MSIS 5683 BIG DATA ANALYTICS TECH

Grocery Market Sales Analysis for Business Decisions

Submitted by:

Avinash Kaur Atwal (A20094270)
Dileep Cherukuri (A11743469)
Ravali Musty (A20101635)
Reno Rajan Christy (A20082578)

Under the guidance of Dr. Bryan Hammer

Table of Contents

Executive Summary

Grocery Market primarily sells customer consumable food items along with other day to day non-perishable items. We intent to do sales analysis on the data for making better business decision, by allowing to identify factors that contribute to high value customers so as to allow business users to increase revenue and profitability. This can potentially help the shop owner and business managers to develop models and schemes for lead generation, possibly do a market basket analysis in the future and as such.

Statement of Scope

The scope of this project is to analyze grocery sales data for sales analysis. We are considering data for USA geographic region alone for the Project. Details on the different country data is excluded as almost all of the data belong to USA Region. Employee details are also excluded for now. We would be doing descriptive analysis and predictive analysis with PySpark to obtain following objective:

- The major factors that contribute to high value purchases from shop by cost and volume per customer.
- Provide with insights on how the business is performing in best performing region with respect to the sales profitability

Beneficiaries

Shop inventory Managers: By being able to customize the inventory at the shop at any time

Marketing Managers: By being able to identify the relevant fast-moving items for better trend identification and marketability to high value customers.

Data Dictionary

Attribute Name	**Description**	**Data Type**	**Source**
Sales ID	Primark key for each sales transaction	Integer	https://www.kaggle.com/marcinex1423/salesdb-grocery
Sales Person ID	Unique ID for the Sales person	Integer	https://www.kaggle.com/marcinex1423/salesdb-grocery
Customer ID	Unique Customer ID number	Integer	https://www.kaggle.com/marcinex1423/salesdb-grocery
Quantity	Quantity bought by customer	Integer	https://www.kaggle.com/marcinex1423/salesdb-grocery
Sales Date	Date of sale	Timestamp	https://www.kaggle.com/marcinex1423/salesdb-grocery
Transaction Number	Transaction number	String	https://www.kaggle.com/marcinex1423/salesdb-grocery

	involved with the sale		
Dis-count	Price off provided on the product	Double	https://www.kaggle.com/marcinex1423/salesdb-grocery
Product ID	Unique ID of the product	Integer	https://www.kaggle.com/marcinex1423/salesdb-grocery
Product Name	Name of Product	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Price	Price of each product	Float	https://www.kaggle.com/marcinex1423/salesdb-grocery
Category ID	Key for Category Table	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Class	Class of product that product belong to	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Modify Date	Date when the product was modified in the system	String	https://www.kaggle.com/marcinex1423/salesdb-grocery

Resistant	Whether product as Durable or weak or not applicable	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Is Allergic	Allergic nature of the product	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Vitality Days	N/A	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Total Price	Quantity of product multiplied by its price	Float	https://www.kaggle.com/marcinex1423/salesdb-grocery
First Name	Customer Name First Initial	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Middle Initial	Customer Name Middle Initial	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Last Name	Customer Name Last Initial	String	https://www.kaggle.com/marcinex1423/salesdb-grocery

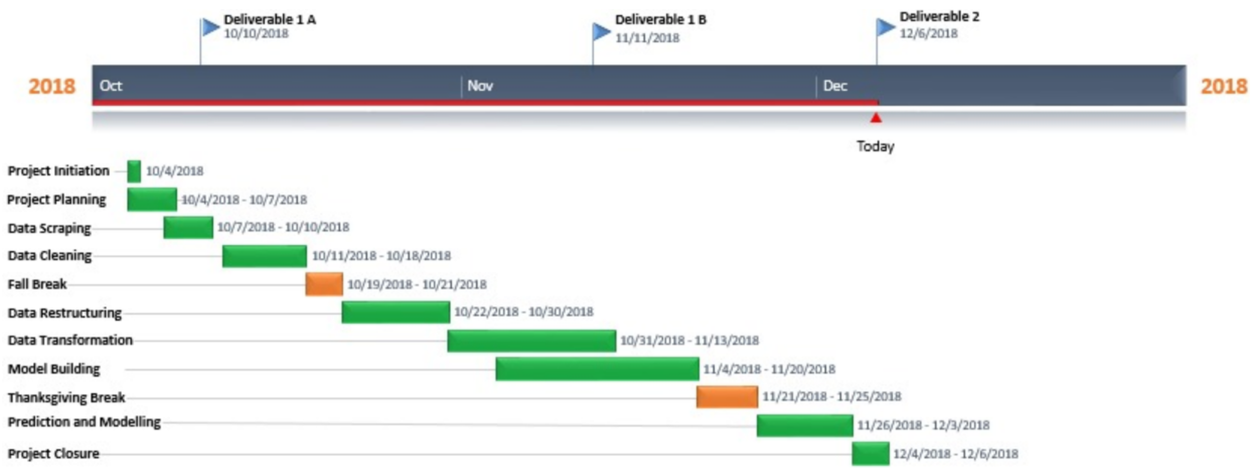
City ID	ID number unique to each city	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Address	Address of Customer	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
City Name	Name of city the store is located in	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Zip Code	Zip code number corresponding to location of store	String	https://www.kaggle.com/marcinex1423/salesdb-grocery
Category Name	Name of category to which product belong to	String	https://www.kaggle.com/marcinex1423/salesdb-grocery

Project Schedule

We have planned to complete this project in 9 weeks (from October 5th, 2018 to December 6th, 2018). Deliverable oriented breakdown of the activities can be seen in the Gantt chart which is encompassing all the task with their timeline.

Legend details

- The activities in green are the tasks which have been completed.
- The one in yellow is the task which is in progress.
- The tasks in blue are yet to be covered.
- We have also considered the breaks (shown in orange).

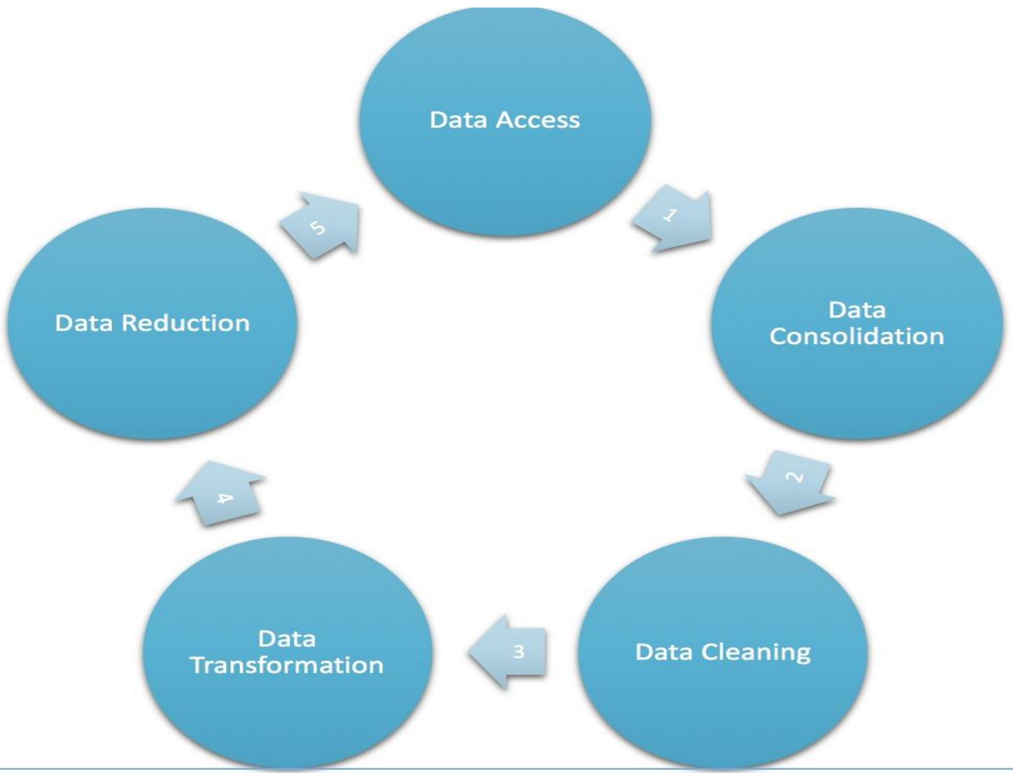


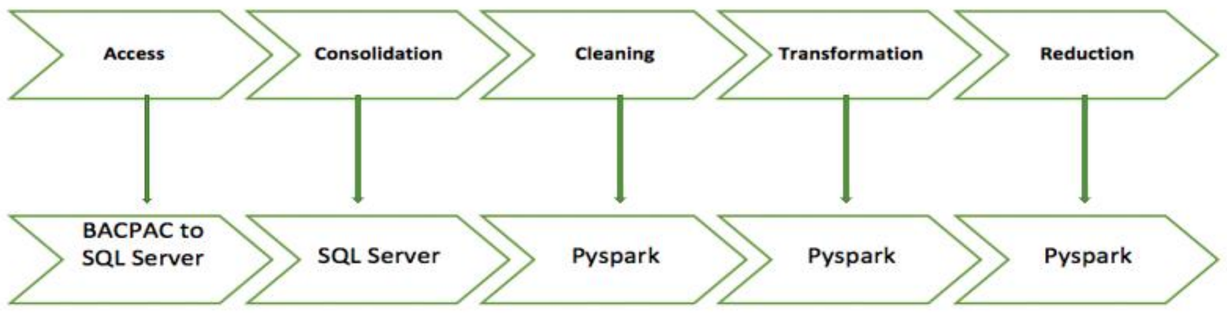
Data of Interest

Grocery Sales Data Analysis specific to United States Regio

Data Preparation:

While the Data access and consolidation was accomplished using SQL server, the remaining data cleaning, transformation and reduction was accomplished in PySpark.





Data Access

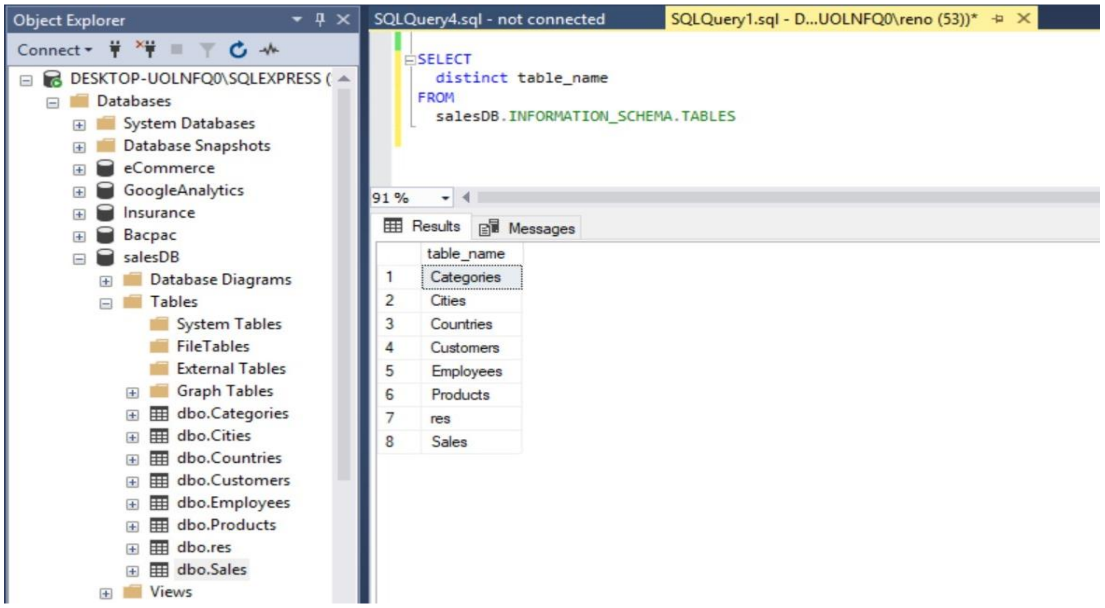
For this project we have taken the dataset from the Kaggle Website. The dataset describes about the sales of different products in a grocery store. It has total of 7 tables containing information about the store's Customers, Cities, Employees, Countries, Products, Categories and Sales. The dataset is a csv file of 463 MB when downloaded from the website. There are totally 41 columns. The location of the dataset is: https://www.kaggle.com/marcinex1423/salesdb-grocery

Table Name	**Number of Columns**
Sales	9
Products	9
Customers	6
Cities	4
Countries	3
Employees	8
Categories	2

Data Consolidation

Since the dataset had 7 tables, we had to consolidate them and connect the tables to proceed further. For the consolidation purpose, we used Microsoft SQL Server Management Studio. Firstly, we downloaded the BACPAC file extension from the website.

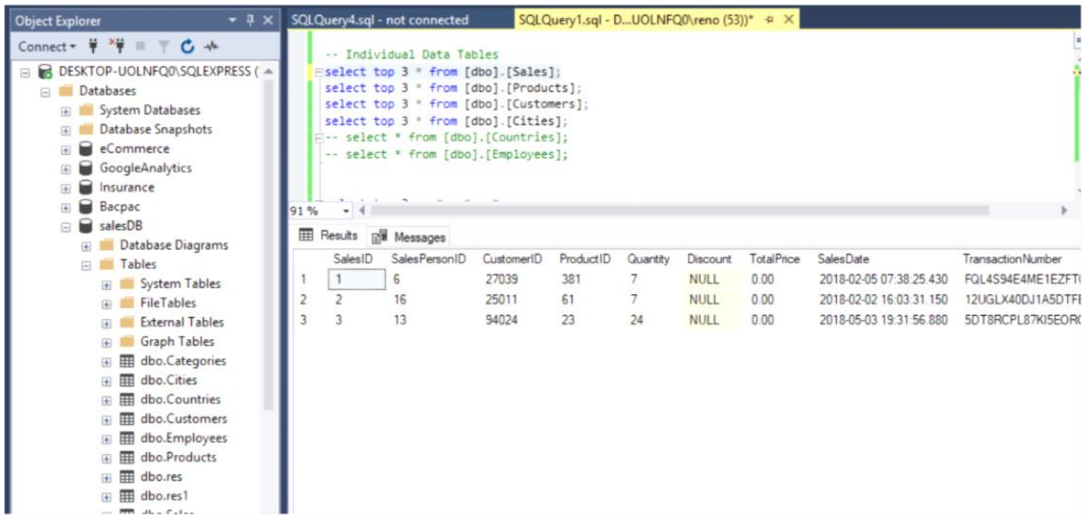
Note: A BACPAC file is a ZIP file with an extension of BACPAC containing the metadata and data from a SQL Server database. A BACPAC file can be stored in Azure blob storage or in local storage in an on-premises location and later imported back into Azure SQL Database or into a SQL Server on-premises installation. Using the BACPAC file, we imported the dataset into SQL Server. The below screen screenshot shows all the tables in the dataset.



```
SELECT
  distinct table_name
FROM
  salesDB.INFORMATION_SCHEMA.TABLES
```

	table_name
1	Categories
2	Cities
3	Countries
4	Customers
5	Employees
6	Products
7	res
8	Sales

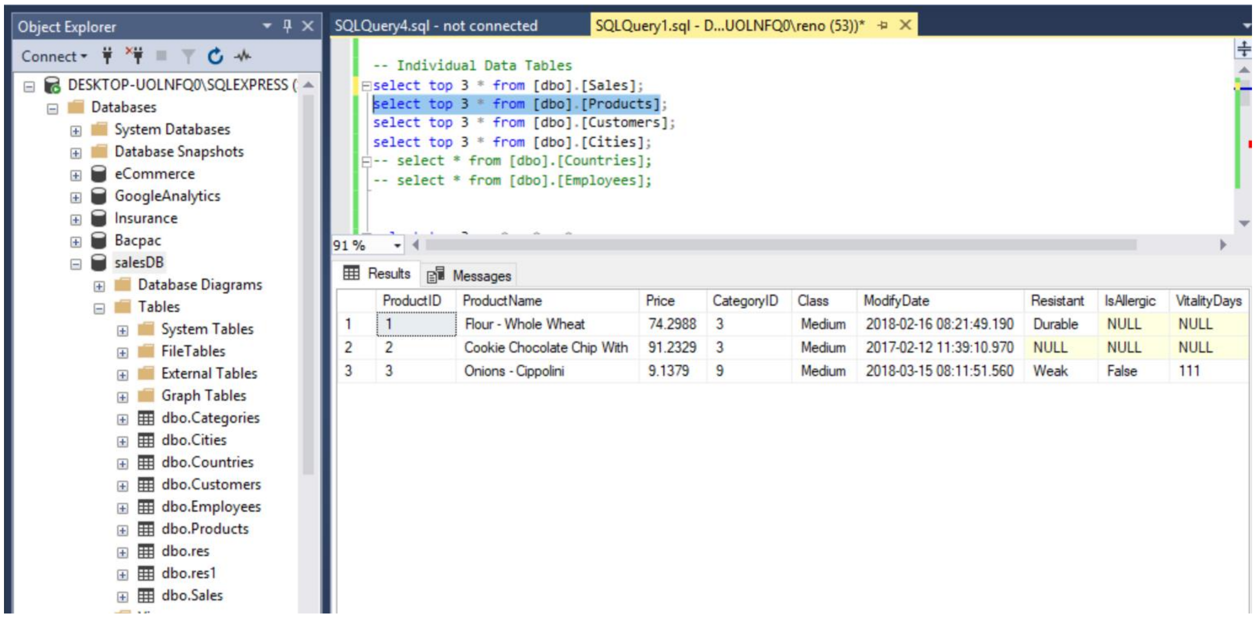
First, we analyzed the columns in each table. We selected top 3 columns of every table to reduce the retrieval time. Below Screenshot gives the Top 3 rows from Sales table:



```
-- Individual Data Tables
select top 3 * from [dbo].[Sales];
select top 3 * from [dbo].[Products];
select top 3 * from [dbo].[Customers];
select top 3 * from [dbo].[Cities];
-- select * from [dbo].[Countries];
-- select * from [dbo].[Employees];
```

	SalesID	SalesPersonID	CustomerID	ProductID	Quantity	Discount	TotalPrice	SalesDate	TransactionNumber
1	1	6	27039	381	7	NULL	0.00	2018-02-05 07:38:25.430	FQL4S94E4ME1EZFT
2	2	16	25011	61	7	NULL	0.00	2018-02-02 16:03:31.150	12UGLX40DJ1A5DTF
3	3	13	94024	23	24	NULL	0.00	2018-05-03 19:31:56.880	5DT8RCPL87KI5EOR

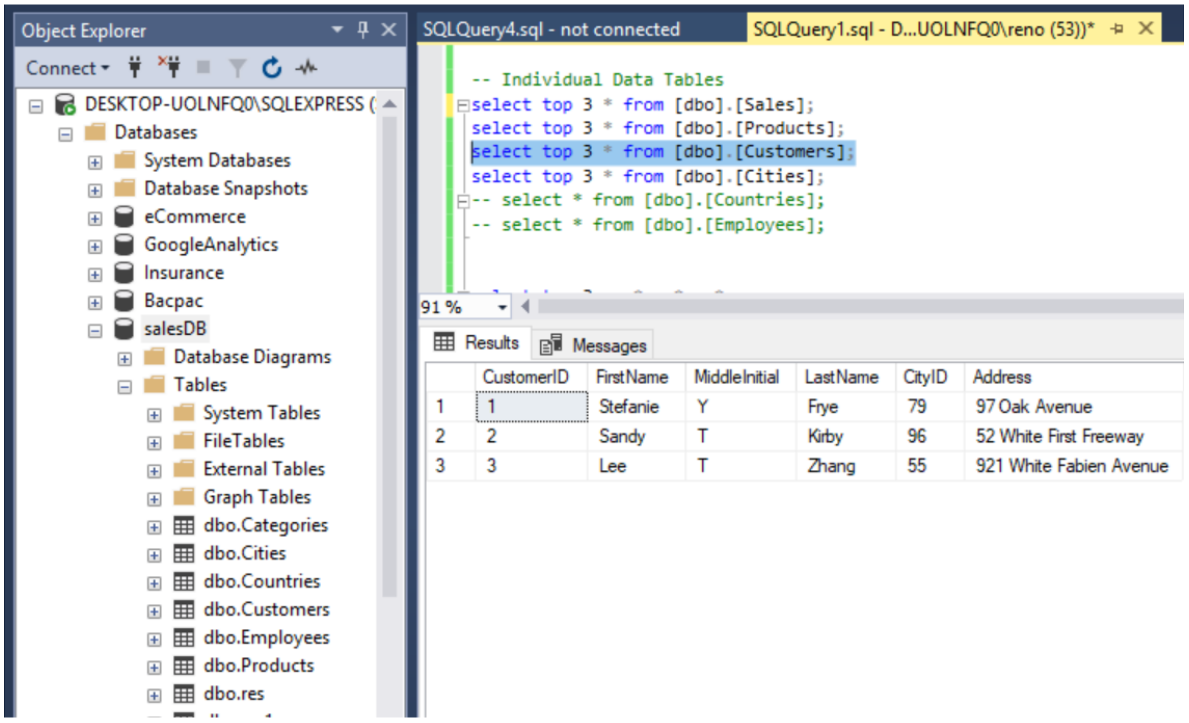
Below Screenshot gives the Top 3 rows from Products table:



```
-- Individual Data Tables
select top 3 * from [dbo].[Sales];
select top 3 * from [dbo].[Products];
select top 3 * from [dbo].[Customers];
select top 3 * from [dbo].[Cities];
-- select * from [dbo].[Countries];
-- select * from [dbo].[Employees];
```

	ProductID	ProductName	Price	CategoryID	Class	ModifyDate	Resistant	IsAllergic	VitalityDays
1	1	Flour - Whole Wheat	74.2988	3	Medium	2018-02-16 08:21:49.190	Durable	NULL	NULL
2	2	Cookie Chocolate Chip With	91.2329	3	Medium	2017-02-12 11:39:10.970	NULL	NULL	NULL
3	3	Onions - Cippolini	9.1379	9	Medium	2018-03-15 08:11:51.560	Weak	False	111

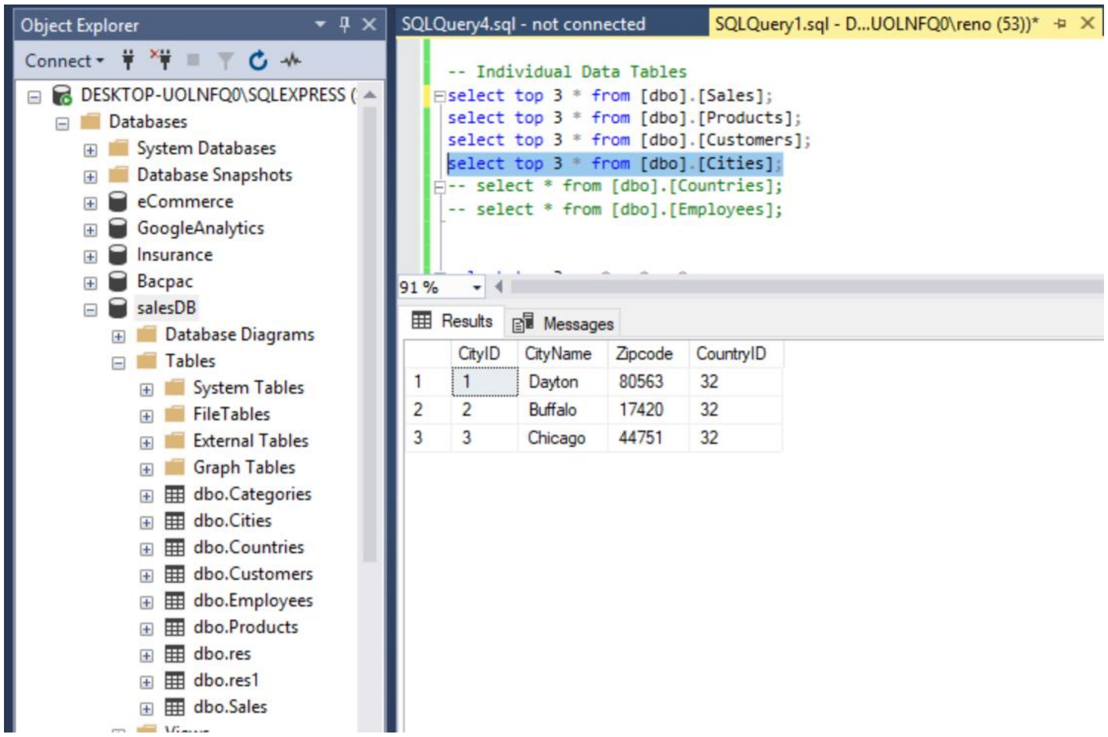
Below Screenshot gives the Top 3 rows from Customers table:



```
-- Individual Data Tables
select top 3 * from [dbo].[Sales];
select top 3 * from [dbo].[Products];
select top 3 * from [dbo].[Customers];
select top 3 * from [dbo].[Cities];
-- select * from [dbo].[Countries];
-- select * from [dbo].[Employees];
```

	CustomerID	FirstName	MiddleInitial	LastName	CityID	Address
1	1	Stefanie	Y	Frye	79	97 Oak Avenue
2	2	Sandy	T	Kirby	96	52 White First Freeway
3	3	Lee	T	Zhang	55	921 White Fabien Avenue

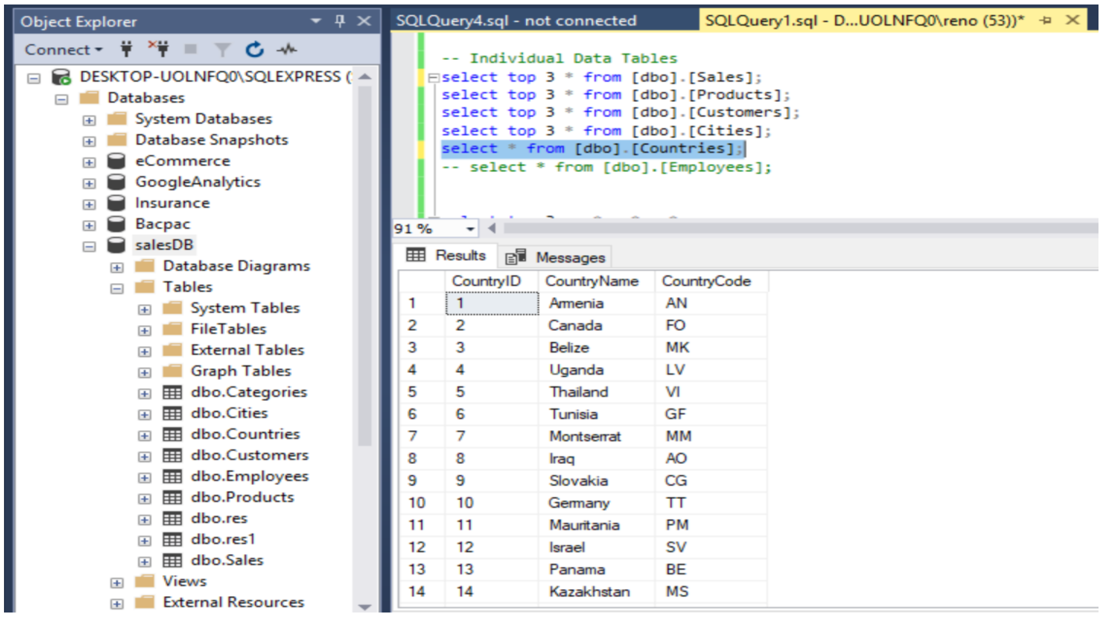
Below Screenshot gives the Top 3 rows from Cities table:



```
-- Individual Data Tables
select top 3 * from [dbo].[Sales];
select top 3 * from [dbo].[Products];
select top 3 * from [dbo].[Customers];
select top 3 * from [dbo].[Cities];
-- select * from [dbo].[Countries];
-- select * from [dbo].[Employees];
```

	CityID	CityName	Zipcode	CountryID
1	1	Dayton	80563	32
2	2	Buffalo	17420	32
3	3	Chicago	44751	32

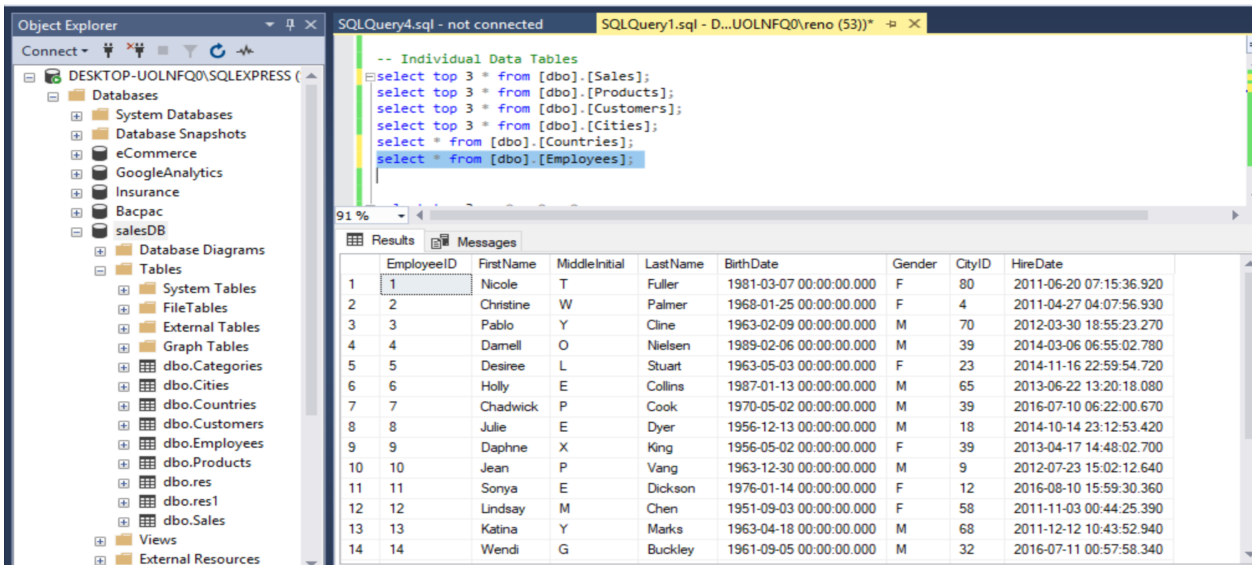
Below Screenshot gives the rows from Countries table:



```
-- Individual Data Tables
select top 3 * from [dbo].[Sales];
select top 3 * from [dbo].[Products];
select top 3 * from [dbo].[Customers];
select top 3 * from [dbo].[Cities];
select * from [dbo].[Countries];
-- select * from [dbo].[Employees];
```

	CountryID	CountryName	CountryCode
1	1	Armenia	AN
2	2	Canada	FO
3	3	Belize	MK
4	4	Uganda	LV
5	5	Thailand	VI
6	6	Tunisia	GF
7	7	Montserrat	MM
8	8	Iraq	AO
9	9	Slovakia	CG
10	10	Germany	TT
11	11	Mauritania	PM
12	12	Israel	SV
13	13	Panama	BE
14	14	Kazakhstan	MS

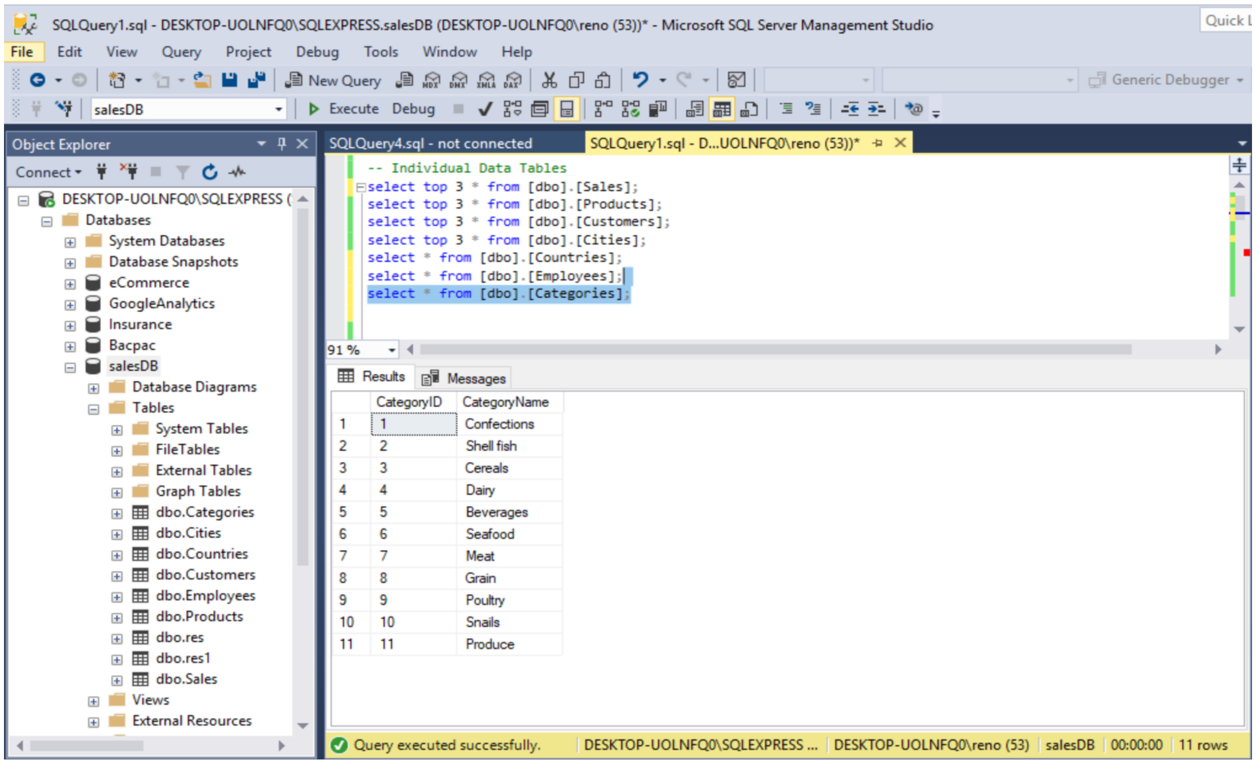
Below Screenshot gives the rows from Employees table:



```
-- Individual Data Tables
select top 3 * from [dbo].[Sales];
select top 3 * from [dbo].[Products];
select top 3 * from [dbo].[Customers];
select top 3 * from [dbo].[Cities];
select * from [dbo].[Countries];
select * from [dbo].[Employees];
```

	EmployeeID	FirstName	MiddleInitial	LastName	BirthDate	Gender	CityID	HireDate
1	1	Nicole	T	Fuller	1981-03-07 00:00:00.000	F	80	2011-06-20 07:15:36.920
2	2	Christine	W	Palmer	1968-01-25 00:00:00.000	F	4	2011-04-27 04:07:56.930
3	3	Pablo	Y	Cline	1963-02-09 00:00:00.000	M	70	2012-03-30 18:55:23.270
4	4	Darnell	O	Nielsen	1989-02-06 00:00:00.000	M	39	2014-03-06 06:55:02.780
5	5	Desiree	L	Stuart	1963-05-03 00:00:00.000	F	23	2014-11-16 22:59:54.720
6	6	Holly	E	Collins	1987-01-13 00:00:00.000	M	65	2013-06-22 13:20:18.080
7	7	Chadwick	P	Cook	1970-05-02 00:00:00.000	M	39	2016-07-10 06:22:00.670
8	8	Julie	E	Dyer	1956-12-13 00:00:00.000	M	18	2014-10-14 23:12:53.420
9	9	Daphne	X	King	1956-05-02 00:00:00.000	F	39	2013-04-17 14:48:02.700
10	10	Jean	P	Vang	1963-12-30 00:00:00.000	M	9	2012-07-23 15:02:12.640
11	11	Sonya	E	Dickson	1976-01-14 00:00:00.000	F	12	2016-08-10 15:59:30.360
12	12	Lindsay	M	Chen	1951-09-03 00:00:00.000	F	58	2011-11-03 00:44:25.390
13	13	Katina	Y	Marks	1963-04-18 00:00:00.000	M	68	2011-12-12 10:43:52.940
14	14	Wendi	G	Buckley	1961-09-05 00:00:00.000	M	32	2016-07-11 00:57:58.340

Below Screenshot gives the rows from Categories table:



```
-- Individual Data Tables
select top 3 * from [dbo].[Sales];
select top 3 * from [dbo].[Products];
select top 3 * from [dbo].[Customers];
select top 3 * from [dbo].[Cities];
select * from [dbo].[Countries];
select * from [dbo].[Employees];
select * from [dbo].[Categories];
```

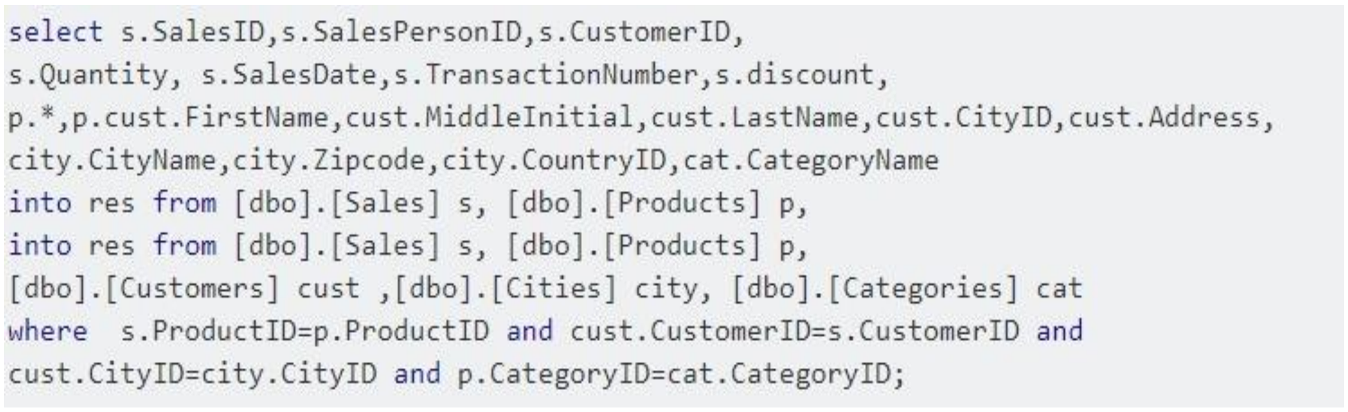
	CategoryID	CategoryName
1	1	Confections
2	2	Shell fish
3	3	Cereals
4	4	Dairy
5	5	Beverages
6	6	Seafood
7	7	Meat
8	8	Grain
9	9	Poultry
10	10	Snails
11	11	Produce

```
Query executed successfully.   DESKTOP-UOLNFQ0\SQLEXPRESS ...   DESKTOP-UOLNFQ0\reno (53)   salesDB   00:00:00   11 rows
```

After analyzing the columns in tables, we ran an SQL query to connect the different tables based on their primary keys, utilizing the INNER JOIN to achieve that.

<u>Note:</u> We have not included Employees and Countries Table in our analysis, as they have either redundant data columns or they do not add value to the analysis.

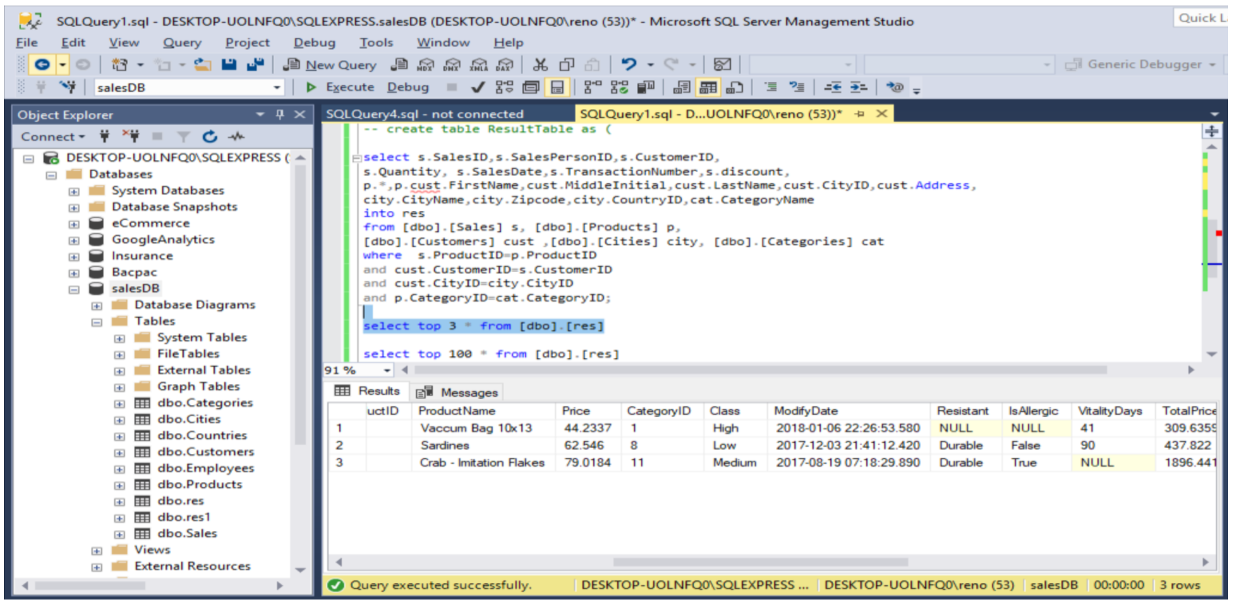
The Query for consolidation used is:



```
select s.SalesID,s.SalesPersonID,s.CustomerID,
s.Quantity, s.SalesDate,s.TransactionNumber,s.discount,
p.*,p.cust.FirstName,cust.MiddleInitial,cust.LastName,cust.CityID,cust.Address,
city.CityName,city.Zipcode,city.CountryID,cat.CategoryName
into res from [dbo].[Sales] s, [dbo].[Products] p,
into res from [dbo].[Sales] s, [dbo].[Products] p,
[dbo].[Customers] cust ,[dbo].[Cities] city, [dbo].[Categories] cat
where  s.ProductID=p.ProductID and cust.CustomerID=s.CustomerID and
cust.CityID=city.CityID and p.CategoryID=cat.CategoryID;
```

In the above query, we joined Sales, Products, Customer, Cities and Category tables using inner join and loaded the extracted the results into another table called ‘res’.

Below screenshot pulls out the first 3 rows from the consolidated data source ‘res’.

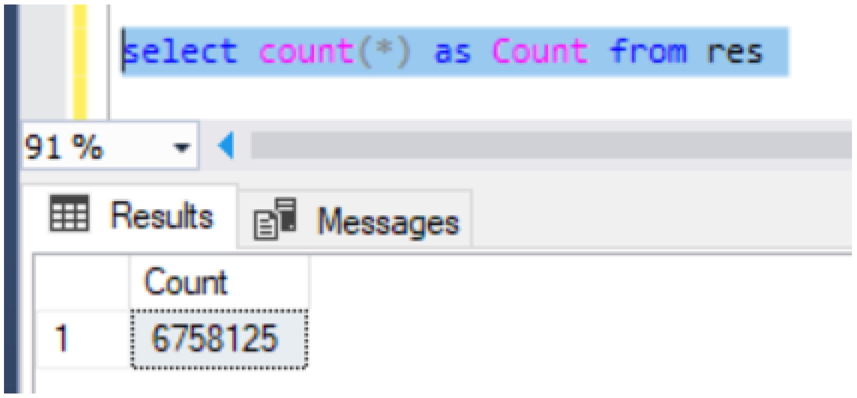


```sql
-- create table ResultTable as (

select s.SalesID,s.SalesPersonID,s.CustomerID,
s.Quantity, s.SalesDate,s.TransactionNumber,s.discount,
p.*,p.cust.FirstName,cust.MiddleInitial,cust.LastName,cust.CityID,cust.Address,
city.CityName,city.Zipcode,city.CountryID,cat.CategoryName
into res
from [dbo].[Sales] s, [dbo].[Products] p,
[dbo].[Customers] cust ,[dbo].[Cities] city, [dbo].[Categories] cat
where  s.ProductID=p.ProductID
and cust.CustomerID=s.CustomerID
and cust.CityID=city.CityID
and p.CategoryID=cat.CategoryID;

select top 3 * from [dbo].[res]

select top 100 * from [dbo].[res]
```

	uctID	ProductName	Price	CategoryID	Class	ModifyDate	Resistant	IsAllergic	VitalityDays	TotalPrice
1		Vaccum Bag 10x13	44.2337	1	High	2018-01-06 22:26:53.580	NULL	NULL	41	309.6359
2		Sardines	62.546	8	Low	2017-12-03 21:41:12.420	Durable	False	90	437.822
3		Crab - Imitation Flakes	79.0184	11	Medium	2017-08-19 07:18:29.890	Durable	True	NULL	1896.441

```
Query executed successfully.   DESKTOP-UOLNFQ0\SQLEXPRESS ...   DESKTOP-UOLNFQ0\reno (53)   salesDB   00:00:00   3 rows
```

After consolidation the total row count is 6758125.



```sql
select count(*) as Count from res
```

	Count
1	6758125

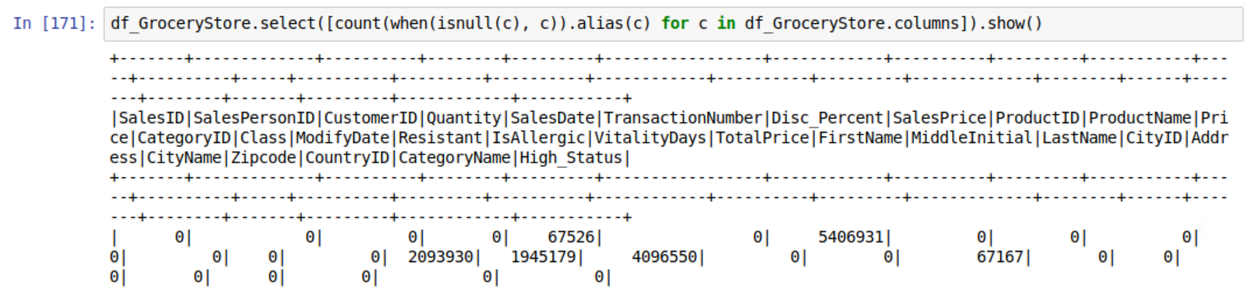
Note: After the consolidation there are 26 columns in total.

Note: Once the data is consolidated, it is exported to a flat file using a comma delimiter.

Data Cleaning

Missing Values Analysis

After obtaining the consolidated data, pre-processing is required to obtain a near accurate model. As a part of pre-processing, the first step is to clean the data for any missing values and erroneous values. The below code snippet identifies null values from the columns


```
In [171]: df_GroceryStore.select([count(when(isnull(c), c)).alias(c) for c in df_GroceryStore.columns]).show()
```

```
+-------+-------------+----------+--------+---------+-----------------+------------+----------+---------+-----------+---
--+----------+-----+----------+---------+----------+------------+----------+---------+-------------+--------+------+----
---+--------+-------+---------+------------+-----------+
|SalesID|SalesPersonID|CustomerID|Quantity|SalesDate|TransactionNumber|Disc_Percent|SalesPrice|ProductID|ProductName|Pri
ce|CategoryID|Class|ModifyDate|Resistant|IsAllergic|VitalityDays|TotalPrice|FirstName|MiddleInitial|LastName|CityID|Addr
ess|CityName|Zipcode|CountryID|CategoryName|High_Status|
+-------+-------------+----------+--------+---------+-----------------+------------+----------+---------+-----------+---
--+----------+-----+----------+---------+----------+------------+----------+---------+-------------+--------+------+----
---+--------+-------+---------+------------+-----------+
|      0|            0|         0|       0|    67526|                0|     5406931|         0|        0|          0|
0|         0|    0|         0|  2093930|   1945179|     4096550|         0|        0|        67167|       0|     0|
0|       0|      0|        0|           0|          0|
```

Variables identified for Data imputation:

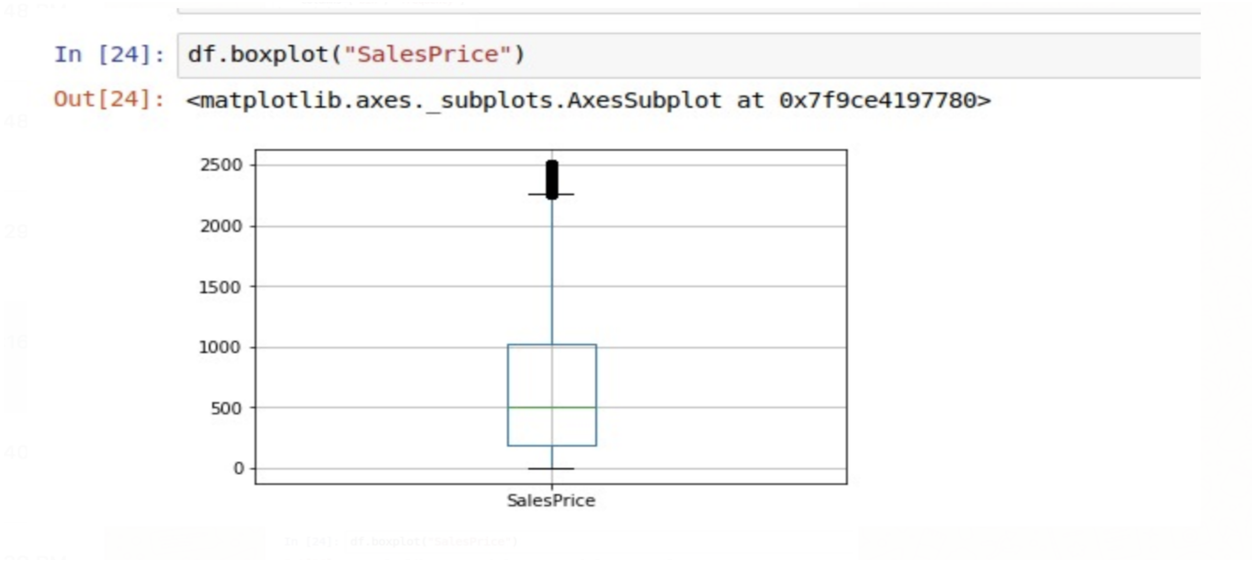
SalesDate (Transaction Date), Discount Percentage and Customer Name.

Irrelevant variables as identified through our business questions:

Resistant, IsAllergic and VitalityDays.

Column Name	**Number of missing rows**	**Cleaning Action**
Sales Date	67526	Impute / Filter
Disc_Percent	5406931	Imputation
Resistant	2093930	Removed in data reduction
Is allergic	1945179	Removed in data reduction
VitalityDays	4096550	Removed in data reduction
Customer Name	67167	Transformation

Outlier Analysis

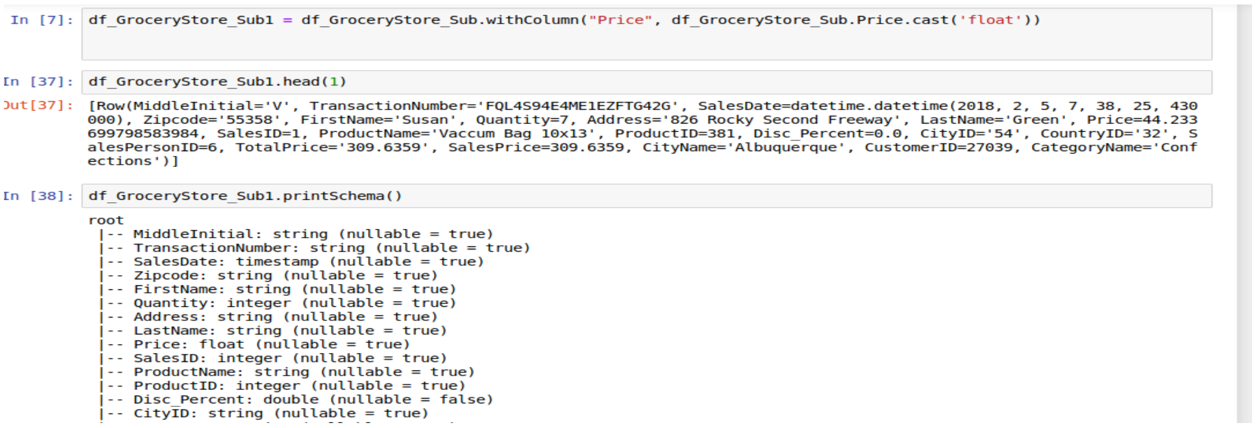


```
In [24]: df.boxplot("SalesPrice")
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9ce4197780>
```

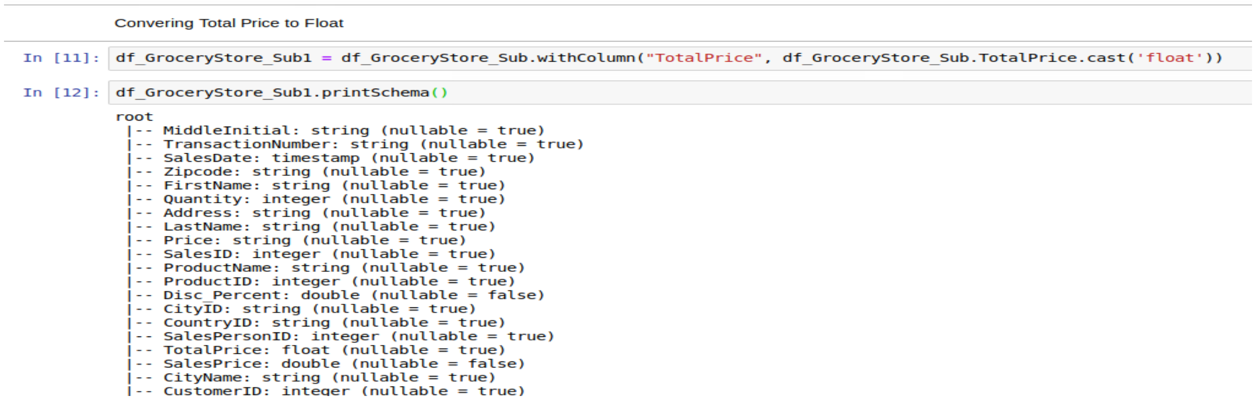
The above box plot of the Sales Price shows few outliers. But it is explainable that there are fewer transactions that happen at a grocery store that are billed above $2000 dollars. Since identifying the transactions that bring greater revenue at one billing cycle is part of our analysis, removing the outliers does not seem a viable option at this point of our analysis. So, until further analysis, we choose to not remove any outliers.

Adjustments to data Types

As part of cleaning, variables with incorrect data type are converted to represent the accurate data type. In the given ‘Grocery Store’ dataset, the variables ‘Price’ and ‘Total Price’ have data type as string, as opposed to float. Therefore, below code snippet is run to change their data types to float.



```
In [7]: df_GroceryStore_Sub1 = df_GroceryStore_Sub.withColumn("Price", df_GroceryStore_Sub.Price.cast('float'))

In [37]: df_GroceryStore_Sub1.head(1)

Out[37]: [Row(MiddleInitial='V', TransactionNumber='FQL4S94E4ME1EZFTG42G', SalesDate=datetime.datetime(2018, 2, 5, 7, 38, 25, 430
         000), Zipcode='55358', FirstName='Susan', Quantity=7, Address='826 Rocky Second Freeway', LastName='Green', Price=44.233
         699798583984, SalesID=1, ProductName='Vaccum Bag 10x13', ProductID=381, Disc_Percent=0.0, CityID='54', CountryID='32', S
         alesPersonID=6, TotalPrice='309.6359', SalesPrice=309.6359, CityName='Albuquerque', CustomerID=27039, CategoryName='Conf
         ections')]

In [38]: df_GroceryStore_Sub1.printSchema()

         root
          |-- MiddleInitial: string (nullable = true)
          |-- TransactionNumber: string (nullable = true)
          |-- SalesDate: timestamp (nullable = true)
          |-- Zipcode: string (nullable = true)
          |-- FirstName: string (nullable = true)
          |-- Quantity: integer (nullable = true)
          |-- Address: string (nullable = true)
          |-- LastName: string (nullable = true)
          |-- Price: float (nullable = true)
          |-- SalesID: integer (nullable = true)
          |-- ProductName: string (nullable = true)
          |-- ProductID: integer (nullable = true)
          |-- Disc_Percent: double (nullable = false)
          |-- CityID: string (nullable = true)
```



Convering Total Price to Float

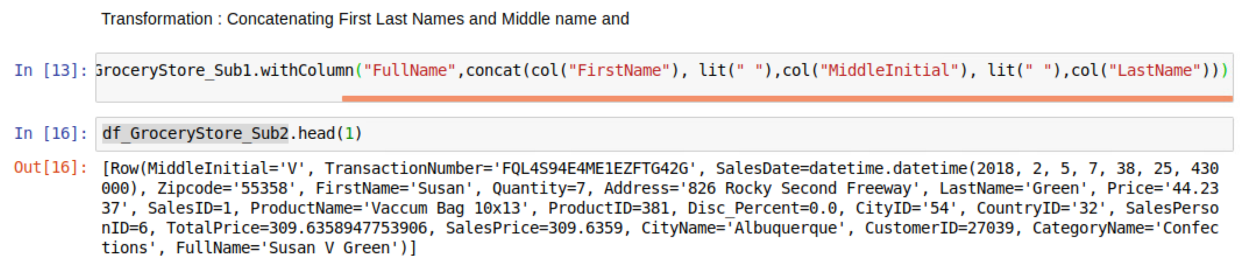
```
In [11]: df_GroceryStore_Sub1 = df_GroceryStore_Sub.withColumn("TotalPrice", df_GroceryStore_Sub.TotalPrice.cast('float'))

In [12]: df_GroceryStore_Sub1.printSchema()

         root
          |-- MiddleInitial: string (nullable = true)
          |-- TransactionNumber: string (nullable = true)
          |-- SalesDate: timestamp (nullable = true)
          |-- Zipcode: string (nullable = true)
          |-- FirstName: string (nullable = true)
          |-- Quantity: integer (nullable = true)
          |-- Address: string (nullable = true)
          |-- LastName: string (nullable = true)
          |-- Price: string (nullable = true)
          |-- SalesID: integer (nullable = true)
          |-- ProductName: string (nullable = true)
          |-- ProductID: integer (nullable = true)
          |-- Disc_Percent: double (nullable = false)
          |-- CityID: string (nullable = true)
          |-- CountryID: string (nullable = true)
          |-- SalesPersonID: integer (nullable = true)
          |-- TotalPrice: float (nullable = true)
          |-- SalesPrice: double (nullable = false)
          |-- CityName: string (nullable = true)
          |-- CustomerID: integer (nullable = true)
```

Data Transformation

After cleansing the data, we need to check for any transformations within the variables to enhance their usability. Below are the prior checks done to proceed on the transformation steps.

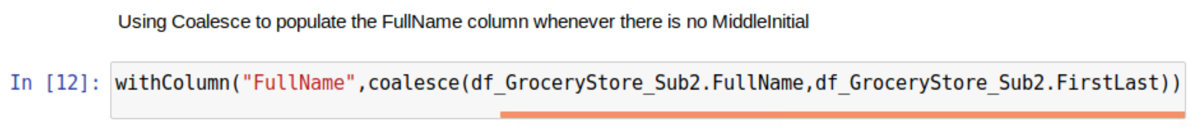
Constructing new attributes

First and Last Name is concatenated to represent a single Variable ‘Full Name’. Below code snippet does the transformation.


Transformation : Concatenating First Last Names and Middle name and

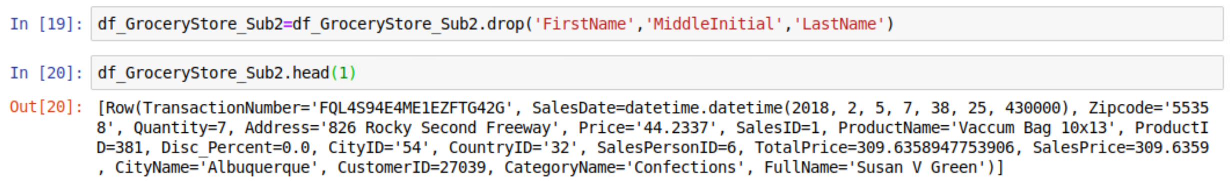
```
In [13]: GroceryStore_Sub1.withColumn("FullName",concat(col("FirstName"), lit(" "),col("MiddleInitial"), lit(" "),col("LastName")))

In [16]: df_GroceryStore_Sub2.head(1)

Out[16]: [Row(MiddleInitial='V', TransactionNumber='FQL4S94E4ME1EZFTG42G', SalesDate=datetime.datetime(2018, 2, 5, 7, 38, 25, 430
         000), Zipcode='55358', FirstName='Susan', Quantity=7, Address='826 Rocky Second Freeway', LastName='Green', Price='44.23
         37', SalesID=1, ProductName='Vaccum Bag 10x13', ProductID=381, Disc_Percent=0.0, CityID='54', CountryID='32', SalesPerso
         nID=6, TotalPrice=309.6358947753906, SalesPrice=309.6359, CityName='Albuquerque', CustomerID=27039, CategoryName='Confec
         tions', FullName='Susan V Green')]
```

Since either of the Middle/Last/First name is being seen as null in some of the cases we decided to coalesce Full name with combination of First Last wherever Middle name is not present.


Using Coalesce to populate the FullName column whenever there is no MiddleInitial

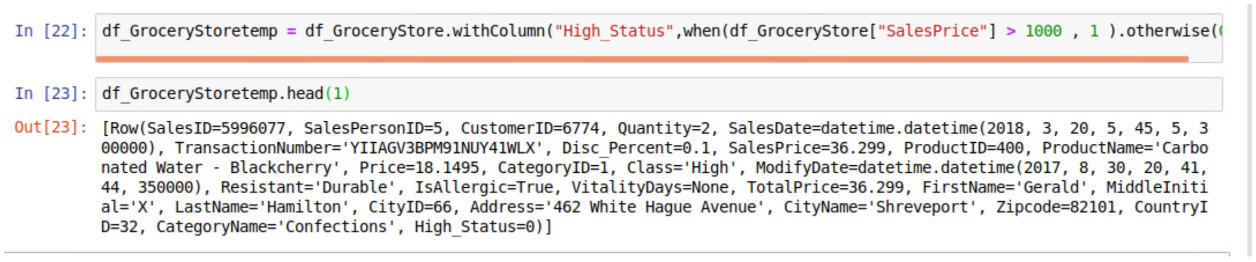
```
In [12]: withColumn("FullName",coalesce(df_GroceryStore_Sub2.FullName,df_GroceryStore_Sub2.FirstLast))
```

Once the new attribute 'Full Name' is created, the columns, 'First Name', 'Middle Name' and 'Last Name' can be deleted to reduce the number of extraneous columns. Below code snippet shows the above the transformation.



```
In [19]: df_GroceryStore_Sub2=df_GroceryStore_Sub2.drop('FirstName','MiddleInitial','LastName')

In [20]: df_GroceryStore_Sub2.head(1)

Out[20]: [Row(TransactionNumber='FQL4S94E4ME1EZFTG42G', SalesDate=datetime.datetime(2018, 2, 5, 7, 38, 25, 430000), Zipcode='5535
         8', Quantity=7, Address='826 Rocky Second Freeway', Price='44.2337', SalesID=1, ProductName='Vaccum Bag 10x13', ProductI
         D=381, Disc_Percent=0.0, CityID='54', CountryID='32', SalesPersonID=6, TotalPrice=309.6358947753906, SalesPrice=309.6359
         , CityName='Albuquerque', CustomerID=27039, CategoryName='Confections', FullName='Susan V Green')]
```

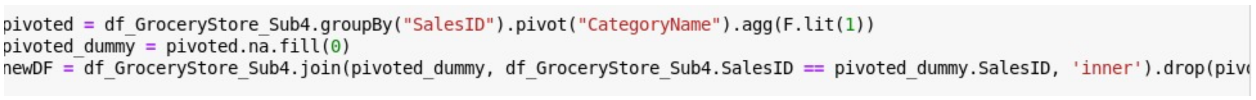
Further in this process we had constructed new variable 'High Status'. 'High Status contains the records of transactions that value over $1000. Using the 'High Status' column, we can identify what kind of transaction get greater revenue to the business. Further analysis into this might reveal any interesting patterns in the sales.



```
In [22]: df_GroceryStoretemp = df_GroceryStore.withColumn("High_Status",when(df_GroceryStore["SalesPrice"] > 1000 , 1 ).otherwise(

In [23]: df_GroceryStoretemp.head(1)

Out[23]: [Row(SalesID=5996077, SalesPersonID=5, CustomerID=6774, Quantity=2, SalesDate=datetime.datetime(2018, 3, 20, 5, 45, 5, 3
         00000), TransactionNumber='YIIAGV3BPM91NUY41WLX', Disc_Percent=0.1, SalesPrice=36.299, ProductID=400, ProductName='Carbo
         nated Water - Blackcherry', Price=18.1495, CategoryID=1, Class='High', ModifyDate=datetime.datetime(2017, 8, 30, 20, 41,
         44, 350000), Resistant='Durable', IsAllergic=True, VitalityDays=None, TotalPrice=36.299, FirstName='Gerald', MiddleIniti
         al='X', LastName='Hamilton', CityID=66, Address='462 White Hague Avenue', CityName='Shreveport', Zipcode=82101, CountryI
         D=32, CategoryName='Confections', High_Status=0)]
```

Transforming text into categories

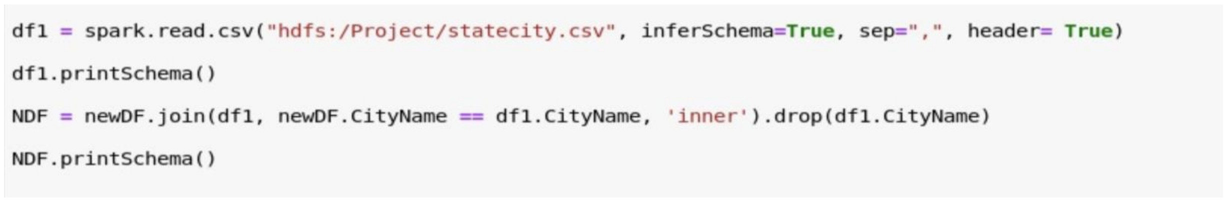
Since one of the main predictors is 'CategoryName' of the products, which has 11 categories, and since it is represented in a text format, we created dummy variables to distinguish each category. Below is the categorization into dummy variables.



```
pivoted = df_GroceryStore_Sub4.groupBy("SalesID").pivot("CategoryName").agg(F.lit(1))
pivoted_dummy = pivoted.na.fill(0)
newDF = df_GroceryStore_Sub4.join(pivoted_dummy, df_GroceryStore_Sub4.SalesID == pivoted_dummy.SalesID, 'inner').drop(piv
```

Discretizing and Aggregating the data

As part of aggregation, we binned the cities that belonged to different regions of north America. The regions include Mid-West region, North-East region, South and west regions respectively. The identified regions in the new dataset were joined with our original dataset (newDF) and the included cities were replaced with respective regions which in effect is binning. Finally, the column 'CityName' is dropped.



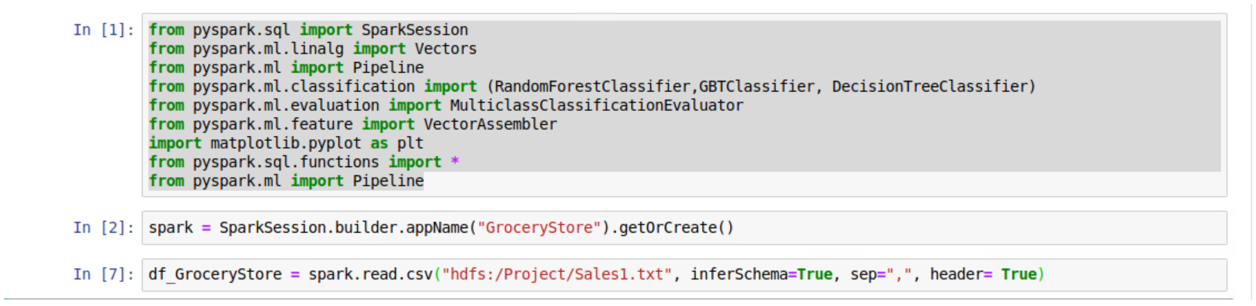
```
df1 = spark.read.csv("hdfs:/Project/statecity.csv", inferSchema=True, sep=",", header= True)
df1.printSchema()
NDF = newDF.join(df1, newDF.CityName == df1.CityName, 'inner').drop(df1.CityName)
NDF.printSchema()
```

Normalizing the data

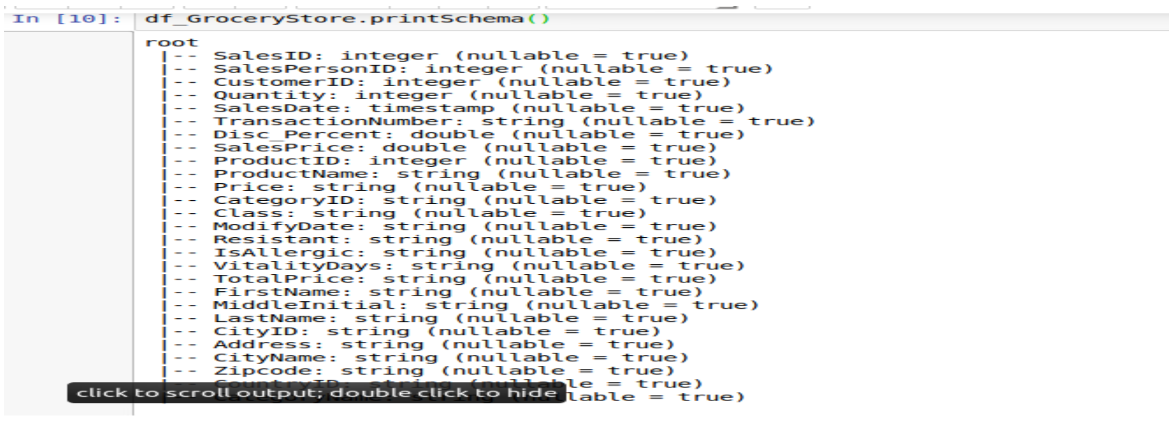
The data points seemed to be almost stable or normal, we did not have to perform any normalization on the dataset.

Data Reduction

Once we obtain the clean data, it is exported as flat file to local machine. Then, it is uploaded to HDFS to be used for our analysis. Below is the screenshot of the Schema of the data.

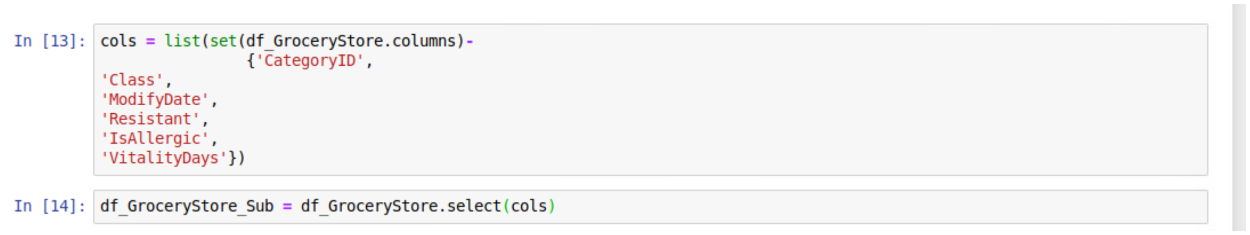


```
In [1]: from pyspark.sql import SparkSession
        from pyspark.ml.linalg import Vectors
        from pyspark.ml import Pipeline
        from pyspark.ml.classification import (RandomForestClassifier,GBTClassifier, DecisionTreeClassifier)
        from pyspark.ml.evaluation import MulticlassClassificationEvaluator
        from pyspark.ml.feature import VectorAssembler
        import matplotlib.pyplot as plt
        from pyspark.sql.functions import *
        from pyspark.ml import Pipeline

In [2]: spark = SparkSession.builder.appName("GroceryStore").getOrCreate()

In [7]: df_GroceryStore = spark.read.csv("hdfs:/Project/Sales1.txt", inferSchema=True, sep=",", header= True)
```



```
In [10]: df_GroceryStore.printSchema()

         root
          |-- SalesID: integer (nullable = true)
          |-- SalesPersonID: integer (nullable = true)
          |-- CustomerID: integer (nullable = true)
          |-- Quantity: integer (nullable = true)
          |-- SalesDate: timestamp (nullable = true)
          |-- TransactionNumber: string (nullable = true)
          |-- Disc_Percent: double (nullable = true)
          |-- SalesPrice: double (nullable = true)
          |-- ProductID: integer (nullable = true)
          |-- ProductName: string (nullable = true)
          |-- Price: string (nullable = true)
          |-- CategoryID: string (nullable = true)
          |-- Class: string (nullable = true)
          |-- ModifyDate: string (nullable = true)
          |-- Resistant: string (nullable = true)
          |-- IsAllergic: string (nullable = true)
          |-- VitalityDays: string (nullable = true)
          |-- TotalPrice: string (nullable = true)
          |-- FirstName: string (nullable = true)
          |-- MiddleInitial: string (nullable = true)
          |-- LastName: string (nullable = true)
          |-- CityID: string (nullable = true)
          |-- Address: string (nullable = true)
          |-- CityName: string (nullable = true)
          |-- Zipcode: string (nullable = true)
          |-- CountryID: string (nullable = true)
   click to scroll output; double click to hide lable = true)
```

The next step is to reduce the data by removing the irrelevant columns. The columns ‘Class’, ‘Modify Date’, ‘Resistant’, ‘Is Allergic’, ‘Vitality Days’ are ambiguous in their representation. Also, there is no clear idea about the description of these columns. Therefore, the columns are dropped from the original data frame. Below screenshot shows the code snippet to delete the columns.



```
In [13]: cols = list(set(df_GroceryStore.columns)-
                         {'CategoryID',
         'Class',
         'ModifyDate',
         'Resistant',
         'IsAllergic',
         'VitalityDays'})

In [14]: df_GroceryStore_Sub = df_GroceryStore.select(cols)
```

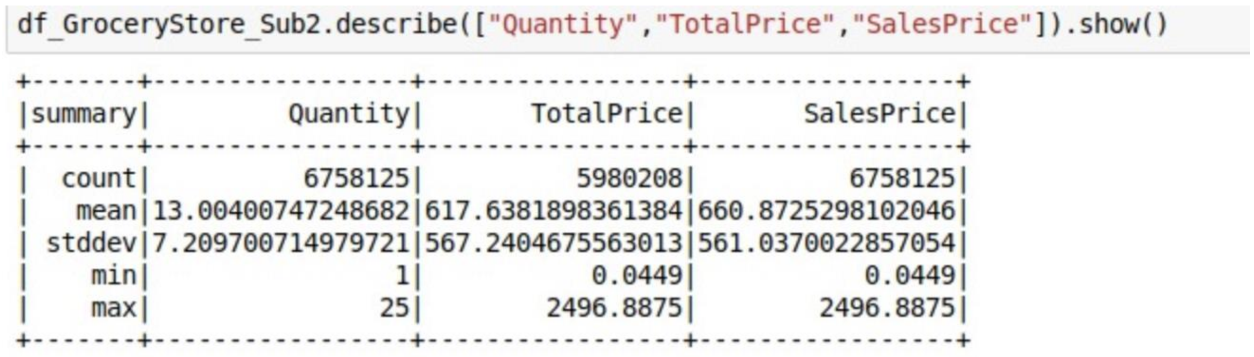
Note: No records have been reduced as part of our reduction process.

Descriptive Statistics

Summary Statistics

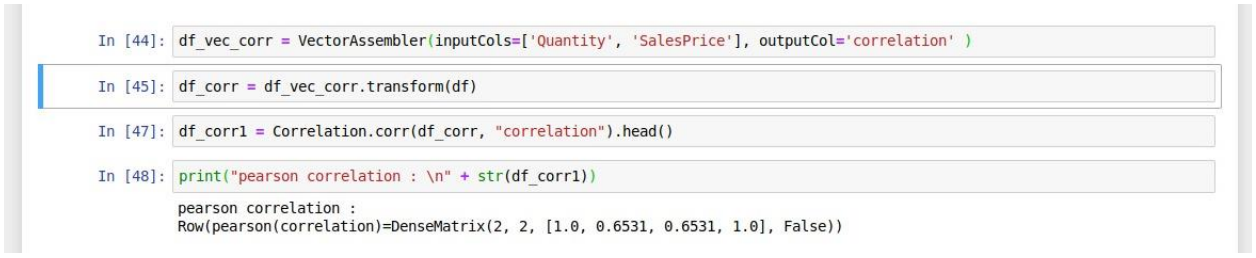
The describe function in PySpark gives the basic descriptive statistics of the data.

Below tables provides the mean, median, minimum value, maximum value, standard deviation and the count of all the records in ‘Sales Price ‘, ‘Total Price’ and ‘Quantity’ columns.



```
df_GroceryStore_Sub2.describe(["Quantity","TotalPrice","SalesPrice"]).show()

+-------+-----------------+-----------------+-----------------+
|summary|         Quantity|       TotalPrice|       SalesPrice|
+-------+-----------------+-----------------+-----------------+
|  count|          6758125|          5980208|          6758125|
|   mean|13.00400747248682|617.6381898361384|660.8725298102046|
| stddev|7.209700714979721|567.2404675563013|561.0370022857054|
|    min|                1|           0.0449|           0.0449|
|    max|               25|        2496.8875|        2496.8875|
+-------+-----------------+-----------------+-----------------+
```

Correlation



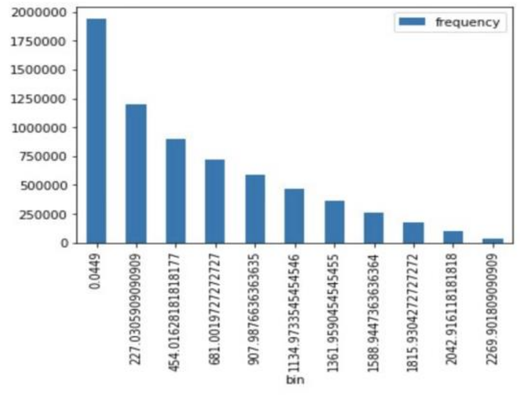
```
In [44]: df_vec_corr = VectorAssembler(inputCols=['Quantity', 'SalesPrice'], outputCol='correlation' )

In [45]: df_corr = df_vec_corr.transform(df)

In [47]: df_corr1 = Correlation.corr(df_corr, "correlation").head()

In [48]: print("pearson correlation : \n" + str(df_corr1))

         pearson correlation :
         Row(pearson(correlation)=DenseMatrix(2, 2, [1.0, 0.6531, 0.6531, 1.0], False))
```

The above Pearson correlation explains that there is a linear correlation between quantity and SalesPrice. Since the dependent variable High Status which indicate High value transaction is being derived from “Total price” information; including it in the predictor variable list does not make sense and hence dropping the same.

Visualizations

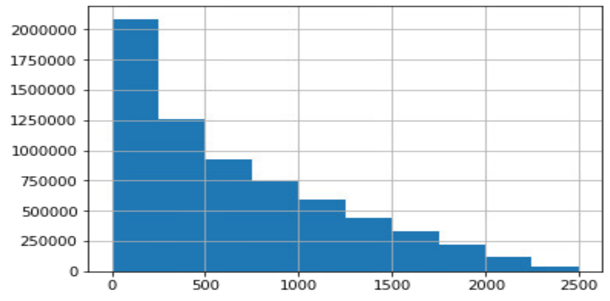
Histogram

```
pd.DataFrame(
    list(zip(*gre_histogram)),
    columns=['bin', 'frequency']
).set_index(
    'bin'
).plot(kind='bar');
```



The above histogram is for the column Quantity. The above graph tells that quantity is skewed. However, this is valid because there are fewer customers who buy products in higher quantities.

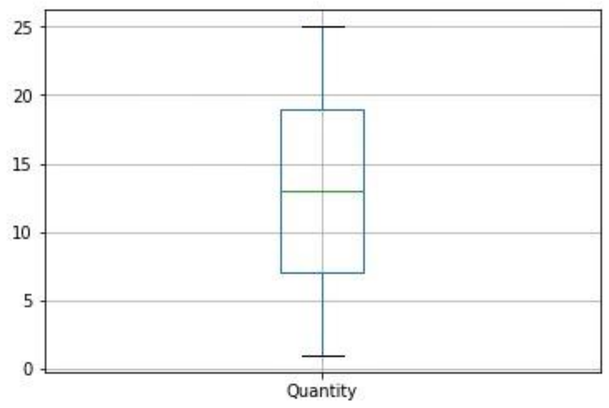
```
In [52]: df['SalesPrice'].hist()
Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x7effa92bdeb8>
```



The above histogram is for the column SalesPrice and from the above graph we can observe that the data is not normal. It can be inferred from the quantity histogram that there are fewer bulk transactions which accounts for higher sales price and vice versa.

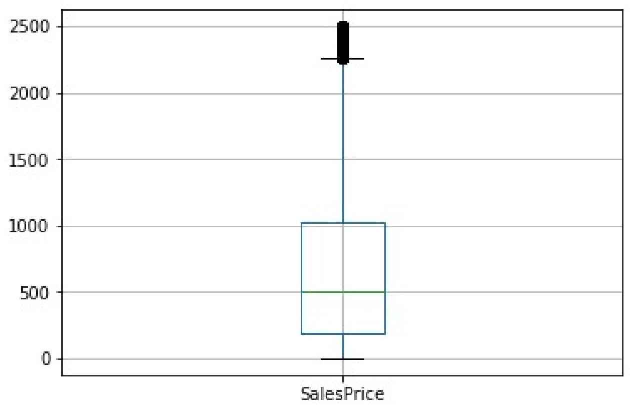
Boxplots

```
In [11]: df.boxplot("Quantity")
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7effce8d32e8>
```



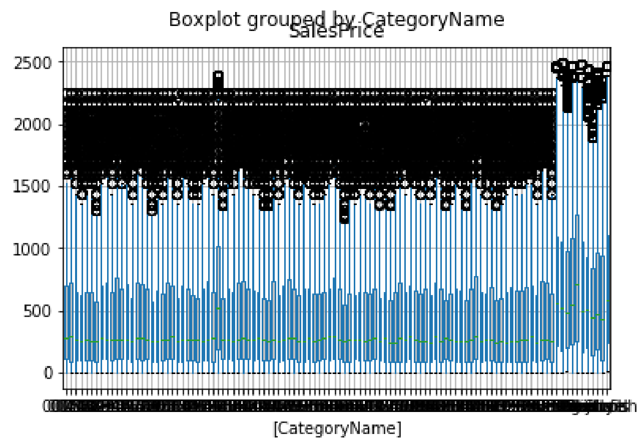
By analyzing the above box plot, it is clearly observed that there are no outliers present in the variable "Quantity" and hence it need not be treated further.

```
In [24]: df.boxplot("SalesPrice")
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9ce4197780>
```



The above box plot is for the column SalesPrice. This graph shows that there some outliers in the data. These outliers were examined and found to be valid since few of the customers purchase in bulk. Those aforementioned outliers possibly relate to the high value transactions which is the prime area of interest in our model.

```
In [22]: df.boxplot(by = "CategoryName")
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7effbbcb1e80>
```



The above box plot is for the column SalesPrice based on CategoryName column. Since there are multiple categories in the CategroyName column the graph is overlapped. However, we can observe that there are many outliers which must be taken while performing the predictive analytics.

Why our data qualifies for the Big-data

Volume

The initial main transactional data was for just 500 MB approximately. After data consolidation with other descriptive variables, the volume of the consolidated data is now 1.49 GB. Also, the overall dataset has around 6.7 M transactional records which is huge data in volume.

Velocity

The dataset which is under our consideration here, consists of daily sales transactional data across the span of seven continuous months. Since this data is getting recorded on the basis of each transaction as soon as it is taking place, this dataset also qualifies for the velocity, which measures the speed of how fast the data is getting generated.

Variety

We also observed that this dataset consists of categorical and continuous variables. This includes descriptive categorical information from different subject areas like customer, sales, employees, cities, products, and corresponding numerical information on those areas. This presents us with a data with a significant variety.

Model Plan

We plan to predict the 'High_status' variable which would categorize the transactions that accounts for high sale and the low sale. Since we have a set of continuous and categorical variables as predictors, we intend to build the following three models to test with our data, considering High_status (binary variable) as our target variable:

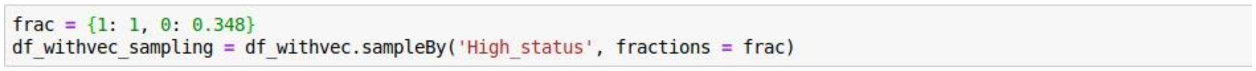
1. Logistic regression
2. Decision tree
3. Random forest

Since our primary target variable is 'High Status' which is a categorical variable with outputs 1 and 0, logistic regression and decision tree would be ideal in such case. The reason we think that these models will serve better for the classification we intend to achieve, I.e. our data is almost equal combination of continuous and categorical variables. Since we know that no particular model performs well in all the cases, and we also know that decision trees are prone to overfitting, but they take the interaction among the variables into account, while logistic regression models are less susceptible to overfitting, however they do not consider the interactions equally. To get a more accurate and stable predictions, we also intend to implement Random forest model and then we will evaluate the performance of all the model to see which one is performing better in the given scenario.

Data Splitting and Sub sampling

Stratified Sampling

The data points seem to be a little unbalanced between the high value transactions and low value transactions, which lead to the step of stratified sampling, where we strategically divide the data points into a balanced set of both high and low transactional values. Below code snippet does the required job.

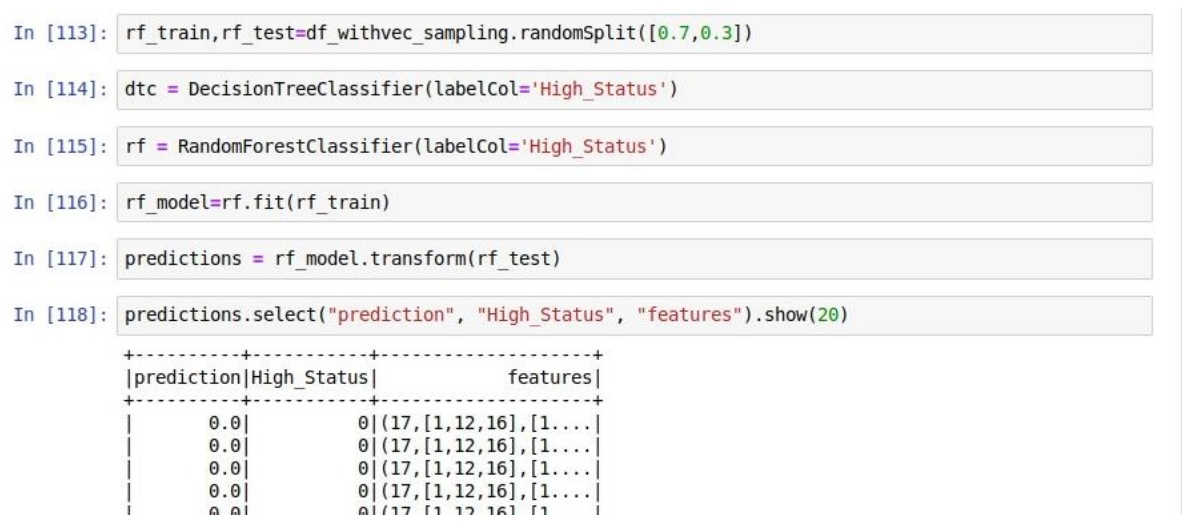


```
frac = {1: 1, 0: 0.348}
df_withvec_sampling = df_withvec.sampleBy('High_status', fractions = frac)
```

Data Spilt

As a next step, dataset is split into training dataset and test dataset. Training dataset is required to train the data points to understand the pattern of the target variable depending upon the changes in predictor variables. While, test dataset is required to predict the values of the target variable based in the learning from training dataset.

Therefore, 70% of the data points are allotted to training dataset and the remaining 30% of the data points are allotted to test data set. Below is the data split code snipped of the dataset into 70:30 ratio of training and test datasets.



```
In [113]: rf_train,rf_test=df_withvec_sampling.randomSplit([0.7,0.3])

In [114]: dtc = DecisionTreeClassifier(labelCol='High_Status')

In [115]: rf = RandomForestClassifier(labelCol='High_Status')

In [116]: rf_model=rf.fit(rf_train)

In [117]: predictions = rf_model.transform(rf_test)

In [118]: predictions.select("prediction", "High_Status", "features").show(20)

          +----------+-----------+-------------------+
          |prediction|High_Status|           features|
          +----------+-----------+-------------------+
          |       0.0|          0|(17,[1,12,16],[1....|
          |       0.0|          0|(17,[1,12,16],[1....|
          |       0.0|          0|(17,[1,12,16],[1....|
          |       0.0|          0|(17,[1,12,16],[1....|
          |       0.0|          0|(17,[1,12,16],[1....|
```

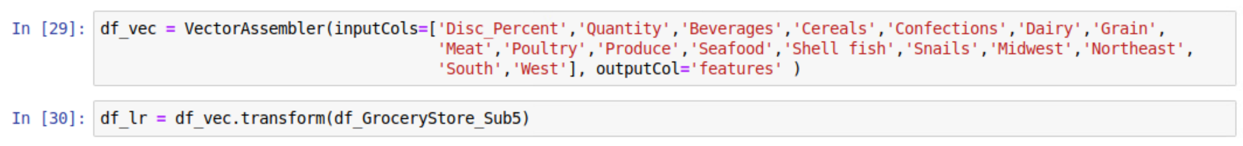
Model Building and Model Assessment

Logistic Regression

Our target variable for the data we have in hand is “High_Status”, which is a dichotomous variable we created to predict whether the transaction value for an instance would be high (encoded-1) or low (encoded-0).

Also, the set of input variables we have is a collection of categorical and continuous variables, and it is observed that prediction with multiple input variables against a dichotomous variable performs better with a logistic regression model, me decided to build it.

Model



```
In [29]: df_vec = VectorAssembler(inputCols=['Disc_Percent','Quantity','Beverages','Cereals','Confections','Dairy','Grain',
                                              'Meat','Poultry','Produce','Seafood','Shell fish','Snails','Midwest','Northeast',
                                              'South','West'], outputCol='features' )

In [30]: df_lr = df_vec.transform(df_GroceryStore_Sub5)
```

In [31]:
```
df_lr.printSchema()
```

```
root
 |-- Disc_Percent: double (nullable = false)
 |-- High_Status: integer (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- Beverages: integer (nullable = true)
 |-- Cereals: integer (nullable = true)
 |-- Confections: integer (nullable = true)
 |-- Dairy: integer (nullable = true)
 |-- Grain: integer (nullable = true)
 |-- Meat: integer (nullable = true)
 |-- Poultry: integer (nullable = true)
 |-- Produce: integer (nullable = true)
 |-- Seafood: integer (nullable = true)
 |-- Shell fish: integer (nullable = true)
 |-- Snails: integer (nullable = true)
 |-- Midwest: integer (nullable = true)
 |-- Northeast: integer (nullable = true)
 |-- South: integer (nullable = true)
 |-- West: integer (nullable = true)
 |-- features: vector (nullable = true)
```

In [32]:
```
dflr_train, dflr_test = df_lr.randomSplit([0.7,0.3])
```

In [33]:
```
dflr_train.count()
```

Out[33]:
```
4632158
```

In [34]:
```
dflr_test.count()
```

Out[34]:
```
1986637
```

Logistic Regression Results

In [35]:
```
lr = LogisticRegression(labelCol='High_Status',featuresCol='features')
```

In [36]:
```
lrModel = lr.fit(dflr_train)
```

In [37]:
```
lrModel.coefficients
```

Out[37]:
```
DenseVector([-0.0311, 0.2466, -1.0732, -1.2521, -1.1857, -1.0365, -0.5553, -1.042, -1.3925, -1.5429, -1.4183, -1.5652, -
1.0208, -1.5738, -1.5716, -1.5672, -1.5711])
```

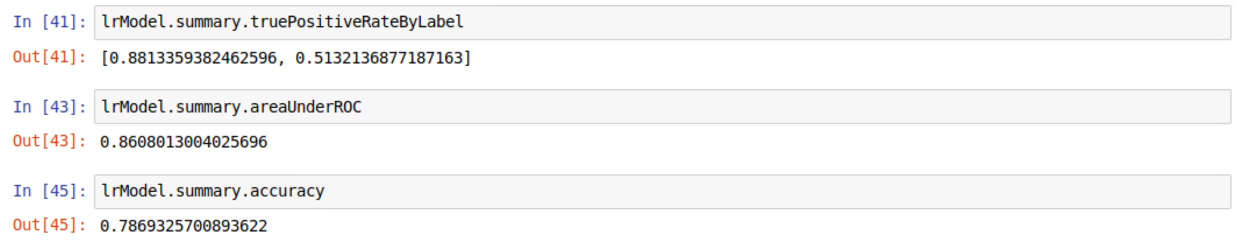
In [38]:
```
lrModel.intercept
```

Out[38]:
```
-2.156907773777088
```

In [40]:
```
test_result.predictions.show()
```

```
+------------+-----------+--------+---------+-------+-----------+-----+-----+----+-------+-------+-------+----------+---
---+-------+---------+-----+----+--------------------+--------------------+--------------------+----------+
|Disc_Percent|High_Status|Quantity|Beverages|Cereals|Confections|Dairy|Grain|Meat|Poultry|Produce|Seafood|Shell fish|Sna
ils|Midwest|Northeast|South|West|            features|       rawPrediction|         probability|prediction|
+------------+-----------+--------+---------+-------+-----------+-----+-----+----+-------+-------+-------+----------+---
---+-------+---------+-----+----+--------------------+--------------------+--------------------+----------+
|         0.0|          0|       1|        0|      0|          0|    0|    0|   0|      0|      0|      0|         0|
1|      0|        0|    0|   1|(17,[1,12,16],[1....|[4.50214429872922...|[0.98903633339606...|       0.0|
|         0.0|          0|       1|        0|      0|          0|    0|    0|   0|      0|      0|      0|         0|
1|      0|        0|    0|   1|(17,[1,12,16],[1....|[4.50214429872922...|[0.98903633339606...|       0.0|
|         0.0|          0|       1|        0|      0|          0|    0|    0|   0|      0|      0|      0|         0|
1|      0|        0|    0|   1|(17,[1,12,16],[1....|[4.50214429872922...|[0.98903633339606...|       0.0|
|         0.0|          0|       1|        0|      0|          0|    0|    0|   0|      0|      0|      0|         0|
1|      0|        0|    0|   1|(17,[1,12,16],[1....|[4.50214429872922...|[0.98903633339606...|       0.0|
|         0.0|          0|       1|        0|      0|          0|    0|    0|   0|      0|      0|      0|         0|
1|      0|        0|    0|   1|(17,[1,12,16],[1....|[4.50214429872922...|[0.98903633339606...|       0.0|
|         0.0|          0|       1|        0|      0|          0|    0|    0|   0|      0|      0|      0|         0|
1|      0|        0|    1|   0|(17,[1,12,15],[1....|[4.49823966005063...|[0.98899391263572...|       0.0|
|         0.0|          0|       1|        0|      0|          0|    0|    0|   0|      0|      0|      0|         0|
1|      0|        0|    1|   0|(17,[1,12,15],[1....|[4.49823966005063...|[0.98899391263572...|       0.0|
|         0.0|          0|       1|        0|      0|          0|    0|    0|   0|      0|      0|      0|         0|
1|      0|        0|    1|   0|(17,[1,12,15],[1....|[4.49823966005063...|[0.98899391263572...|       0.0|
```

Logistic Regression Accuracy



```
In [41]: lrModel.summary.truePositiveRateByLabel
Out[41]: [0.8813359382462596, 0.5132136877187163]

In [43]: lrModel.summary.areaUnderROC
Out[43]: 0.8608013004025696

In [45]: lrModel.summary.accuracy
Out[45]: 0.7869325700893622
```

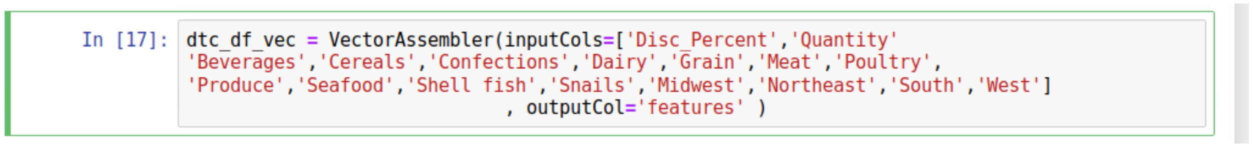
Interpretation

- Assessment of the True Positive rate suggests that 88.13 % of the actual high transactions were predicted as high, which represents a good sensitivity of the model.
- While accessing the ROC curve here, we observed 86.08% area is under the curve, indicating a fairly good model performance.
- Also, the overall accuracy of the regression model is obtained as 78.69% in predicting the high transactions accurately.

Decision Tree

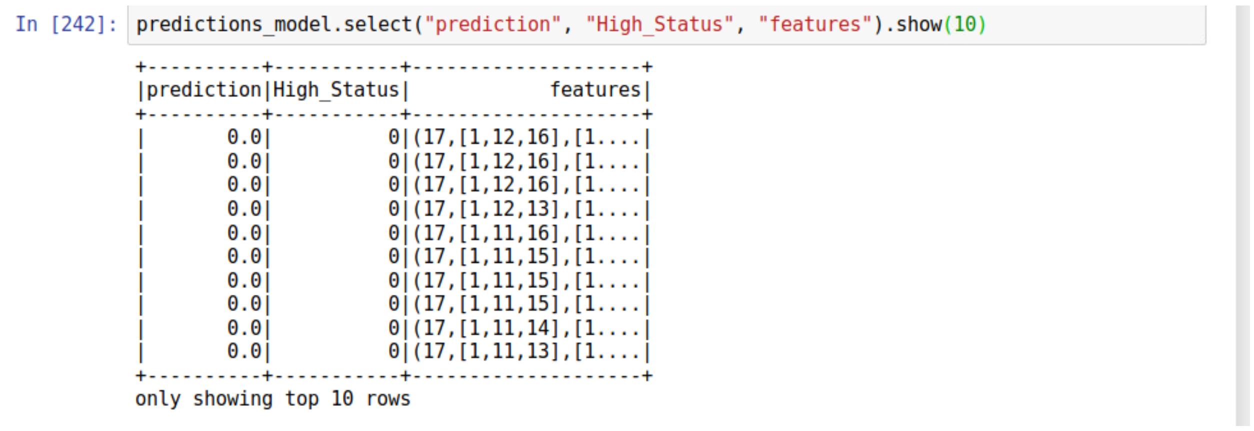
Decision trees best explains the most significant variable that is influencing the target variable, by splitting the dataset into two parts. Our interest in finding out the most significant input variables, the ease of understanding that a decision tree offers, and better interpretability in classifying our target variable, made us to build this model and analyze.

Decision Tree Model

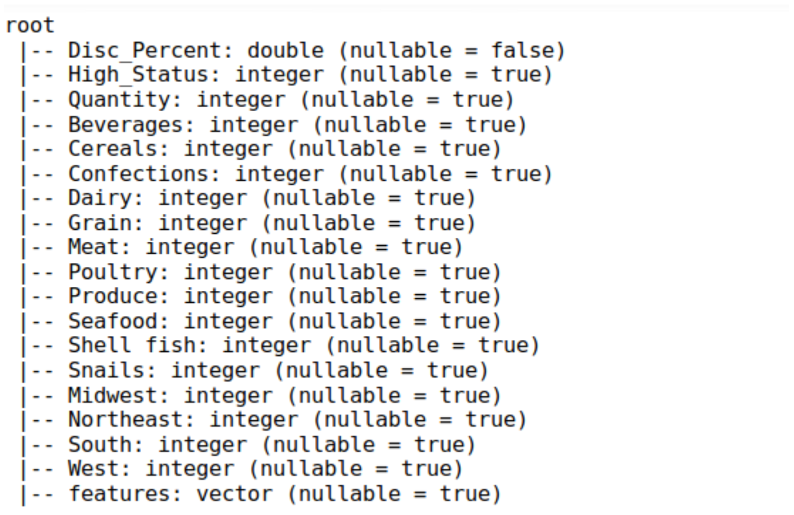


```
In [17]: dtc_df_vec = VectorAssembler(inputCols=['Disc_Percent','Quantity'
         'Beverages','Cereals','Confections','Dairy','Grain','Meat','Poultry',
         'Produce','Seafood','Shell fish','Snails','Midwest','Northeast','South','West']
                                          , outputCol='features' )
```

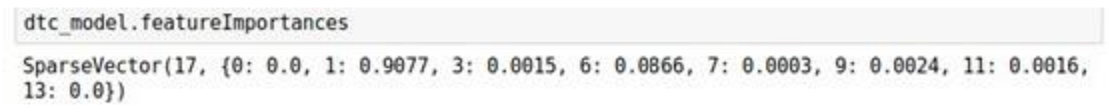
Decision Tree Results



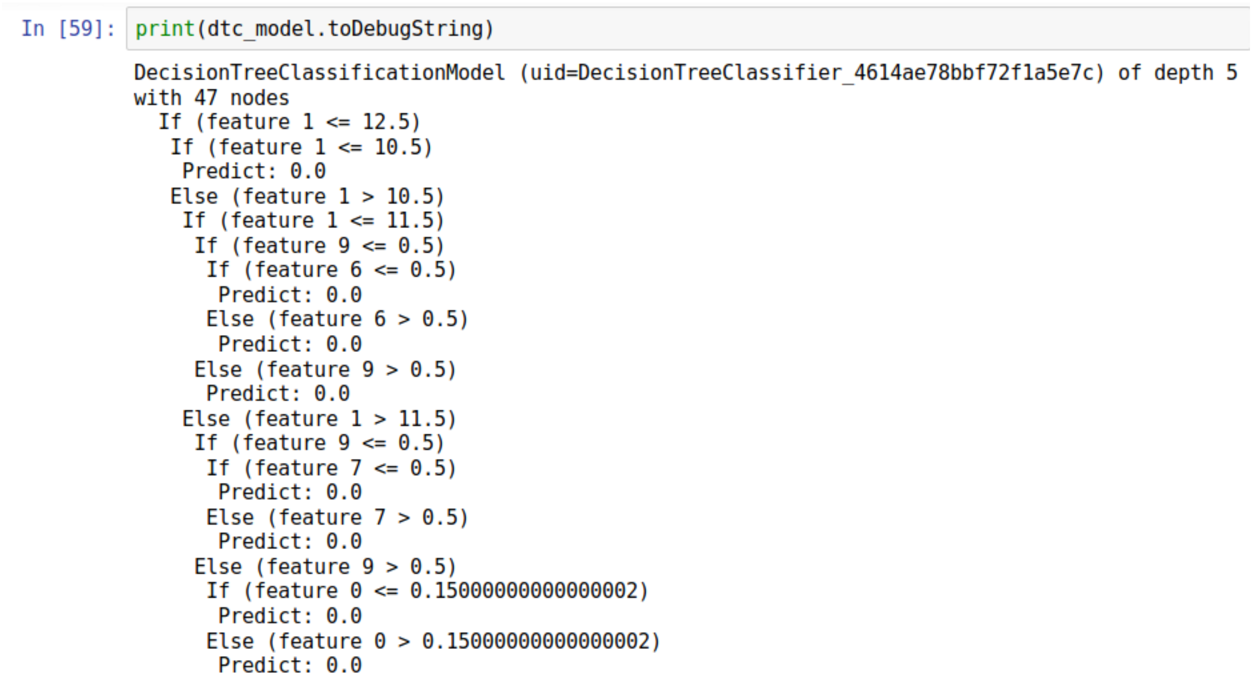
```
In [242]: predictions_model.select("prediction", "High_Status", "features").show(10)

+----------+-----------+-------------------+
|prediction|High_Status|           features|
+----------+-----------+-------------------+
|       0.0|          0|(17,[1,12,16],[1....|
|       0.0|          0|(17,[1,12,16],[1....|
|       0.0|          0|(17,[1,12,16],[1....|
|       0.0|          0|(17,[1,12,13],[1....|
|       0.0|          0|(17,[1,11,16],[1....|
|       0.0|          0|(17,[1,11,15],[1....|
|       0.0|          0|(17,[1,11,15],[1....|
|       0.0|          0|(17,[1,11,15],[1....|
|       0.0|          0|(17,[1,11,14],[1....|
|       0.0|          0|(17,[1,11,13],[1....|
+----------+-----------+-------------------+
only showing top 10 rows
```



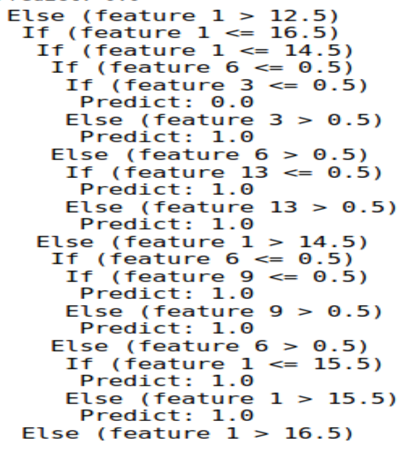
```
root
 |-- Disc_Percent: double (nullable = false)
 |-- High_Status: integer (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- Beverages: integer (nullable = true)
 |-- Cereals: integer (nullable = true)
 |-- Confections: integer (nullable = true)
 |-- Dairy: integer (nullable = true)
 |-- Grain: integer (nullable = true)
 |-- Meat: integer (nullable = true)
 |-- Poultry: integer (nullable = true)
 |-- Produce: integer (nullable = true)
 |-- Seafood: integer (nullable = true)
 |-- Shell fish: integer (nullable = true)
 |-- Snails: integer (nullable = true)
 |-- Midwest: integer (nullable = true)
 |-- Northeast: integer (nullable = true)
 |-- South: integer (nullable = true)
 |-- West: integer (nullable = true)
 |-- features: vector (nullable = true)
```



```
dtc_model.featureImportances

SparseVector(17, {0: 0.0, 1: 0.9077, 3: 0.0015, 6: 0.0866, 7: 0.0003, 9: 0.0024, 11: 0.0016,
13: 0.0})
```

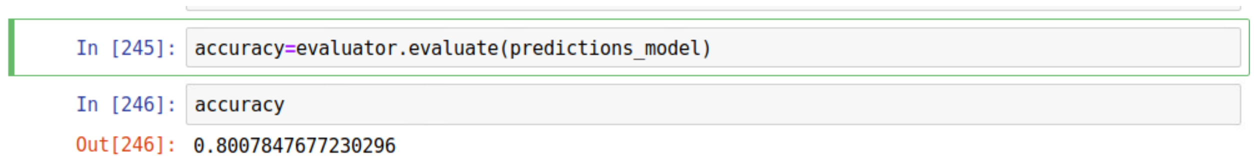


```
In [59]: print(dtc_model.toDebugString)

DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4614ae78bbf72f1a5e7c) of depth 5
with 47 nodes
  If (feature 1 <= 12.5)
   If (feature 1 <= 10.5)
    Predict: 0.0
   Else (feature 1 > 10.5)
    If (feature 1 <= 11.5)
     If (feature 9 <= 0.5)
      If (feature 6 <= 0.5)
       Predict: 0.0
      Else (feature 6 > 0.5)
       Predict: 0.0
     Else (feature 9 > 0.5)
      Predict: 0.0
    Else (feature 1 > 11.5)
     If (feature 9 <= 0.5)
      If (feature 7 <= 0.5)
       Predict: 0.0
      Else (feature 7 > 0.5)
       Predict: 0.0
     Else (feature 9 > 0.5)
      If (feature 0 <= 0.15000000000000002)
       Predict: 0.0
      Else (feature 0 > 0.15000000000000002)
       Predict: 0.0
```



```
Else (feature 1 > 12.5)
 If (feature 1 <= 16.5)
  If (feature 1 <= 14.5)
   If (feature 6 <= 0.5)
    If (feature 3 <= 0.5)
     Predict: 0.0
    Else (feature 3 > 0.5)
     Predict: 1.0
   Else (feature 6 > 0.5)
    If (feature 13 <= 0.5)
     Predict: 1.0
    Else (feature 13 > 0.5)
     Predict: 1.0
  Else (feature 1 > 14.5)
   If (feature 6 <= 0.5)
    If (feature 9 <= 0.5)
     Predict: 1.0
    Else (feature 9 > 0.5)
     Predict: 1.0
   Else (feature 6 > 0.5)
    If (feature 1 <= 15.5)
     Predict: 1.0
    Else (feature 1 > 15.5)
     Predict: 1.0
 Else (feature 1 > 16.5)
```

Decision Tree Accuracy



```
In [245]: accuracy=evaluator.evaluate(predictions_model)

In [246]: accuracy

Out[246]: 0.8007847677230296
```

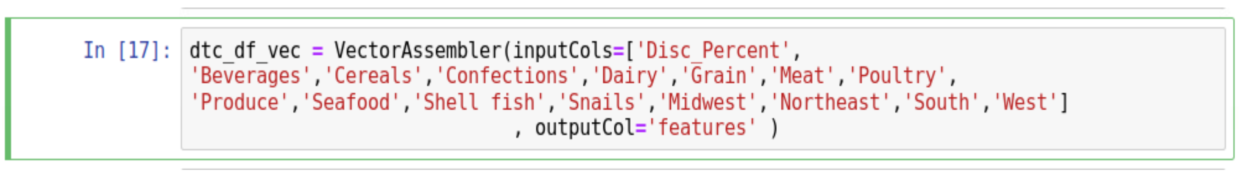
Decision Tree Interpretation

- Broadly, we observed from the classification that whenever the sale quantity is high, the product categories that are accounting for the high transactions are observed to be Grains, Produce and Cereal.
- An accuracy of 80.08% indicates that the model is performing significantly good, predicting high value transactions.
- While assessing Feature Importance for the decision tree classifier model, we observed that the variable "Quantity" is showing comparatively exceptional high importance of 0.9077, which is quite obvious as well because in ideal situations if the quantity in the sale increases, the value for transaction would also increase.
- Since "Quantity" seems to have a feature importance of 0.90, it is indeed the most significant factor in determining the high value transaction status, which is followed by grain category component.
- This interpretation made us re-run the model without "Quantity" and observe what we come up with, as explained after this.

Note: Our iterative assessments with models without the independent variable "Quantity" yielded very poor results toward the tone of 50-57% accuracy, an example of which is shown below.

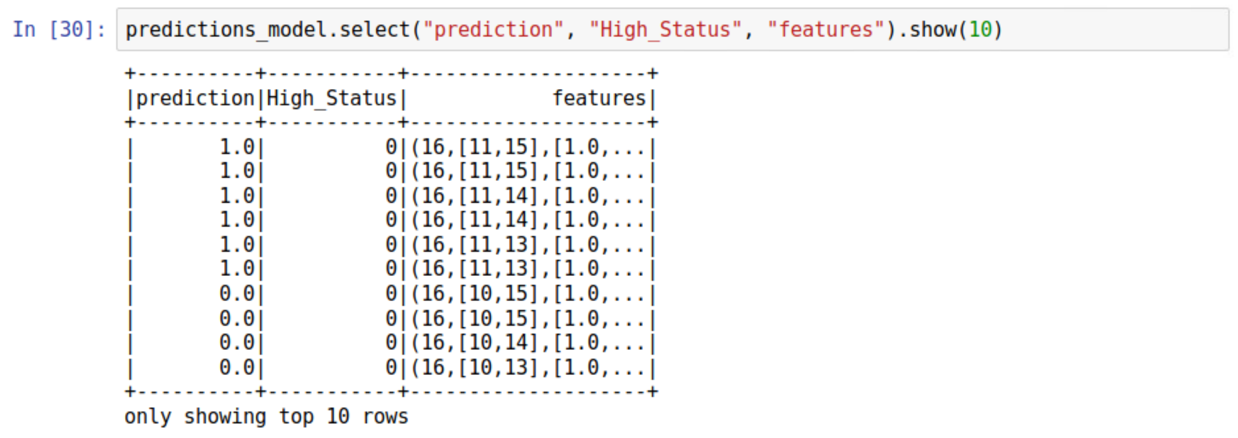
Re-running the Decision tree model with "Quantity" variable

Decision Tree Model (without Quantity)

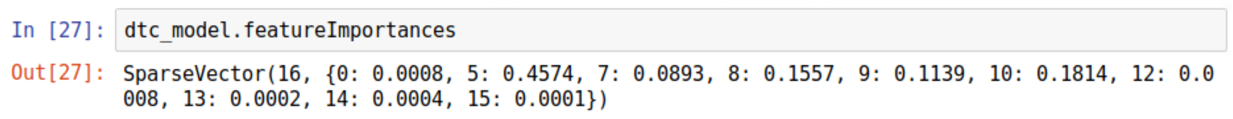


```
In [17]: dtc_df_vec = VectorAssembler(inputCols=['Disc_Percent',
         'Beverages','Cereals','Confections','Dairy','Grain','Meat','Poultry',
         'Produce','Seafood','Shell fish','Snails','Midwest','Northeast','South','West']
                                                 , outputCol='features' )
```

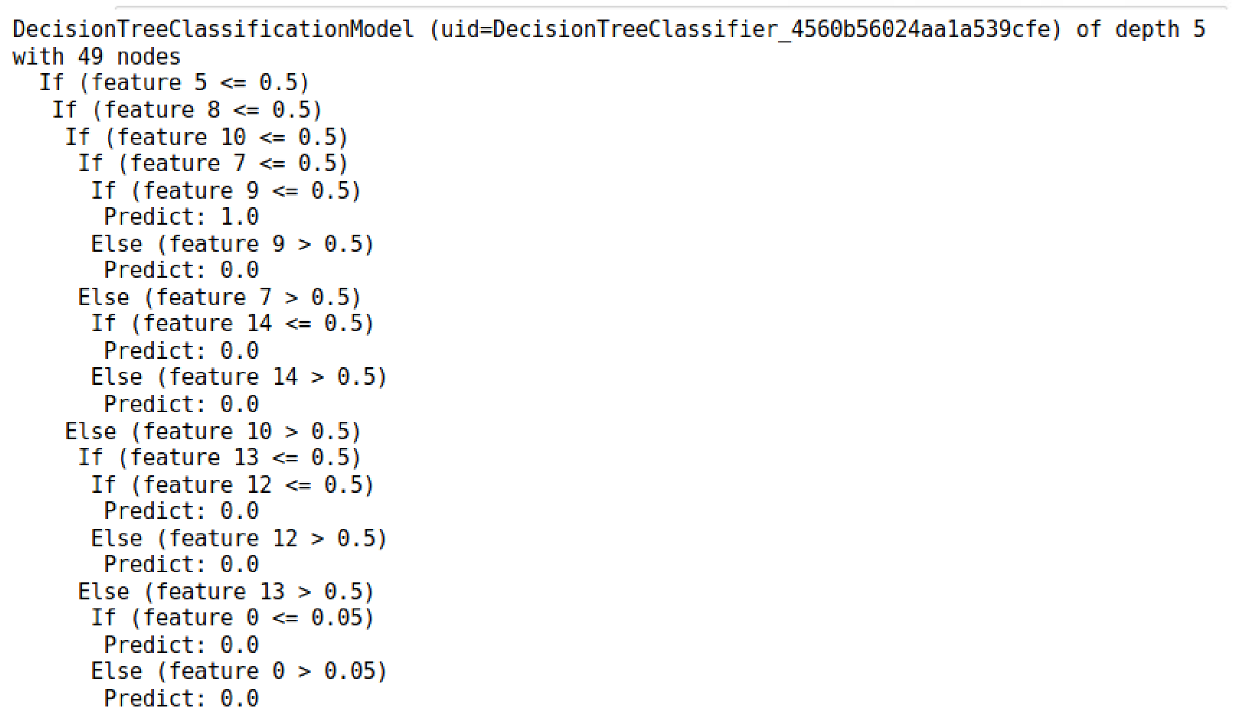
Decision Tree Results (Without Quantity)



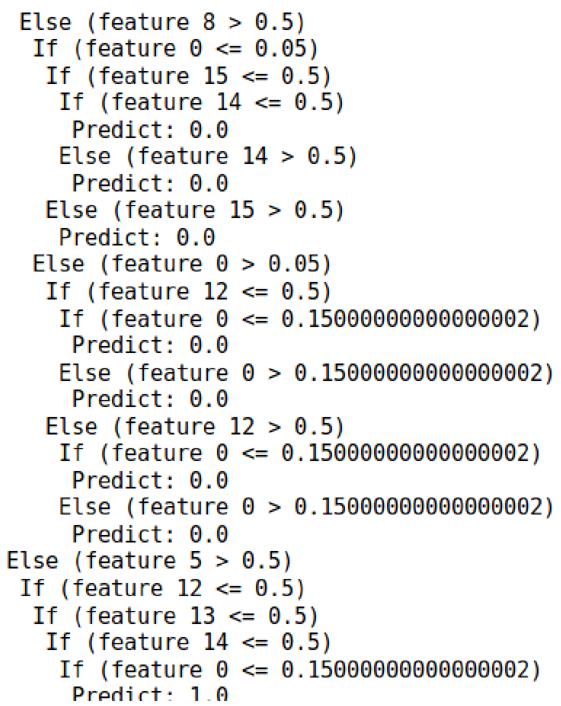
```
In [30]: predictions_model.select("prediction", "High_Status", "features").show(10)

         +----------+-----------+--------------------+
         |prediction|High_Status|            features|
         +----------+-----------+--------------------+
         |       1.0|          0|(16,[11,15],[1.0,...|
         |       1.0|          0|(16,[11,15],[1.0,...|
         |       1.0|          0|(16,[11,14],[1.0,...|
         |       1.0|          0|(16,[11,14],[1.0,...|
         |       1.0|          0|(16,[11,13],[1.0,...|
         |       1.0|          0|(16,[11,13],[1.0,...|
         |       0.0|          0|(16,[10,15],[1.0,...|
         |       0.0|          0|(16,[10,15],[1.0,...|
         |       0.0|          0|(16,[10,14],[1.0,...|
         |       0.0|          0|(16,[10,13],[1.0,...|
         +----------+-----------+--------------------+
         only showing top 10 rows
```



```
In [27]: dtc_model.featureImportances

Out[27]: SparseVector(16, {0: 0.0008, 5: 0.4574, 7: 0.0893, 8: 0.1557, 9: 0.1139, 10: 0.1814, 12: 0.0
         008, 13: 0.0002, 14: 0.0004, 15: 0.0001})
```

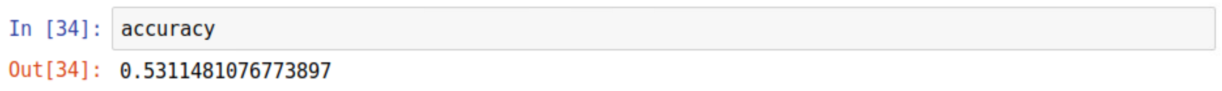


```
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4560b56024aa1a539cfe) of depth 5
with 49 nodes
  If (feature 5 <= 0.5)
   If (feature 8 <= 0.5)
    If (feature 10 <= 0.5)
     If (feature 7 <= 0.5)
      If (feature 9 <= 0.5)
       Predict: 1.0
      Else (feature 9 > 0.5)
       Predict: 0.0
     Else (feature 7 > 0.5)
      If (feature 14 <= 0.5)
       Predict: 0.0
      Else (feature 14 > 0.5)
       Predict: 0.0
    Else (feature 10 > 0.5)
     If (feature 13 <= 0.5)
      If (feature 12 <= 0.5)
       Predict: 0.0
      Else (feature 12 > 0.5)
       Predict: 0.0
     Else (feature 13 > 0.5)
      If (feature 0 <= 0.05)
       Predict: 0.0
      Else (feature 0 > 0.05)
       Predict: 0.0
```



```
 Else (feature 8 > 0.5)
  If (feature 0 <= 0.05)
   If (feature 15 <= 0.5)
    If (feature 14 <= 0.5)
     Predict: 0.0
    Else (feature 14 > 0.5)
     Predict: 0.0
   Else (feature 15 > 0.5)
    Predict: 0.0
  Else (feature 0 > 0.05)
   If (feature 12 <= 0.5)
    If (feature 0 <= 0.15000000000000002)
     Predict: 0.0
    Else (feature 0 > 0.15000000000000002)
     Predict: 0.0
   Else (feature 12 > 0.5)
    If (feature 0 <= 0.15000000000000002)
     Predict: 0.0
    Else (feature 0 > 0.15000000000000002)
     Predict: 0.0
Else (feature 5 > 0.5)
 If (feature 12 <= 0.5)
  If (feature 13 <= 0.5)
   If (feature 14 <= 0.5)
    If (feature 0 <= 0.15000000000000002)
     Predict: 1.0
```

Decision Tree Accuracy (Without Quantity)



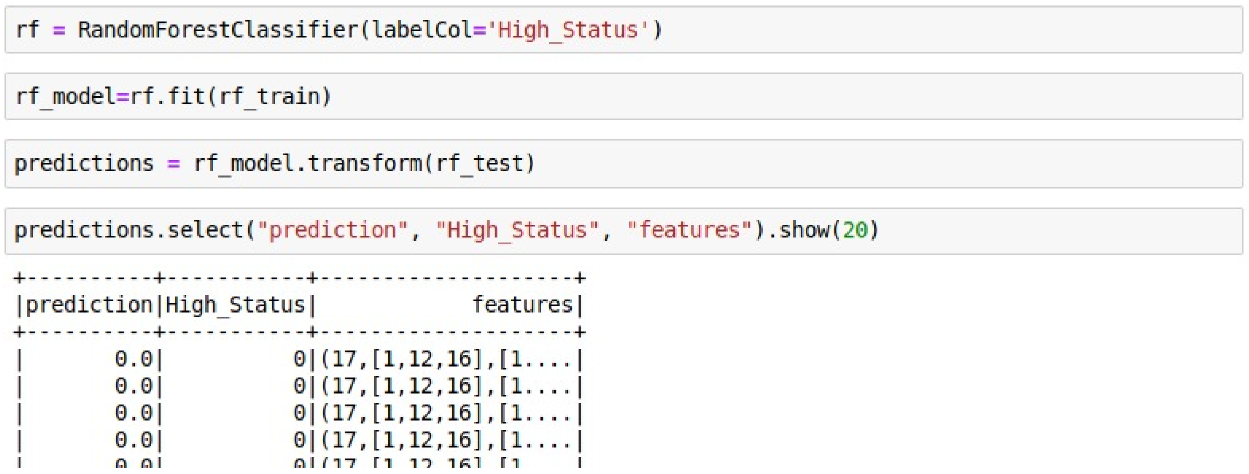
```
In [34]: accuracy
Out[34]: 0.5311481076773897
```

Interpretation

- Contrary to our expectations and as explained above, we noticed a sharp decrease in accuracy from 80.07% with Quantity variable, to 53.11% without it.
- The model without the Quantity highlights the importance of category Grain with feature importance of 0.45 followed by produce and sea food. This further illustrates the importance of categories grain and produce.

Random Forest model

Since the dataset is huge, we used Random Forest in order to minimize the problem of overfitting. Since this technique selects the best out of all the trees, the result is expected to be the average of 5 trees. This model is flexible to any new kind of data and so there will be less chances of over fitting



```
rf = RandomForestClassifier(labelCol='High_Status')
```

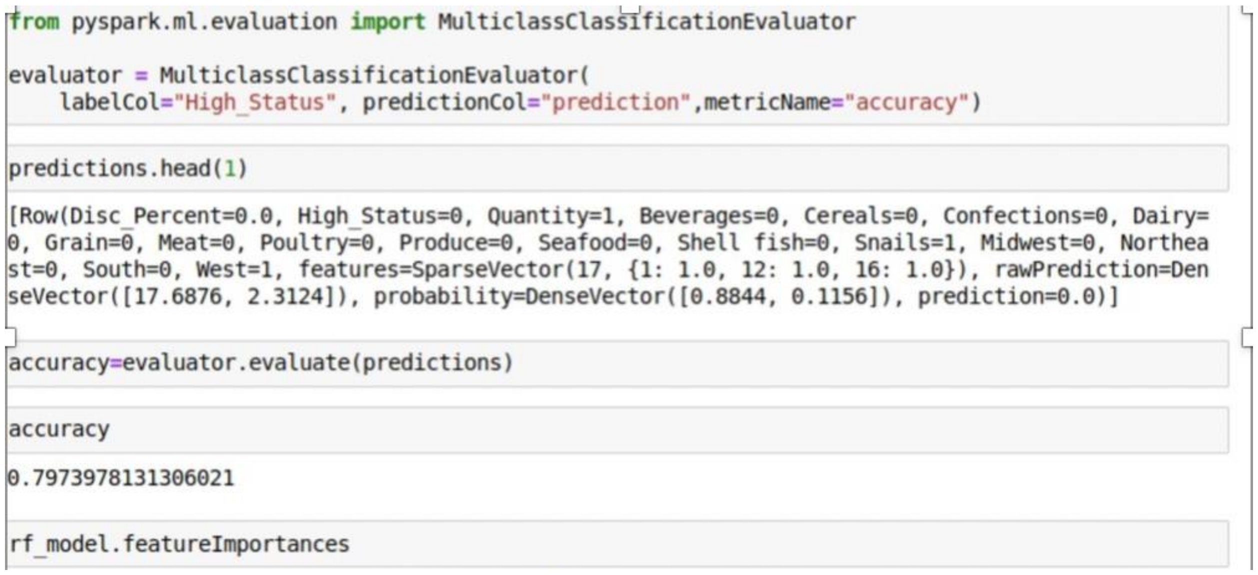
```
rf_model=rf.fit(rf_train)
```

```
predictions = rf_model.transform(rf_test)
```

```
predictions.select("prediction", "High_Status", "features").show(20)
```

```
+----------+-----------+--------------------+
|prediction|High_Status|            features|
+----------+-----------+--------------------+
|       0.0|          0|(17,[1,12,16],[1....|
|       0.0|          0|(17,[1,12,16],[1....|
|       0.0|          0|(17,[1,12,16],[1....|
|       0.0|          0|(17,[1,12,16],[1....|
```

Results and Accuracy



```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(
    labelCol="High_Status", predictionCol="prediction",metricName="accuracy")
```

```
predictions.head(1)
```

```
[Row(Disc_Percent=0.0, High_Status=0, Quantity=1, Beverages=0, Cereals=0, Confections=0, Dairy=
0, Grain=0, Meat=0, Poultry=0, Produce=0, Seafood=0, Shell fish=0, Snails=1, Midwest=0, Northea
st=0, South=0, West=1, features=SparseVector(17, {1: 1.0, 12: 1.0, 16: 1.0}), rawPrediction=Den
seVector([17.6876, 2.3124]), probability=DenseVector([0.8844, 0.1156]), prediction=0.0)]
```

```
accuracy=evaluator.evaluate(predictions)
```

```
accuracy
```

```
0.7973978131306021
```

```
rf_model.featureImportances
```

One of the intents of running a random forest was to obtain a higher accuracy rate since random forest combines the results from multiple trees. In contrast, there is a slight decrease in the accuracy rate of the random forest in comparison to a generalized decision tree and both yield more or less same results.

“Quantity” has a significant share in the predictability in classifying the target variable “High transaction status”. Upon further observation, there were many instances where the combination of high quantity coupled with certain categories like Grain, Cereal, and Produce resulted in final classification as High transaction status. On the other hand, few of the instances like that of Sea-food also resulted in high transactions rarely. Further subjective analysis revealed the aspect of increased production of Corn, Grain and other produce in the mid-west region of the US wherein most of the stores seems to be located.

Comparison

Model / Measure	Logistic Regression	Decision Tree	Random Forest
Accuracy	78.65%	80.07%	79.73%

From the above table, by examining the accuracies of three models, we can see that decision tree performs the better. We can justify this by interpreting the nodes of decision tree which seems to be true in a real scenario. By taking into account, the explainability and accuracy of the decision tree, we concluded it to be the better performing model among the three.

Project Result and its Real-world relevance

Ideally, one might assume that categories like meat or confectionaries would account for the larger value transactions since these are mostly consumed by most American households regularly and are of comparatively larger price per unit of consumption. But in contrast, given the fact that the Mid west is the grain basket of United States of American explains for the relevance of Quantity, Produce, Cereal and Grains as the important factors in predicting the high value transactions. Products from these categories might been purchased in bulk / whole sale and then transported across to different states for further sales across those markets which accounts for higher quantity and higher transaction price.

***