block	block	1	16,321	4,185.548	3,467.836	12,025,887.974	1.025	0.618	335,329,340	0	0	0	0	?	80116	1 16.321
lot	lot	1	9,116	335.198	657.617	432,460.157	3.897	26.245	26,854,736	0	0	0	0	?	80116	1 9.116
zipcode	zipcode	0	11,697	10,884	549.455	301,901.031	-0.722	0.551	871,982,558	0	0	0	0	?	80116	0 11.697

Snippet of data loaded into the address table:

	id	borough	neighborhood	block	lot	addr	zipcode
1	1	3	BATH BEACH	6,357	1	8683 14 AVENUE	11,228
2	2	3	BATH BEACH	6,357	12	8651 14TH AVENUE	11,228
3	3	3	BATH BEACH	6,357	21	8629 14TH AVENUE	11,228
4	4	3	BATH BEACH	6,357	57	56 BAY 7TH	11,228
5	5	3	BATH BEACH	6,357	60	64 BAY 7 STREET	11,228
6	6	3	BATH BEACH	6,357	63	72 BAY 7 STREET	11,228
7	7	3	BATH BEACH	6,357	66	82 BAY 7 STREET	11,228
8	8	3	BATH BEACH	6,358	11	55 BAY 7TH STREET	11,228
9	9	3	BATH BEACH	6,358	15	43 BAY 7 STREET	11,228
10	10	3	BATH BEACH	6,358	17	37 BAY 7 STREET	11,228
11	11	3	BATH BEACH	6,358	37	12 BAY 8TH STREET	11,228
12	12	3	BATH BEACH	6,358	41	24 BAY 8TH STREET	11,228
13	13	3	BATH BEACH	6,358	42	26 BAY 8TH STREET	11,228
14	14	3	BATH BEACH	6,358	48	44 BAY 8 STREET	11,228

Snippet of data loaded into the area table:

	id	land_sqaurefeet	gross_sqaurefeet
1	1	0	0
2	2	0	1,680
3	3	21	0
4	4	84	0
5	5	98	0
6	6	121	0
7	7	163	0
8	8	192	0
9	9	200	0
10	10	209	0

Snippet of data loaded into the category table:

	id	tax_class	building_class	building_category
1	1	1	A0	01 ONE FAMILY HOMES
2	2	1	A1	01 ONE FAMILY HOMES
3	3	1	A2	01 ONE FAMILY HOMES
4	4	1	A3	01 ONE FAMILY HOMES
5	5	1	A4	01 ONE FAMILY HOMES
6	6	1	A5	01 ONE FAMILY HOMES
7	7	1	A7	01 ONE FAMILY HOMES
8	8	1	A9	01 ONE FAMILY HOMES
9	9	1	S0	01 ONE FAMILY HOMES
10	10	1	S1	01 ONE FAMILY HOMES
11	11	1	B1	02 TWO FAMILY HOMES

Snippet of data loaded into the date table:

	id	year	month	dayofmonth	quarter	rawdate
1	1	2,005	1	1	1	2005-01-01
2	2	2,005	1	2	1	2005-01-02
3	3	2,005	1	3	1	2005-01-03
4	4	2,005	1	4	1	2005-01-04
5	5	2,005	1	5	1	2005-01-05

Snippet of data loaded into the unit_type table:

id	residential_units	commercial_units
1	0	0
2	0	1
3	0	2
4	0	3
5	0	4

Snippet of data loaded into the sales fact table:

	sale_price	address_id	category_id	unit_type_id	area_id	date_id
1	825000	1	11	29	4,623	296
2	710000	2	11	29	8,160	82
3	850000	3	11	29	8,163	184
4	1370000	4	11	29	8,167	3
5	695000	5	16	37	8,167	32

3.2 Hadoop Implementation

For implementation of project 2 we make use of Hadoop.

Here are the complete step by step implementation of ETL using Hadoop:

- First we create and execute a Docker Hadoop container. Hadoop container is created from a docker image, and the docker image is generated via a docker file. The docker container is run by the command:
  docker exec -it 641962d5b70f bash
  Here the container id is 641962d5b70f

- Next we create directories in Hadoop. We specify a directory for input as well as output. The input directory created here was in:

root/exerc/input
And the output directory path is:
root/exerc/output

- All 9 CSV files of NY property sales data are loaded into the Hadoop systems. Here is a screenshot of the files in /tmp folder

```
root@6b9a3e12a3f4:/tmp# ls /tmp/*csv
/tmp/2005_sales_brooklyn_05.csv   /tmp/2009_bronx.csv   /tmp/2010_statenisland.csv  /tmp/2011_queens.csv        /tmp/2014_manhattan.csv
/tmp/2008_sales_statenisland.csv  /tmp/2010_queens.csv  /tmp/2011_manhattan.csv     /tmp/2012_statenisland.csv
```

- The code for each of these steps are:
  - docker cp .Temp\ny\2005_sales_brooklyn_05.csv 6b9a3e12a3f4:/tmp (for all 9 csv files)
  - docker exec -it 6b9a3e12a3f4 bash
  - hadoop fs -mkdir /root
  - hadoop fs -mkdir /root/exerc/input
  - Hadoop fs -put /tmp/2005_sales_brooklyn_05.csv /root/exerc/input (for all 9 files)
  - hadoop fs -ls /root/exerc/input
- Next using 3 java files we perform Map reduce. We create MapIt.java, , ReduceIt.java, and MapReduceIt.java using the nano command.
- We use the MapIt function specific to the scope of the job.
  - Mapit function reads line by line
  - First we text variables to string using toString()
  - Next if the length of my_line is equal to zero we return
  - Since our data is in form of CSV files we use "," delimiter to separate all tokens/attributes
  - Since we are loading a structured data we do not consider those lines who have tokens != 21.
  - Next we assign each token a variable and do some preprocessing on it. Like trimming, replacing ",","$", converting sales and other numeric attributes to Double, converting rawdate to yyyy-MM-dd HH:mm:ss format,etc.
  - Since this is loaded in key value pairs we assign the combination of each attribute (a line) as a key and value as 1. Hence all unique values of lines will have key value as 1.

```java
public class MapIt extends Mapper<Object, Text, Text, IntWritable>{
    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
    // ------------------------------------------------------------------------
    // --- your code should start here
        String my_line=value.toString();
               //System.out.println("Length of line is "+my_line.length());

        if(my_line.trim().length()==0){
            //System.out.println(my_line);
            return;
        }


        String[] words = my_line.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)", -1);
        //System.out.println(words.length);
        if(words.length!=21){
            //System.out.println(words[1]+","+words[20]+","+words[19] );
            return;
        }
        //System.out.println("Sales Price original"+words[19] );

        String borough = words[0].trim();
        String neighborhood = words[1].trim();
        String building_class_category = words[2].trim();
        String block=words[4].trim();
        String lot=words[5].trim();
        String address = words[8].trim().replace(',',' ');
        String zipcode = words[10].trim();
        String residential_units = words[11].trim();
        String commercial_units = words[12].trim();
        //String land_squarefeet = words[14].trim().replace(',',' ');
        //String gross_squarefeet = words[15].trim().replace(',',' ');
        String tax_class = words[17].trim(); //tax class at time of sale
        String building_class = words[18].trim(); //building_class at time of sale
        String rawdate = words[20];

        try {
            //String sales_price = words[19].replaceAll("[$,]", "");
            String sales_price = words[19].replaceAll("[$,]", "").replace("\"","");
```

```
        //String sales_price = words[19].replaceAll("[$,]", "");
        String sales_price = words[19].replaceAll("[$,]", "").replace("\"","");
        Double.parseDouble(sales_price);
    } catch (NumberFormatException e)

    {
        //System.out.println(words[19].replaceAll("[$,]", "").replace("\"",""));
       // System.out.println("Error with"+ Double.parseDouble(words[19].replaceAll("[$,]", "").toString()));
        return;
    }
    String  sales_price = words[19].replaceAll("[$,]", "").replace("\"","");
    double sale_price = Double.parseDouble(sales_price);
    //System.out.println("Sales Price is "+sales_price+","+words[19]);
    if(neighborhood.length()==0){
        return;
    }

    try {             String land_squarefeet = words[14].replaceAll("[$,]", "").replace("\"","");
        Double.parseDouble(land_squarefeet);
    } catch (NumberFormatException e){
        return;
    }
    String  land_squarefeet = words[14].replaceAll("[$,]", "").replace("\"","");
    Double.parseDouble(land_squarefeet);
    try {
        String gross_squarefeet = words[15].replaceAll("[$,]", "").replace("\"","");
        Double.parseDouble(gross_squarefeet);
    } catch (NumberFormatException e){
        return;
    }
    String  gross_squarefeet = words[15].replaceAll("[$,]", "").replace("\"","");
    Double.parseDouble(gross_squarefeet);

    SimpleDateFormat dateFormatInput = new SimpleDateFormat("MM/d/yyyy");
    SimpleDateFormat dateFormatOutput = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    try {
        Date orderdate = dateFormatInput.parse(rawdate.trim());
        // if orderdate is valid format it for the Hive table
        rawdate = dateFormatOutput.format(orderdate);
    } catch (Exception e) {  // data is notin the proper format
        //System.out.println("Date-Time is "+date_time);
```

```
        SimpleDateFormat dateFormatInput = new SimpleDateFormat("MM/d/yyyy");
        SimpleDateFormat dateFormatOutput = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        try {
            Date orderdate = dateFormatInput.parse(rawdate.trim());
            // if orderdate is valid format it for the Hive table
            rawdate = dateFormatOutput.format(orderdate);
        } catch (Exception e) {  // data is notin the proper format
            //System.out.println("Date-Time is "+date_time);
            return; // skip this line
        }

        String my_str_key = borough+","+neighborhood+","+building_class_category+","+block+","+lot+","+address+","+zipcode+","+residential_units+","+commer>
        int my_int_val =1;
Text my_key = new Text(my_str_key);
IntWritable my_value = new IntWritable(my_int_val);
context.write(my_key,my_value);
    // --- your code should end here
    // ----------------------------------------------------------------------
    }
}
// end of the map class
```

- After creating the source files, they are complied, and the jar file is created.
- With the help of code provided by Dr. Barb I input the code and complied the MapReduceIt and ReduceIt java files. Then ran them using the runit.sh file.
- The above steps were done using the below codes
    - nano MapIt.java

- nano ReduceIt.java
- nano MapReduceIt.java
- hadoop com.sun.tools.javac.Main *.java
- jar cf mri.jar *.class
- cd /usr/local/hadoop
- bin/hadoop jar /root/inclass/mri.jar MapReduceIt /root/exerc/input /root/exerc/output

- Here I gave the input and output directory path as mentioned above and saved the results in text format at /tmp/result.txt.
  - nano runit.sh
  - chmod a+x runit.sh
  - ./runit.sh
  - nano /tmp/result.txt

- This file contains all the attributes separated by comma. It will be used to load the attributes into the Hive tables.

- Next to load into Hive tables. First we need to create a database. To do so we used:

```
0: jdbc:hive2://localhost:10000> CREATE DATABASE IF NOT EXISTS assignmenthive;
No rows affected (0.595 seconds)
0: jdbc:hive2://localhost:10000> SHOW DATABASES;
+-----------------+
|  database_name  |
+-----------------+
| assignmenthive  |
| default         |
+-----------------+
2 rows selected (0.133 seconds)
0: jdbc:hive2://localhost:10000> USE assignmenthive;
No rows affected (0.05 seconds)
```

- To create the table with our own attributes and their data types we use:

  CREATE TABLE prop_sales(borough INT,neighborhood VARCHAR(100),building_class_category VARCHAR(100),block INT,lot INT,address VARCHAR(100),zipcode INT,residential_units INT,commercial_units INT,land_squarefeet INT,gross_sqaurefeet INT,tax_class INT,building_class VARCHAR(20),sale_price INT,sale_date TIMESTAMP,count VARCHAR(50)) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','STORED AS TEXTFILE;

- To check the created table we can just type DESCRIBE prop_sales;

```
0: jdbc:hive2://localhost:10000> DESCRIBE prop_sales;
+--------------------------+----------------+----------+
|         col_name         |   data_type    | comment  |
+--------------------------+----------------+----------+
| borough                  | int            |          |
| neighborhood             | varchar(100)   |          |
| building_class_category  | varchar(100)   |          |
| block                    | int            |          |
| lot                      | int            |          |
| address                  | varchar(100)   |          |
| zipcode                  | int            |          |
| residential_units        | int            |          |
| commercial_units         | int            |          |
| land_squarefeet          | int            |          |
| gross_sqaurefeet         | int            |          |
| tax_class                | int            |          |
| building_class           | varchar(20)    |          |
| sale_price               | int            |          |
| sale_date                | timestamp      |          |
| count                    | varchar(50)    |          |
+--------------------------+----------------+----------+
```

- Then to load the data into the created table we type:
  - load data local inpath '/tmp/result.txt' overwrite into table assignmenthive.prop_sales;

```
| prop_sales.borough  | prop_sales.neighborhood  |      prop_sales.building_class_category      | prop_sales.block  | pr
op_sales.lot  |              prop_sales.address              | prop_sales.zipcode  | prop_sales.residential_units  | p
rop_sales.commercial_units  | prop_sales.land_squarefeet  | prop_sales.gross_sqaurefeet  | prop_sales.tax_class  | prop_
sales.building_class  | prop_sales.sale_price  |  prop_sales.sale_date  | prop_sales.count  |
| 5                   | WEST NEW BRIGHTON        | 01  ONE FAMILY HOMES                            | 123               | 61
            | 298 BRIGHTON AVENUE       | 10301               | 1                             | 0
     | 4200                      | 1152                       | 1                     | A1                              |
 310000               | 2010-06-24 00:00:00.0  |      1            |
| 5                   | WEST NEW BRIGHTON        | 01  ONE FAMILY HOMES                            | 124               | 10
            | 311 STANLEY AVENUE        | 10301               | 1                             | 0
     | 2832                      | 1560                       | 1                     | A1                              |
```

- By using Select count(*) from assignmenthive.prop_sales; we check the number of lines loaded

```
0: jdbc:hive2://localhost:10000> select count(*) from assignmenthive.prop_sales;
+----------+
|   _c0    |
+----------+
| 143564   |
+----------+
```

- As we can see that the entire 9 files of NY property sales data are loaded into the HIVE table on port number 10,000.
- Using this port number, “student” as credentials and loacalhost as hostname we connect Hive connector in Knime to our Hive implementation on top of Hadoop.
- After which using the DB query reader in Knime we can see all the lines that were loaded into the Hive table. As we can see the number of rows and columns called through Db query reader is exact similar to the lines loaded on hive.

Table "database" - Rows: 143564 | Spec - Columns: 16 | Properties | Flow Variables

Row ID	borough	neighbor...	building_class_category	block	lot	address	zipcode	residen...	commer...	land_sq...	gross_s...	tax_class	building...	sale_price	sale_date	count
Row0	1	ALPHABET CITY	01 ONE FAMILY DWELLINGS	402	42	96 AVENUE B	10009	1	1	1400	1839	1	S1	0	2014-12-05T00...	1
Row1	1	ALPHABET CITY	01 ONE FAMILY HOMES	376	43	743 EAST 6...	10009	1	1	2090	3680	1	S1	10	2011-08-26T00...	1
Row2	1	ALPHABET CITY	02 TWO FAMILY DWELLINGS	372	37	17 AVENUE D	10002	2	1	826	2075	1	S2	0	2014-01-31T00...	1
Row3	1	ALPHABET CITY	02 TWO FAMILY HOMES	373	1	40 AVENUE C	10009	2	1	1923	4800	1	S2	0	2011-12-21T00...	1
Row4	1	ALPHABET CITY	03 THREE FAMILY DWELLI...	377	66	243 EAST 7...	10009	3	0	2381	3084	1	C0	2900000	2014-11-26T00...	1
Row5	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	372	31	316 EAST 3...	10009	4	0	5746	2700	2	C3	3500000	2011-09-08T00...	1
Row6	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	372	51	300 EAST 2...	10009	8	2	2650	7620	2	C7	0	2014-07-18T00...	1
Row7	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	373	15	324 EAST 4...	10009	11	0	2212	8294	2	C3	0	2014-05-30T00...	1
Row8	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	373	63	285 EAST 3...	10009	4	0	2084	2917	2	C3	3100000	2014-08-05T00...	1
Row9	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	376	54	719 E 6TH ST	10009	16	0	3437	9180	2	C4	2966835	2011-07-25T00...	1
Row10	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	377	2	116 AVENUE C	10009	22	3	4510	19830	2	C7	0	2011-06-10T00...	1

- To call only those set of lines which have unique values, we get rid of lines with count more than 1 by using select * from `assignmenthive`.`prop_sales` WHERE COUNT=1. The number of lines then reduces from 143564 to 141417.

Table "database" - Rows: 141417 | Spec - Columns: 16 | Properties | Flow Variables

Row ID	borough	neighbor...	building_class_category	block	lot	address	zipcode	residen...	commer...	land_sq...	gross_s...	tax_class	building...	sale_price	sale_date	count
Row0	1	ALPHABET CITY	01 ONE FAMILY DWELLINGS	402	42	96 AVENUE B	10009	1	1	1400	1839	1	S1	0	2014-12-05T00...	1
Row1	1	ALPHABET CITY	01 ONE FAMILY HOMES	376	43	743 EAST 6...	10009	1	1	2090	3680	1	S1	10	2011-08-26T00...	1
Row2	1	ALPHABET CITY	02 TWO FAMILY DWELLINGS	372	37	17 AVENUE D	10002	2	1	826	2075	1	S2	0	2014-01-31T00...	1
Row3	1	ALPHABET CITY	02 TWO FAMILY HOMES	373	1	40 AVENUE C	10009	2	1	1923	4800	1	S2	0	2011-12-21T00...	1
Row4	1	ALPHABET CITY	03 THREE FAMILY DWELLI...	377	66	243 EAST 7...	10009	3	0	2381	3084	1	C0	2900000	2014-11-26T00...	1
Row5	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	372	31	316 EAST 3...	10009	4	0	5746	2700	2	C3	3500000	2011-09-08T00...	1
Row6	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	372	51	300 EAST 2...	10009	8	2	2650	7620	2	C7	0	2014-07-18T00...	1
Row7	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	373	15	324 EAST 4...	10009	11	0	2212	8294	2	C3	0	2014-05-30T00...	1
Row8	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	373	63	285 EAST 3...	10009	4	0	2084	2917	2	C3	3100000	2014-08-05T00...	1
Row9	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	376	54	719 E 6TH ST	10009	16	0	3437	9180	2	C4	2966835	2011-07-25T00...	1
Row10	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	377	2	116 AVENUE C	10009	22	3	4510	19830	2	C7	0	2011-06-10T00...	1

Row ID	S Column	D Min	D Max	D Mean	D Std. de...	D Variance	D Skewness	D Kurtosis	D Overall ...	I No. mis...	I No. NaNs	I No. +∞s	I No. -∞s	D Median	I Row co...	Histogram
borough	borough	1	5	2.941	1.432	2.049	-0.222	-1.329	415,890	0	0	0	0	?	141417	
block	block	1	16,322	3,946.97	3,616.907	13,082,017....	1.188	0.81	558,168,601	0	0	0	0	?	141417	
lot	lot	1	9,116	371.449	665.673	443,120.555	3.516	23.509	52,529,191	0	0	0	0	?	141417	
zipcode	zipcode	0	11,697	10,755.965	655.416	429,570.217	-3.001	45.523	1,521,076,268	0	0	0	0	?	141417	
residential_units	residential_...	0	774	2.006	12.005	144.132	31.477	1,361.904	283,678	0	0	0	0	?	141417	
commercial_u...	commercial_...	0	448	0.217	3.057	9.346	85.617	9,724.315	30,756	0	0	0	0	?	141417	
land_squarefeet	land_square...	0	11,320,000	2,692.187	48,772.61	2,378,767,4...	200.949	43,815.141	380,720,977	0	0	0	0	?	141417	
gross_sqaure...	gross_sqaur...	0	2,725,731	3,817.185	24,776.769	613,888,28...	32.49	2,028.117	539,814,884	0	0	0	0	?	141417	
tax_class	tax_class	1	4	1.716	0.91	0.829	1.419	1.321	242,672	0	0	0	0	?	141417	
sale_price	sale_price	0	9,992,000	523,863.61	951,322.986	905,015,42...	4.666	28.531	73,396,434,...	1311	0	0	0	?	141417	

3.3 Reflective analysis of data preparation in relational data warehouse vs Hadoop.

While preparing the data for data warehousing, the level of granularity for the date had to be determined. For the date dimension, it had to be determined if we need to consider the month of year, day of year, etc. Another aspect that needed to be determined was which rows are to be pruned, groupby and considered for missing values. Here we pruned those rows which had no sales price data. Another analysis done were checking if all the date values are in right date format and to make sure the name of the attributes matched to database attribute names.

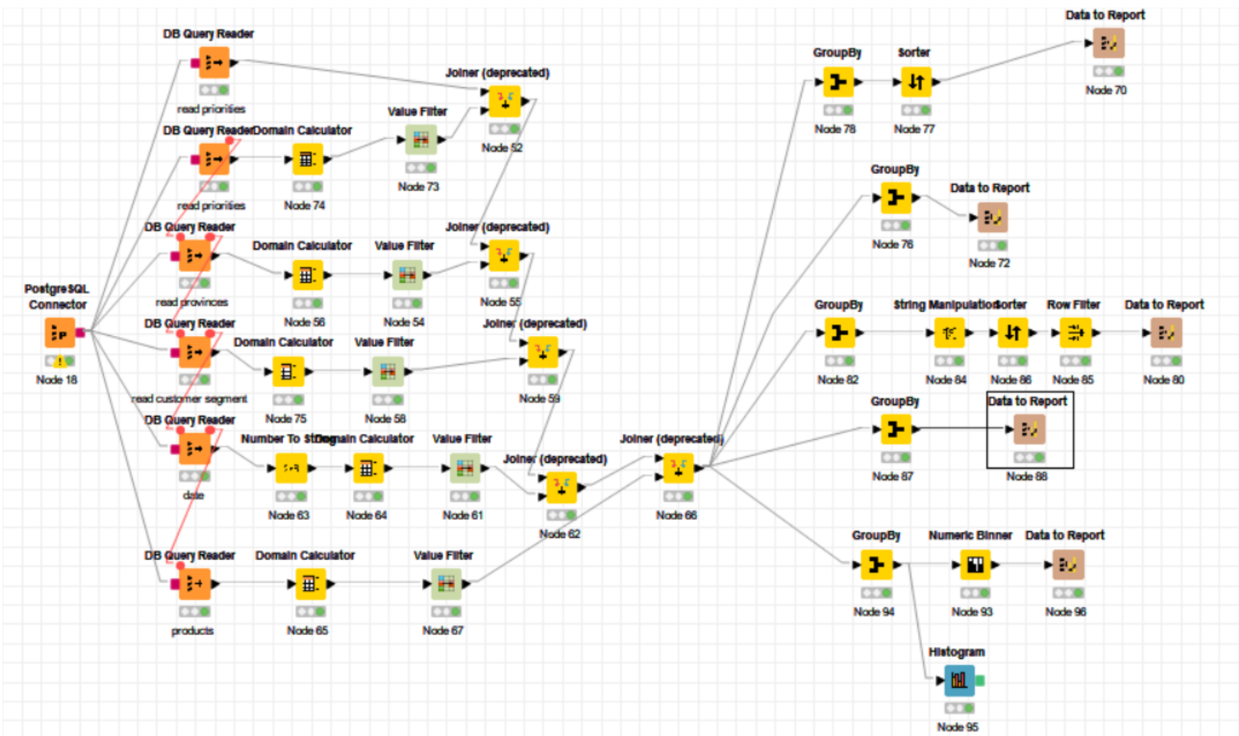
Whereas transforming and loading using Hadoop was a little more challenging as each attribute had to be converted and loaded to their proper respective variable data types. This was done using the MapIt reduce. Using count as an additional column we can prune those lines which have count more than 1 (to remove duplicates). Since we did not do other pre-processing like groupby on the data (unlike transform process followed in warehousing), we stored a little number of rows/lines using Hadoop implementation. The mapit function loads line by line.

As transforming in warehouse implementation required data transformation of each dimension separately, it was more complex.

4. Reporting System

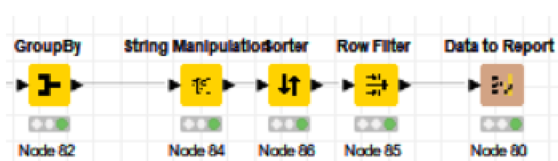
4.1 Relational Data Warehouse Implementation

Overall reporting workflow in Knime:

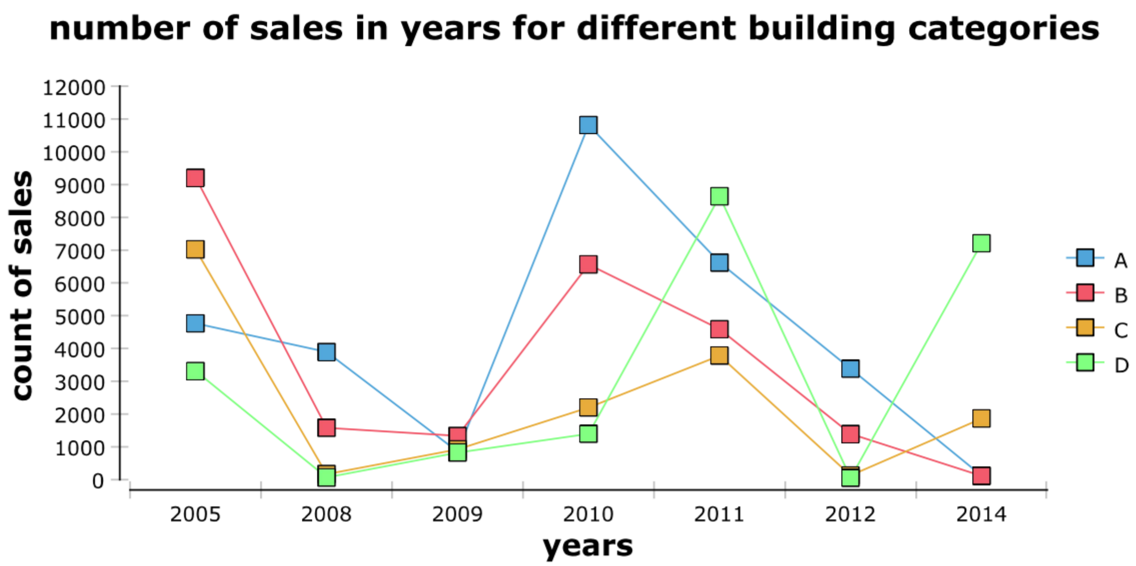


Business question1: Number of sales over the years for different building categories.

Here we manually aggregate sales price to count using GroupBy as we need the count of sales done. Then we group by building class and year. Since the building categories are names as A1,A2,A3………....Z9, we need to remove the last number to just keep the building categories (A,B,C,D..). We do this by using string manipulation by removing last character (removeChars($building_class$,"0,1,2,3,4,5,6,7,8,9")). After this using sorter and row filter we choose only the top most classes (A, B, C, D) for our visualizations. Finally, we put count of sales on y axes, years on x axes, and building class in y-series grouping.

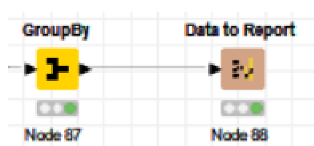


We can observe that all building categories had a great fall in the number of sales in 2012. At the same time all building categories had an increase in sales count in 2010 with category A recording the highest number of sales.

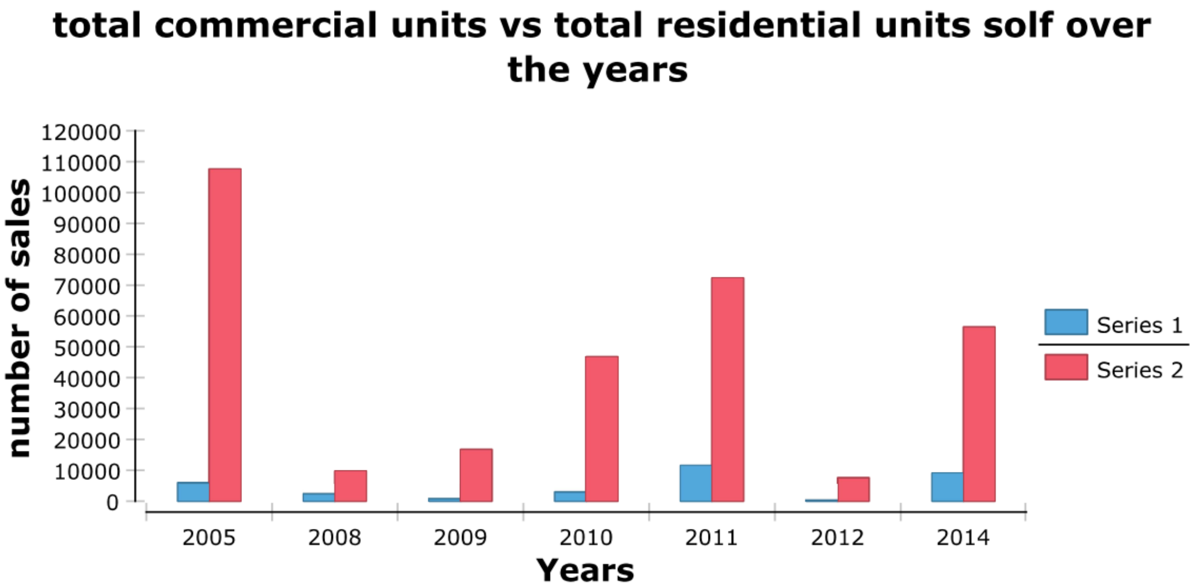


Business Question2: To determine the number of commercial units and residential units sold over the years

Here we use GroupBy to group year-wise, then we manually aggregate commercial units and residential units as total sum. We also manually aggregate sales_price as count (same as before). Then using Data to report we build a bar graph with number of sales in y-axis, years in x-axis and numer of total units as y-series grouping (series1 as commercial units, series 2 as residential units) .

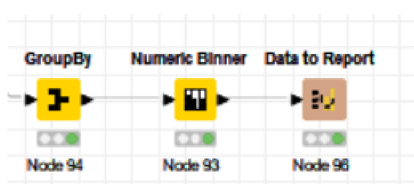


Here we can see that rersidential units sold over the years are much more than commercial units. 2008 and 2009 had a drastic drop in number of sales in both type of units.

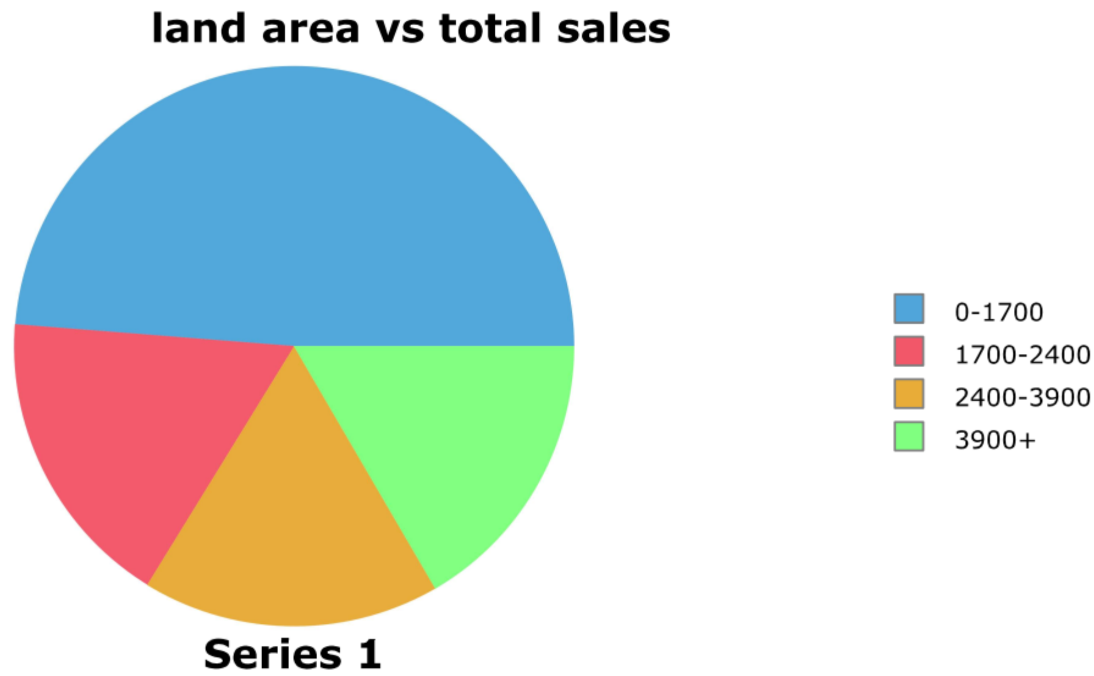


Business question3: Number of sales per year according to different binned land areas.

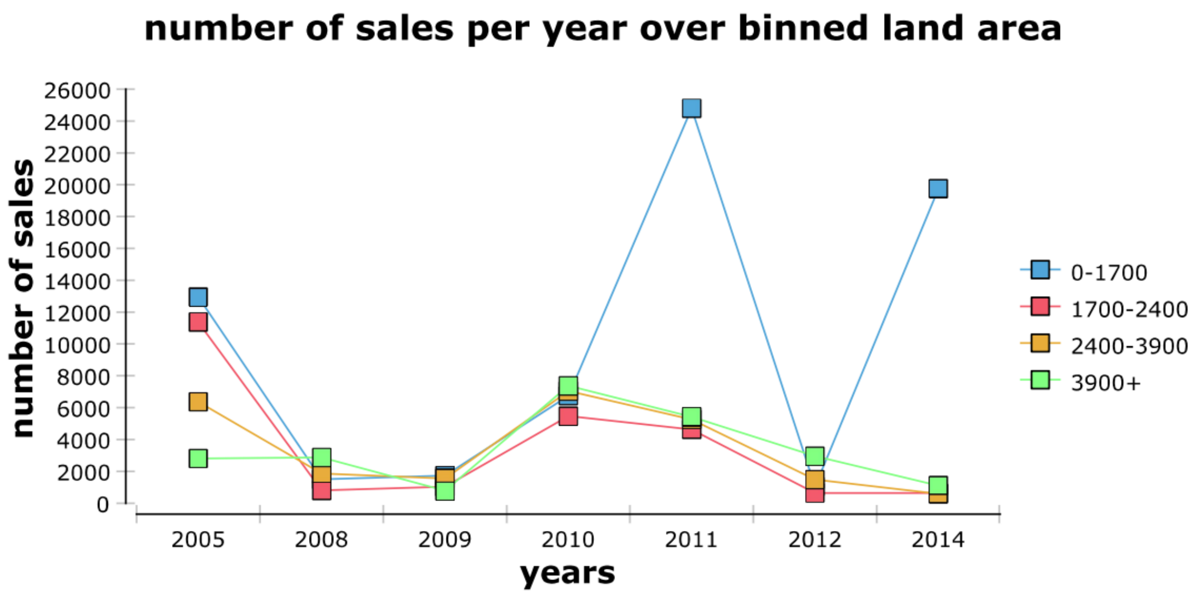
Here we use GroupBy to group year, land_squarefeet, then we manually aggregate commercial sales_price as count (same as before). After which we make bins for the land square feet with the help of numeric binner. Since the land square feet was continous values we made 4 bins(0-1700, 1700-2400, 2400-3900, 3900+). Then using Data to report we build a pie chart and line graph with number of sales in y-axis, years in x-axis and binned land areas as y-series grouping.



First representation (pie chart) shows that almost 50% of the total sales over the years are of properties with land square feet lower than 1700. The other three land square feet bins have similar portion of sales.

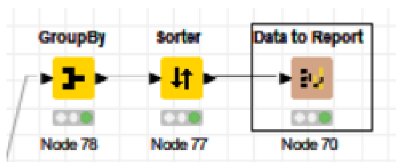


The second representation (line graph) proves that land square feet below the 1700 mark was almost always highest number of sales per year. It had a massive rise in the number of sales in the year 2011 and 2014. Whereas again 2012 had a drop in number of sales across all the categories

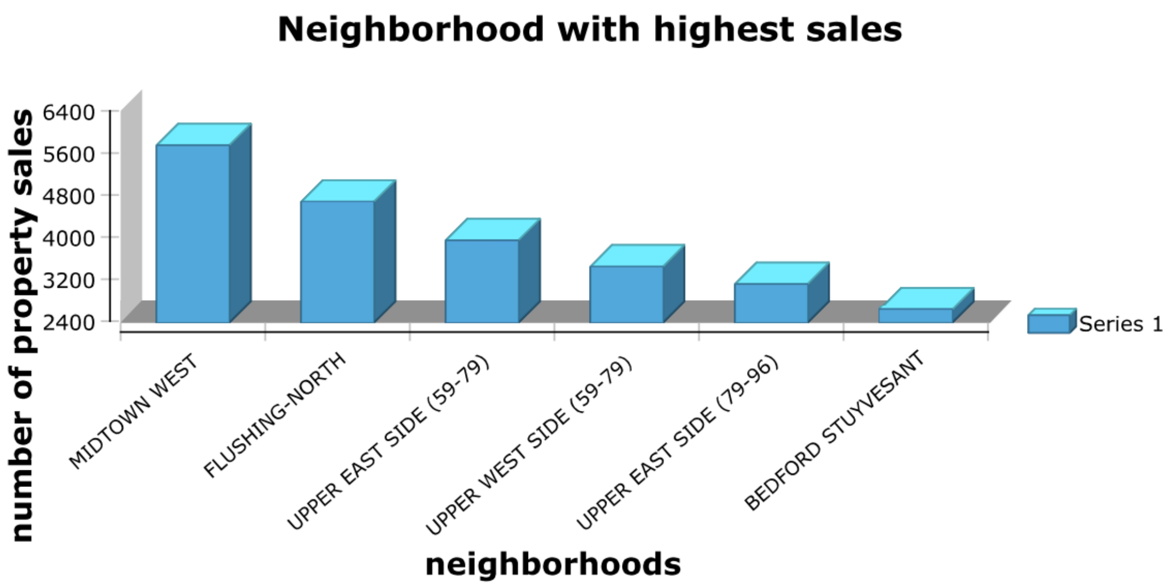


Business question4: To calculate the top 5 neighborhood with the greatest number of sales

To start with we will prepare the data, this can be achieved by using joiner nodes. To connect 2 tables at a time. Then we will deploy group by node to group the data by neighborhood, and count of sale. Our next step is to use sorter node to sort the table in ascending order to get the top neighborhood with highest sales. Next, we prune the rows after 6 rows. This allows us to isolate the top neighborhoods. Next, we will deploy reporting node and shift to BIRT tool. Then, insert bar chart, and assign x axis as borough and y axis as count of sales. The below image can be used to answer our business question.

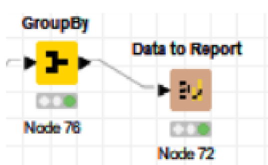


From the above graph we can analyze that MIDTOWN WEST has the highest sales. And BEDFORD STUYVESANT has the sixth highest sales.

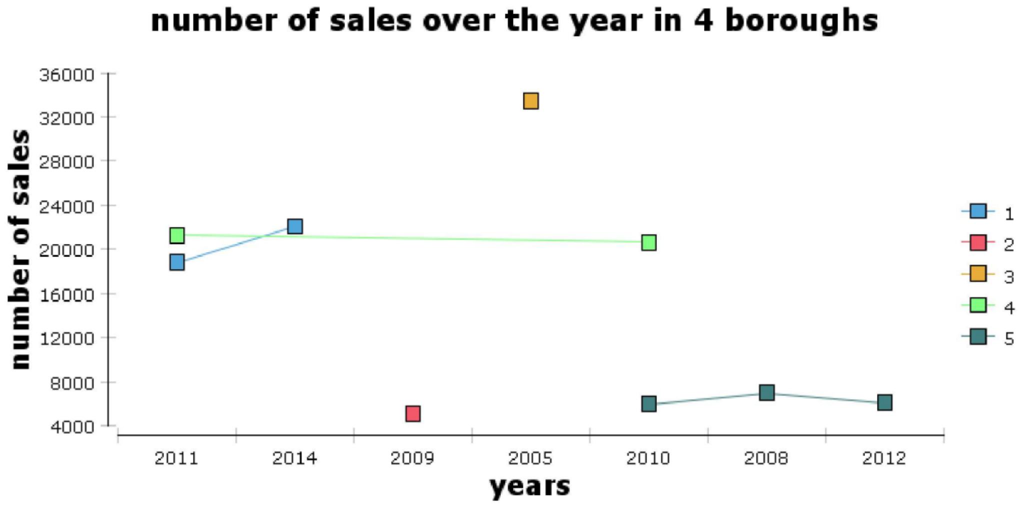


Business question5: Number of sales over the years in four boroughs

Here we use GroupBy to group year-wise and boroughs, then we manually aggregate commercial sales_price as count (same as before). Then using Data to report we build a bar graph with number of sales in y-axis, years in x-axis and numer of total units as y-series grouping (series1 as commercial units, series 2 as residential units) .



Here we do not have multiple year data for borough 2 and 3, hence they are represented only by a dot and cannot show a trend over the year. Whereas we can see that borough 4 and 5 had quite the same number of sales over the years. Borough 1 had an increase in the number of sales from 2011 to 2014.



4.2 Hadoop Implementation

After creating Hive table and storing data, using the DB query reader in Knime we can see all the lines that were loaded into the Hive table. As we can see the number of rows and columns called through Db query reader is exact similar to the lines loaded on hive.

Table "database" - Rows: 143564 Spec - Columns: 16 Properties Flow Variables

Row ID	borough	neighbor...	building_class_category	block	lot	address	zipcode	residen...	commer...	land_sq...	gross_s...	tax_class	building...	sale_price	sale_date	count
Row0	1	ALPHABET CITY	01 ONE FAMILY DWELLINGS	402	42	96 AVENUE B	10009	1	1	1400	1839	1	S1	0	2014-12-05T00...	1
Row1	1	ALPHABET CITY	01 ONE FAMILY HOMES	376	43	743 EAST 6...	10009	1	1	2090	3680	1	S1	10	2011-08-26T00...	1
Row2	1	ALPHABET CITY	02 TWO FAMILY DWELLINGS	372	37	17 AVENUE D	10002	2	1	826	2075	1	S2	0	2014-01-31T00...	1
Row3	1	ALPHABET CITY	02 TWO FAMILY HOMES	373	1	40 AVENUE C	10009	2	1	1923	4800	1	S2	0	2011-12-21T00...	1
Row4	1	ALPHABET CITY	03 THREE FAMILY DWELLI...	377	66	243 EAST 7...	10009	3	0	2381	3084	1	C0	2900000	2014-11-26T00...	1
Row5	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	372	31	316 EAST 3...	10009	4	0	5746	2700	2	C3	3500000	2011-09-08T00...	1
Row6	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	372	51	300 EAST 2...	10009	8	2	2650	7620	2	C7	0	2014-07-18T00...	1
Row7	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	373	15	324 EAST 4...	10009	11	0	2212	8294	2	C3	0	2014-05-30T00...	1
Row8	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	373	63	285 EAST 3...	10009	4	0	2084	2917	2	C3	3100000	2014-08-05T00...	1
Row9	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	376	54	719 E 6TH ST	10009	16	0	3437	9180	2	C4	2966835	2011-07-25T00...	1
Row10	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	377	2	116 AVENUE C	10009	22	3	4510	19830	2	C7	0	2011-06-10T00...	1

To call only those set of lines which have unique values, we get rid of lines with count more than 1 by using select * from `assignmenthive`.`prop_sales` WHERE COUNT=1. The number of lines then reduces from 143564 to 141417.

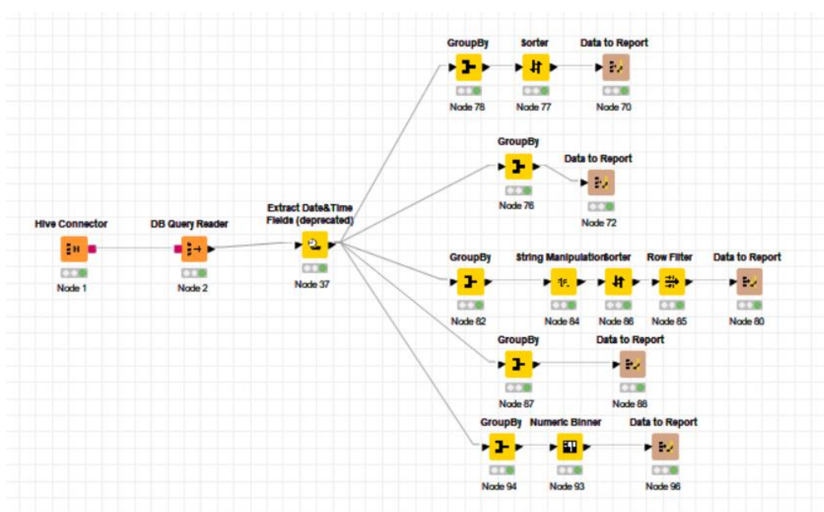
Table "database" - Rows: 141417 Spec - Columns: 16 Properties Flow Variables

Row ID	borough	neighbor...	building_class_category	block	lot	address	zipcode	residen...	commer...	land_sq...	gross_s...	tax_class	building...	sale_price	sale_date	count
Row0	1	ALPHABET CITY	01 ONE FAMILY DWELLINGS	402	42	96 AVENUE B	10009	1	1	1400	1839	1	S1	0	2014-12-05T00...	1
Row1	1	ALPHABET CITY	01 ONE FAMILY HOMES	376	43	743 EAST 6...	10009	1	1	2090	3680	1	S1	10	2011-08-26T00...	1
Row2	1	ALPHABET CITY	02 TWO FAMILY DWELLINGS	372	37	17 AVENUE D	10002	2	1	826	2075	1	S2	0	2014-01-31T00...	1
Row3	1	ALPHABET CITY	02 TWO FAMILY HOMES	373	1	40 AVENUE C	10009	2	1	1923	4800	1	S2	0	2011-12-21T00...	1
Row4	1	ALPHABET CITY	03 THREE FAMILY DWELLI...	377	66	243 EAST 7...	10009	3	0	2381	3084	1	C0	2900000	2014-11-26T00...	1
Row5	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	372	31	316 EAST 3...	10009	4	0	5746	2700	2	C3	3500000	2011-09-08T00...	1
Row6	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	372	51	300 EAST 2...	10009	8	2	2650	7620	2	C7	0	2014-07-18T00...	1
Row7	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	373	15	324 EAST 4...	10009	11	0	2212	8294	2	C3	0	2014-05-30T00...	1
Row8	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	373	63	285 EAST 3...	10009	4	0	2084	2917	2	C3	3100000	2014-08-05T00...	1
Row9	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	376	54	719 E 6TH ST	10009	16	0	3437	9180	2	C4	2966835	2011-07-25T00...	1
Row10	1	ALPHABET CITY	07 RENTALS - WALKUP AP...	377	2	116 AVENUE C	10009	22	3	4510	19830	2	C7	0	2011-06-10T00...	1

Row ID	S Column	D Min	D Max	D Mean	D Std. de...	D Variance	D Skewness	D Kurtosis	D Overall ...	I No. mis...	I No. NaNs	I No. +∞s	I No. -∞s	D Median	I Row co...	Histogram
borough	borough	1	5	2.941	1.432	2.049	-0.222	-1.329	415,890	0	0	0	0	?	141417	1 5
block	block	1	16,322	3,946.97	3,616.907	13,082,017....	1.188	0.81	558,168,601	0	0	0	0	?	141417	1 16.322
lot	lot	1	9,116	371.449	665.673	443,120.555	3.516	23.509	52,529,191	0	0	0	0	?	141417	1 9.116
zipcode	zipcode	0	11,697	10,755.965	655.416	429,570.217	-3.001	45.523	1,521,076,268	0	0	0	0	?	141417	0 11.697
residential_units	residential_...	0	774	2.006	12.005	144.132	31.477	1,361.904	283,678	0	0	0	0	?	141417	0 774
commercial_u...	commercial_...	0	448	0.217	3.057	9.346	85.617	9,724.315	30,756	0	0	0	0	?	141417	0 448
land_squarefeet	land_square...	0	11,320,000	2,692.187	48,772.61	2,378,767,4...	200.949	43,815.141	380,720,977	0	0	0	0	?	141417	0 11.320.000
gross_sqaure...	gross_sqaur...	0	2,725,731	3,817.185	24,776.769	613,888,28...	32.49	2,028.117	539,814,884	0	0	0	0	?	141417	0 2.725.731
tax_class	tax_class	1	4	1.716	0.91	0.829	1.419	1.321	242,672	0	0	0	0	?	141417	1 4
sale_price	sale_price	0	9,992,000	523,863.61	951,322.986	905,015,42...	4.666	28.531	73,396,434,...	1311	0	0	0	?	141417	0 9.992.000

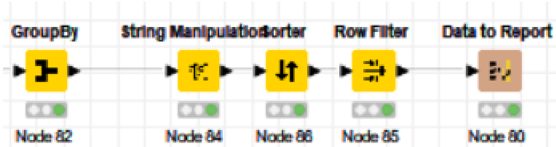
Overall reporting workflow in Knime followed these steps before answering each business question:

- First using the Hive connecter the table stored in Hive were called on Knime by using the right port number, user credentials and database name.
- Then using the DB query reader we selected all the data where the count is equal to one. By doing so we make sure we do not include the duplicate values.
- Initially using the Extract date&Time node we separate the month number, year and store in a separate column for analysis purposes.

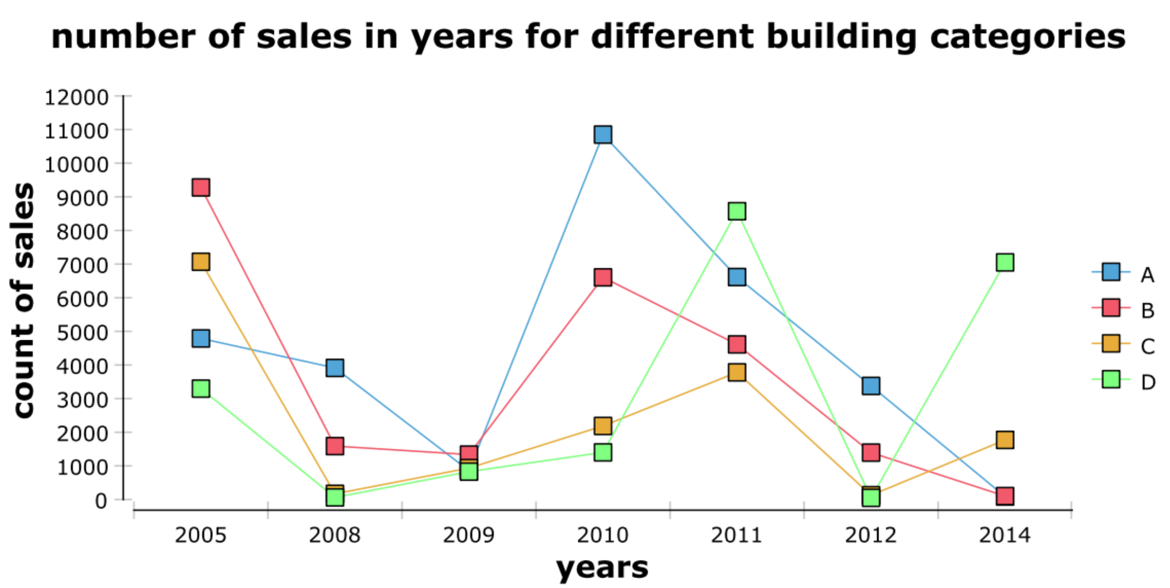


Business question1: Number of sales over the years for different building categories.

Here we manually aggregate sales_price to count using GroupBy as we need the count of sales done. Then we group by building class and year. Since the building categories are names as A1,A2,A3………...Z9, we need to remove the last number to just keep the building categories (A,B,C,D..). We do this by using string manipulation by removing last character (removeChars($building_class$,"0,1,2,3,4,5,6,7,8,9")). After this using sorter and row filter we choose only the top most classes (A,B,C,D) for our visualizations. Finally we put count of sales on y axes, years on x axes, and building class in y-series grouping.

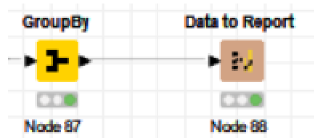


We can observe that all building categories had a great fall in the number of sales in 2012. At the same time all building categories had an increase in sales count in 2010 with category A recording the highest number of sales.

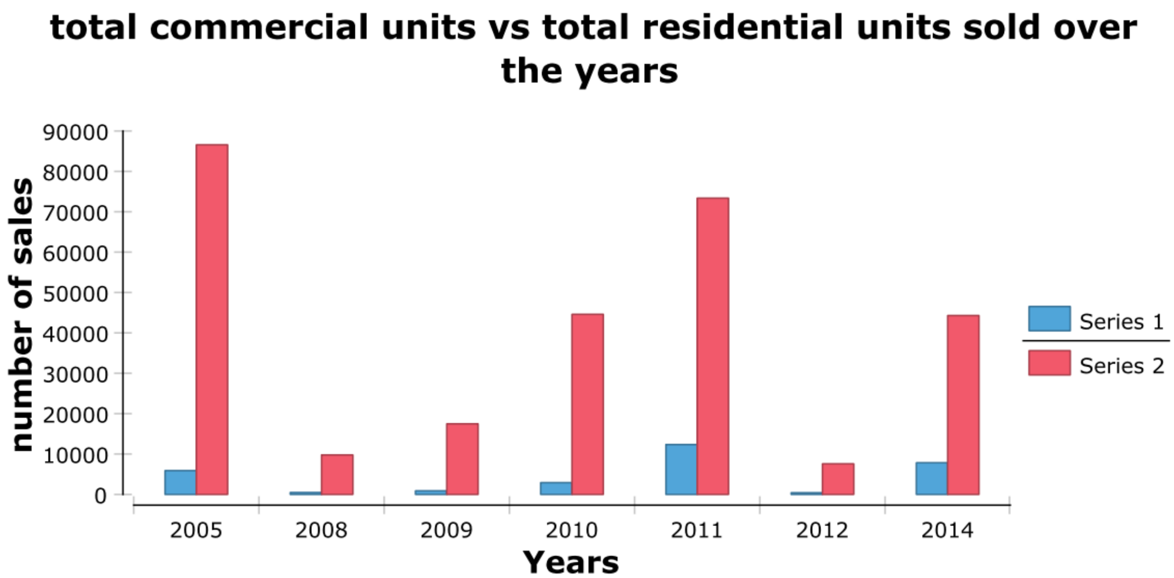


Business Question2: To determine the number of commercial units and residential units sold over the years

Here we use GroupBy to group year-wise, then we manually aggregate commercial units and residential units as total sum. We also manually aggregate sales_price as count (same as before). Then using Data to report we build a bar graph with number of sales in y-axis, years in x-axis and numer of total units as y-series grouping (series1 as commercial units, series 2 as residential units) .

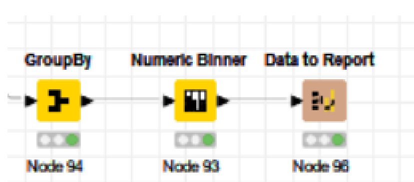


Here we can see that rersidential units sold over the years are much more than commercial units. 2008 and 2009 had a drastic drop in number of sales in both type of units.

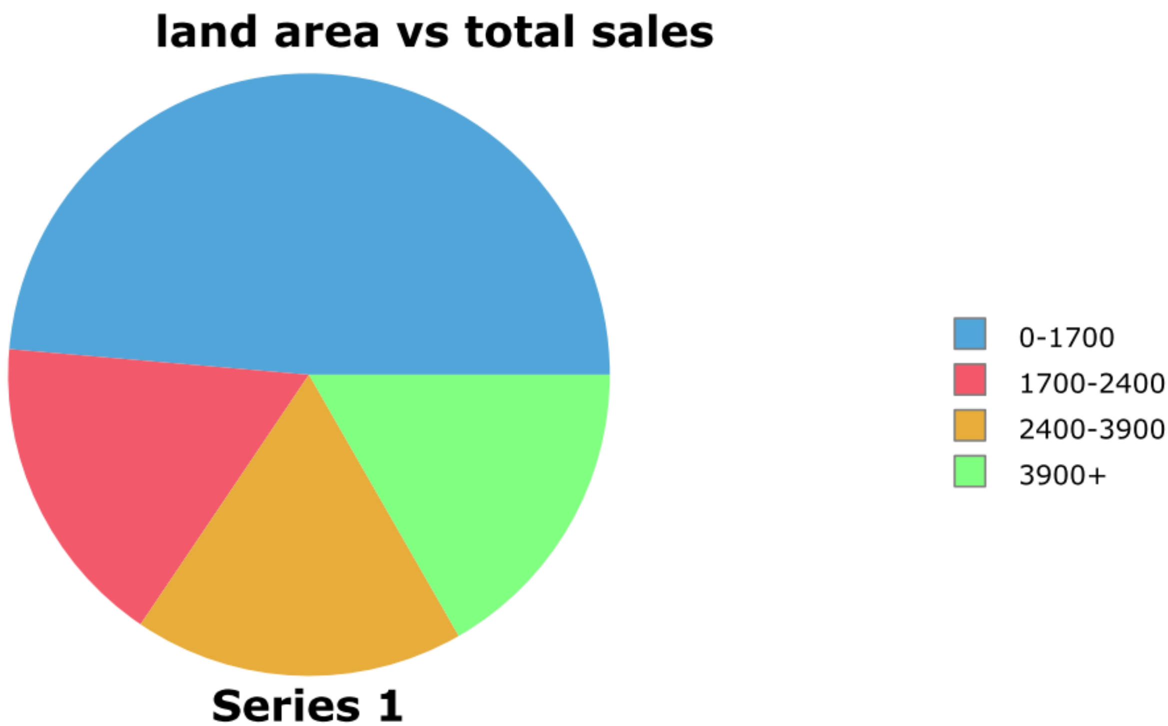


Business question3: Number of sales per year according to different binned land areas.

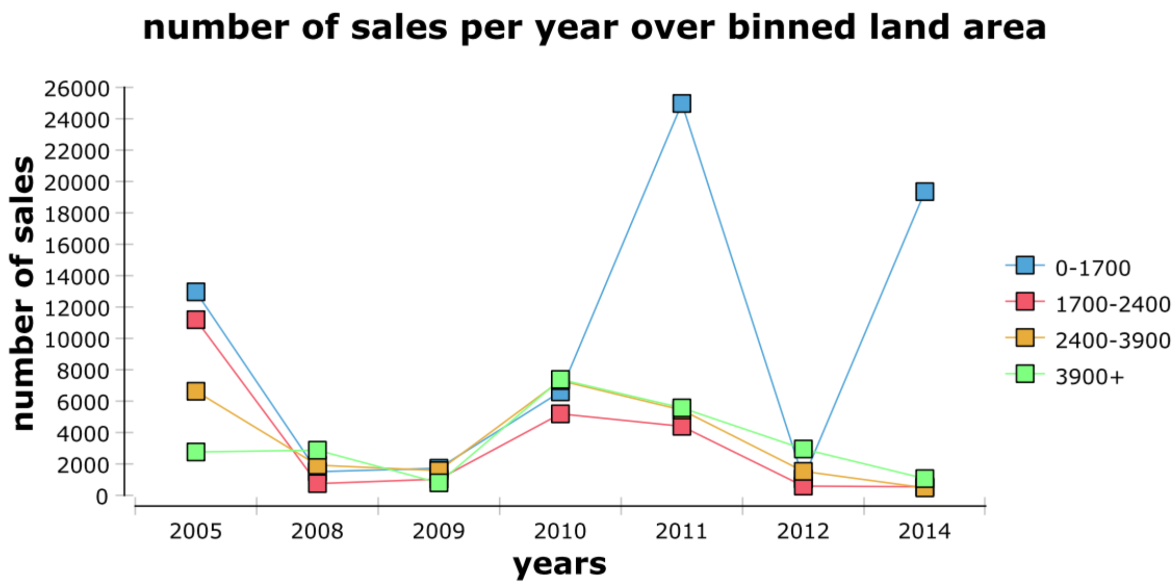
Here we use GroupBy to group year, land_squarefeet, then we manually aggregate commercial sales_price as count (same as before). After which we make bins for the land square feet with the help of numeric binner. Since the land square feet was continous values we made 4 bins(0-1700, 1700-2400, 2400-3900, 3900+). Then using Data to report we build a pie chart and line graph with number of sales in y-axis, years in x-axis and binned land areas as y-series grouping.



First representation (pie chart) shows that almost 50% of the total sales over the years are of properties with land square feet lower than 1700. The other three land square feet bins have similar portion of sales.

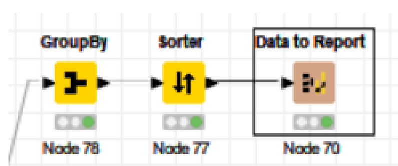


The second representation (line graph) proves that land square feet below the 1700 mark was almost always highest number of sales per year. It had a massive rise in the number of sales in the year 2011 and 2014. Whereas again 2012 had a drop in number of sales across all the categories

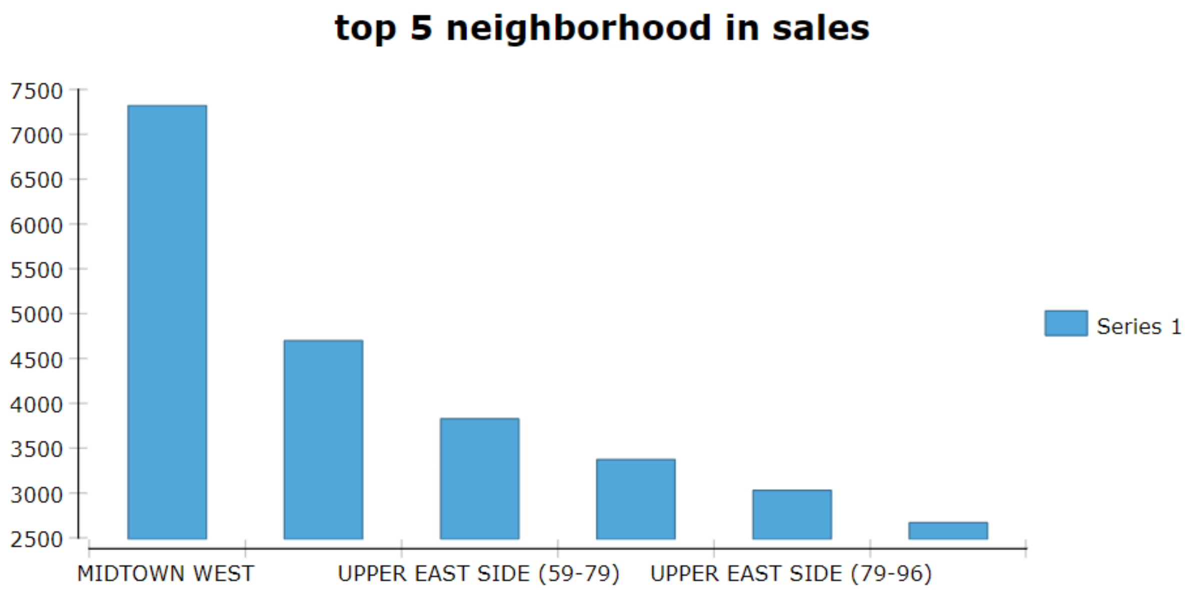


Business question4: To calculate the top 5 neighborhood with the greatest number of sales

To start with we will prepare the data, this can be achieved by using joiner nodes. To connect 2 tables at a time. Then we will deploy group by node to group the data by neighborhood, and count of sale. Our next step is to use sorter node to sort the table in ascending order to get the top neighborhood with highest sales. Next, we prune the rows after 6 rows. This allows us to isolate the top neighborhoods. Next, we will deploy reporting node and shift to BIRT tool. Then, insert bar chart, and assign x axis as borough and y axis as count of sales. The below image can be used to answer our business question.

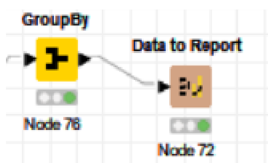


From the above graph we can analyze that MIDTOWN WEST has the highest sales. And BEDFORD STUYVESANT has the sixth highest sales.

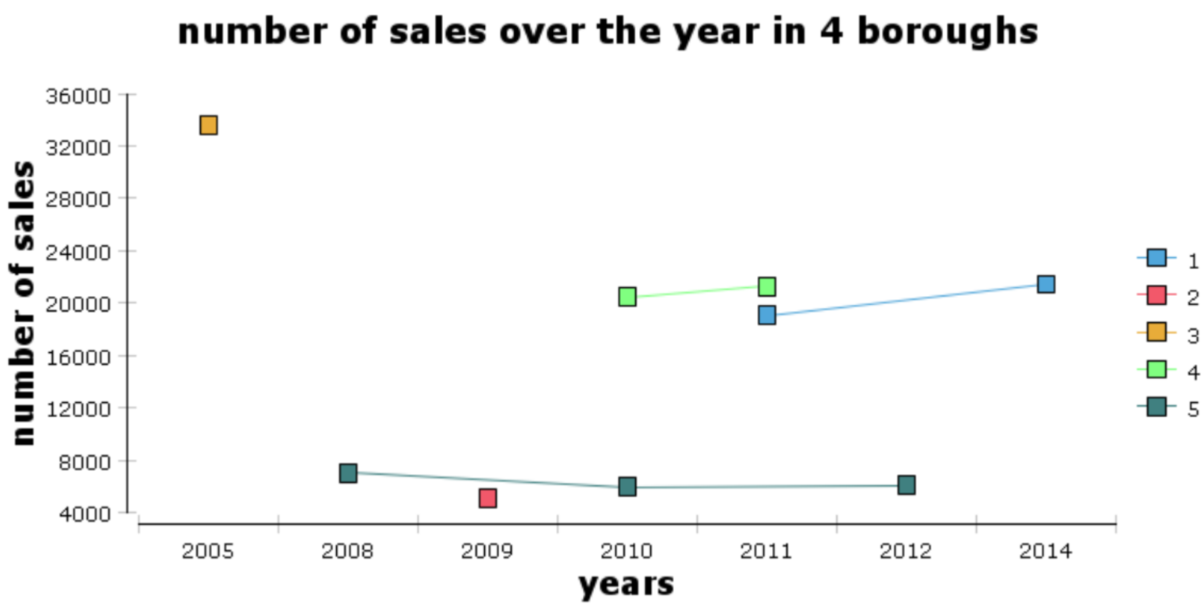


Business question5: Number of sales over the years in four boroughs

Here we use GroupBy to group year-wise and boroughs, then we manually aggregate commercial sales_price as count (same as before). Then using Data to report we build a bar graph with number of sales in y-axis, years in x-axis and numer of total units as y-series grouping (series1 as commercial units, series 2 as residential units) .



Here we do not have multiple year data for borough 2 and 3, hence they are represented only by a dot and cannot show a trend over the year. Whereas we can see that boroughs 4 and 5 had quite the same number of sales over the years. Borough 1 had an increase in the number of sales from 2011 to 2014.



Reflective analysis of using Knime for reporting:

While doing the reporting to answer the business questions, there were several factors to be carefully considered while filtering, sorting, and grouping. These were very repetitive work as it had to be done for each question and also for each dimension. There is no direct feature to select filter or sort and group by using any drop down or one click options like in other visualization tools. The number of charts/graphs available were also limited while using Knime (for example, it lacked histogram graph in Knime report). The chart preview was also not accurate to the real chart output and the data preview for column selection in charts was not complete and hence represented only a part of the data. The output of the report was not shown directly on the Knime software but was loaded on third party app in form of html file.

4.3 Reflective analysis of result in relational data warehouse vs Hadoop.

In data warehouse since we had used groupby and data cleaning methods, the total number of lines/rows loaded were lesser than that of Hadoop. Hence the number of sales parameters has changed in the two reports to some extent. The steps followed for reporting in case of data warehouse were quite time taking as we had to use domain calculator, value filters, joiners for each dimension to join it with the fact table with their respective ids. Whereas Hadoop implementation did not require this step as only one table was loaded and used for the reports.

Conclusions

Overall according to the steps, we went through for implementation of Hadoop and warehouse, I believe Hadoop was less complex as warehousing required creation and loading of each dimension (only in allocated schema design), this was a repetitive tedious task. Since warehousing also required to create multiple dimension tables and to link key of each dimension table to the fact table it was more complicated and time taking. As warehouse behaves like a data sink the data loaded in the warehouse must be transformed and pre-processed, hence at the user end they can directly run data reporting on pre-processed clean data. In very few cases, this cleaner and higher quality data can also be a disadvantage when some business questions or queries require data to be analyzed on the raw non processed data. Since data warehouse acts like a data sink it also cannot be reconfigured, whereas Hadoop can be reconfigured multiple times. Here are the overall advantages and disadvantages of each implementation:

Data Warehouse	**Hadoop**
A warehouse can't ingest information that has no allocated schema, as it uses schema pattern on write mechanism	Hadoop favors schema on read to process the data.
It uses schema-for-write logic to process the data.	It deals with schema-for-read logic to process the data.
Data warehouse can respond to critical workload with response in seconds through indices	Hadoop sacrifices on the availability. Hadoop fails to provide high performance to critical workload that requires response in minutes
In general, with data warehouse you have cleaner and high-quality data	It has partially less pre-processed cleaner data as Hadoop does not have data quality solutions.
Here we analyze only the structured and processed data	Here we can analyze any type of data (raw, structured, un-structured, semi-structured). The data we loaded was barely processed.
It is less agile.	It is highly agile
We cannot reconfigure, it is fixed configuration.	We can configure and reconfigure as needed.