Store Sales Prediction

A Big Data project to predict sales for the thousands of products sold at Favorita stores located in Ecuador.

University of Calabria
Big Data Course

UNIVERSITÀ
DELLA CALABRIA

UNIVERSITÀ DELLA CALABRIA

Store Sales Prediction

A Big Data project to predict sales for the thousands of products sold at Favorita stores located in Ecuador.

Student Name	Student Number
Andrea De Seta	227755
Lorenzo Guastalegname	223959
Luca Laboccetta	223968
Marco Bellizzi	223966

Instructor:	**Prof. Ricca Francesco**
Teaching Assistant:	**Ph.D. Mazzotta Giuseppe**
Institution:	**University of Calabria**
Faculty:	**Computer Science**
Source code:	https://github.com/lucalabo/BigData2022

Contents

1 Introduction **1**
1.1 Cluster setup . . . 1
1.2 Python environment . . . 3

2 Data Ingestion **4**
2.1 Data loading into MySQL . . . 4
2.2 Import into HDFS . . . 4
2.3 Import into Hive . . . 5

3 Data Understanding **6**
3.1 Data Structure . . . 6
3.2 Data Quality . . . 8
3.2.1 General statistics . . . 8
3.2.2 Results . . . 9
3.2.3 Missing values . . . 10
3.2.4 Results . . . 12
3.3 General Data analysis . . . 13

4 Data Cleaning and Transformation **16**
4.1 Product family prediction . . . 16
4.1.1 Mapper . . . 16
4.1.2 Reducer . . . 18
4.2 Cluster sales prediction . . . 18
4.2.1 Join Job . . . 18
4.2.2 GroupBy Job . . . 20

5 Modelling and Evaluation **22**
5.1 Prophet . . . 22
5.2 Prophet on Spark . . . 22
5.3 Model tuning . . . 22
5.4 Prophet metrics . . . 23
5.5 Results for families model . . . 23
5.6 Results for clusters model . . . 26
5.7 Running the model . . . 27

6 Conclusions **29**

References **30**

1 Introduction

In this project, we will predict sales for the thousands of product families sold at Favorita stores located in Ecuador. The training data, taken from kaggle[1], includes dates, stores and products informations, whether that item was being promoted, as well as the sales numbers.
The dataset is composed as follows:

- **train.csv** - The train dataset contains the time series, the sales column that we will predict and other useful data.
- **stores.csv** - It contains all the metadata like city, state and cluster.
- **oil.csv** - This dataset includes the daily oil price (from 2013 to 2017).
- **holiday-events.csv** - Holidays-events contains all the holidays and all the events, local and national.
- **Additional info** - We must pay attention to the wages for the public sector because the payments occurs twice a month and they could affect the sales. Also, a magnitude 7.8 earthquake struck Ecuador on April 16, 2016.

1.1. Cluster setup

The cluster in which the entire predictive task is carried out is composed in this way:

- **Master** - The machine is a Microsoft Azure Standard B2s with 2 virtual cpu, 4 GiB RAM memory and 30 GiB Disk memory running on Ubuntu 18.04.
- **Slaves** - Our cluster is composed by 6 workers based on AWS EC2 t2.micro machine with 1 virtual CPU, 1 GiB RAM memory and 20 GB Disk Space
- **Clients** - The client(s) are our PC. Connected to the cluster through SSH.

In every cluster machine we installed a few of services that we needed in order to do the various step regarding the project (Data Ingestion, Data Understanding, Data Cleaning and Transformation, Data Analysis, Modelling, Model Evaluation). These services are:

- **Hadoop Ecosystem v3.2.2**: The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

[1] https://www.kaggle.com/c/store-sales-time-series-forecasting

- **MySql**: MySQL is an open-source relational database management system (RDBMS). A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

- **Hive v2.3.9**: Apache Hive is a distributed, fault-tolerant data warehouse system that enables analytics at a massive scale. A data warehouse provides a central store of information that can easily be analyzed to make informed, data driven decisions. Hive allows users to read, write, and manage petabytes of data using SQL.

- **Sqoop v1.4.7**: Sqoop is a command-line interface application for transferring data between relational databases and Hadoop. It supports incremental loads of a single table or a free form SQL query as well as saved jobs which can be run multiple times to import updates made to a database since the last import.Using Sqoop, Data can be moved into HDFS/hive/hbase from MySQL/PostgreSQL/Oracle/SQL Server/DB2 and vice versa.

- **Spark v3.2.0**: Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching and optimized query execution for fast queries against data of any size. Simply put, Spark is a fast and general engine for large-scale data processing. The fast part means that it's faster than previous approaches to work with Big Data like classical MapReduce. The secret for being faster is that Spark runs on memory (RAM), and that makes the processing much faster than on disk drives. The general part means that it can be used for multiple things like running distributed SQL, creating data pipelines, ingesting data into a database, running Machine Learning algorithms, working with graphs or data streams, and much more.

1.2. Python environment

In the master, we configured a python environment based on conda[2], in order to run a jupyter server, and to install some libraries necessary for execution of the various tasks. The version installed is available at the following link[3]. After the installation, we can clone our environment with all necessary libraries, with the following command

```
conda env create bigdata -f environment.yml
conda activate bigdata
conda pack -f -o bigdata.tar.gz
```

The command conda pack, allows us to regroup all the package in a tar.gz and pass them all the workers in order to work with all the libraries needed.

[2] https://docs.conda.io/projects/conda/en/latest/index.html
[3] https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

2

Data Ingestion

2.1. Data loading into MySQL

First of all, we created the various table into MySql attending the format of the dataset given by kaggle. For this purpose, we writed and executed the following queries:

```
create table holidays_events( date_ date, type varchar(255), locale varchar(255),
                              locale_name varchar(255), description varchar(255),
                              transferred varchar(255) );

create table oil( date_ date, dcoilwtico varchar(255) );

create table stores( store_nbr integer, city varchar(255), state varchar(255),
                     type varchar(255), cluster integer );

create table test( id varchar(255) primary key, date date, store_nbr integer,
                   family varchar(255), onpromotion integer );

create table train( id varchar(255) primary key, date date, store_nbr integer,
                    family varchar(255), sales double, onpromotion intege r );

create table transactions( date date, store_nbr integer, transactions integer );
```

Then, in order to populate the tables created, we used the command LOAD DATA LOCAL INFILE for each csv files.

2.2. Import into HDFS

For the import of the data inside the HDFS (Hadoop Distribuited File System), first, we create a folder into our HDFS named ”store_sales” and then we used the command Sqoop Import as follows:

```
sqoop-import-all-tables --connect jdbc:mysql://master/store_sales --username
ubuntu -P --warehouse-dir store_sales --connect
```

2.3. Import into Hive

The final step of the Data Ingestion regarded the Hive Import; to do this, we executed, for each MySql table imported into the HDFS, the command external table in order to process them for the next steps with the following instructions:

```
create external table oil(date_oil date, dcoilwtico string)
row format delimited fields terminated by ',' lines terminated by '\n'
location '/user/ubuntu/store_sales/oil';

create external table holidays_events(date_ date, type string, locale string, locale_name string,
                                      description string, transferred string)
row format delimited fields terminated by ',' lines terminated by '\n'
location '/user/ubuntu/store_sales/holidays_events';

create external table sample_submission(id string, sales double, primary key(id) disable novalidate)
row format delimited fields terminated by ',' lines terminated by '\n'
location '/user/ubuntu/store_sales/sample_submission';

create external table stores(store_nbr integer, city string, state string, type string, cluster integer)
row format delimited fields terminated by ',' lines terminated by '\n'
location '/user/ubuntu/store_sales/stores';

create external table test(id string,date_ date,store_nbr integer, family string,
                           onpromotion integer,primary key(id) disable novalidate)
row format delimited fields terminated by ',' lines terminated by '\n'
location '/user/ubuntu/store_sales/test';

create external table train(id string,date_ date,store_nbr integer, family string,
                            sales double,onpromotion integer,primary key(id) disable novalidate)
row format delimited fields terminated by ',' lines terminated by '\n'
location '/user/ubuntu/store_sales/train';

create external table transactions(date_ date,store_nbr integer, transactions integer)
row format delimited fields terminated by ',' lines terminated by '\n'
location '/user/ubuntu/store_sales/transactions';
```

3

Data Understanding

3.1. Data Structure

The data that will be analyzed are provided from the various csv files given by the Kaggle competition ”*Store Sales - Time Series Forecasting*”.

holidays_events.csv

Feature name	Type	Description
Date	Nominal	Occurance date
Type	Nominal	Type of the event
Locale	Nominal	Type of the event in terms of location
Locale_name	Nominal	Name of the location
Description	Nominal	Name of the event
Transferred	Nominal	If the event is moved to another date

Table 3.1: holidays_events.csv Data Structure

oil.csv

Feature name	Type	Description
Date	Nominal	Occurance date
Dcoilwtico	Numeric	Daily price of the oil

Table 3.2: oil.csv Data Structure

stores.csv

Feature name	Type	Description
Store_nbr	Numeric	Unique identifier of a store
City	Nominal	City where the store is located
State	Nominal	State where the store is located
Type	Nominal	Type of store
Cluster	Numeric	Grouping of similar stores

Table 3.3: stores.csv Data Structure

test.csv

Feature name	Type	Description
Id	Numeric	Unique identifier of a sale
Date	Nominal	Date of a sale
Store_nbr	Numeric	Unique identifier of a store
Family	Nominal	Name of the product sold
Onpromotion	Numeric	Identifies an eventual promotion

Table 3.4: test.csv Data Structure

train.csv

Feature name	Type	Description
Id	Numeric	Unique identifier of a sale
Date	Nominal	Date of a sale
Store_nbr	Numeric	Unique identifier of a store
Family	Nominal	Name of the product sold
Sales	Numeric	Sale value
Onpromotion	Numeric	Identifies an eventual promotion

Table 3.5: train.csv Data Structure

train.csv Data Structure

Feature name	Type	Description
Date	Nominal	Date of the transaction
Store_nbr	Numeric	Unique identifier of a store
Transactions	Numeric	Number of transaction done

Table 3.6: transactions.csv Data Structure

3.2. Data Quality

3.2.1. General statistics

We did a simple Map Reduce aimed to obtain the general statistic from the train csv file in order to have a general look on the dataset and for the future prediction, described later in the documentation.
In the mapper we write in the context as key, the column name while as value the value of that row.
In the reducer we simply do the simple operations in order to obtain the min, max and average of the values previously written.

Map Reduce

Listing 3.1: General statistics Map Reduce

```
1
2  //MAPPER
3
4  public class ReadMapper extends Mapper<Object, Text, Text, Text>
5  {
6      @Override
7      protected void map(final Object key, final Text value, final Mapper<Object,
           Text, Text, Text>.Context context)
8              throws IOException, InterruptedException
9      {
10         // 0 id, 1 date, 2 store number, 3 family, 4 sales, 5 onpromotion
11         final String[] values = value.toString().split(","); //$NON-NLS-1$
12
13         context.write(new Text("id"), new Text(values[0])); //$NON-NLS-1$
14         context.write(new Text("store_number"), new Text(values[2])); //$NON-NLS-1
               $
15         context.write(new Text("sales"), new Text(values[4])); //$NON-NLS-1$
16         context.write(new Text("onpromotion"), new Text(values[5])); //$NON-NLS-1$
17     }
18 }
19
20 //REDUCER
21
22 public class StoreReducer extends Reducer<Text, Text, Text, Text>
23 {
24
25     @Override
26     protected void reduce(final Text key, final Iterable<Text> values, final
           Reducer<Text, Text, Text, Text>.Context context)
27             throws IOException, InterruptedException
28     {
29
30         int cont = 0;
31         Double min = null;
32         Double max = null;
33         Double average = 0.0;
34
35         for (final Text value : values)
36         {
37
38             final Double attribute = Double.valueOf(value.toString());
39
40             cont++;
41
42             if (cont == 1)
43             {
44                 min = attribute;
45                 max = attribute;
```

```
46                  }
47                  else
48                  {
49                      if (attribute < min)
50                      {
51                          min = attribute;
52                      }
53                      if (attribute > max)
54                      {
55                          max = attribute;
56                      }
57                  }
58
59                  average += attribute;
60
61              }
62
63              context.write(key, new Text(cont + "," + min + "," + max + "," + average /
                    cont)); //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$
64
65      }
66  }
```

3.2.2. Results

Feature name	Count	Min	Max	Avg
id	3000888	0.0	3000887	1500443.5
onpromotion	3000888	0.0	741.0	2.5872081863768326
sales	3000888	0.0	124717.0	355.1942998882502
store_number	3000888	1.0	54.0	27.5

Table 3.7: Statistics of numerical values of train set

3.2.3. Missing values

Map Reduce

In this Job Map Reduce we will calculate the number of missing values inside the following tables for each column:

- Holidays Events
- Transaction
- Oil
- Stores
- Test
- Train

Job configuration

In the Job configuration we can see that we read from the hdfs for each table,and each table will have his mapper.

Listing 3.2: Job configuration

```
Job j = Job.getInstance(conf,"MissingCountJob");
                    j.setJarByClass(MissingCountJob.class);
                    j.setReducerClass(MissingCountReducer.class);
                    j.setMapOutputKeyClass(Text.class);
                    j.setMapOutputValueClass(IntWritable.class);
                    MultipleInputs.addInputPath(j, new Path("store_sales/train"),
                            TextInputFormat.class, TrainMapper.class);
                    MultipleInputs.addInputPath(j, new Path("store_sales/stores"),
                            TextInputFormat.class, StoreMapper.class);
                    MultipleInputs.addInputPath(j, new Path("store_sales/oil"),
                            TextInputFormat.class, OilMapper.class);
                    MultipleInputs.addInputPath(j, new Path("store_sales/test"),
                            TextInputFormat.class, TestMapper.class);
                    MultipleInputs.addInputPath(j, new Path("store_sales/
                        transactions"),
                            TextInputFormat.class, TransactionMapper.class);
                    MultipleInputs.addInputPath(j, new Path("store_sales/
                        holidays_events"),
                            TextInputFormat.class, HolidaysEventsMapper.class);
                    FileOutputFormat.setOutputPath(j, new Path("count_null_values"
                        ));
                    j.waitForCompletion(true);
```

Mapper

We will see only one Mapper because each of them has the same behaviour,the difference is the table that the Mapper will read. The goal of the Mapper is to write inside the context as key the name of the Table and the name of the columns, as value we will only write one for increment the number of missing values,in case the value of the column is missing.

Listing 3.3: Mapper

```
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
    IntWritable>.Context context)
            throws IOException, InterruptedException {
        String[] fields=value.toString().split(",",-1);
        for(int i=0;i<fields.length;i++) {
            if(fields[i].isEmpty() || fields[i].equals(null)) {
                if(i==0)
                    context.write(new Text("HolidaysEvents_date"), new IntWritable
                        (1));
                else if(i==1)
                    context.write(new Text("HolidaysEvents_type"), new IntWritable
                        (1));
                else if(i==2)
                    context.write(new Text("HolidaysEvents_locale"), new
                        IntWritable(1));
                else if(i==3)
                    context.write(new Text("HolidaysEvents_localeName"), new
                        IntWritable(1));
                else if(i==4)
                    context.write(new Text("HolidaysEvents_description"), new
                        IntWritable(1));
                else if(i==5)
                    context.write(new Text("HolidaysEvents_transferred"), new
                        IntWritable(1));

            }
            else {
                if(i==0)
                    context.write(new Text("HolidaysEvents_date"), new IntWritable
                        (0));
                else if(i==1)
                    context.write(new Text("HolidaysEvents_type"), new IntWritable
                        (0));
                else if(i==2)
                    context.write(new Text("HolidaysEvents_locale"), new
                        IntWritable(0));
                else if(i==3)
                    context.write(new Text("HolidaysEvents_localeName"), new
                        IntWritable(0));
                else if(i==4)
                    context.write(new Text("HolidaysEvents_description"), new
                        IntWritable(0));
                else if(i==5)
                    context.write(new Text("HolidaysEvents_transferred"), new
                        IntWritable(0));
            }

        }
```

Reducer

The goal of the reducer will be only to increment the value of the missing values and write inside the context the total amount of missing values.

Listing 3.4: Reducer

```
protected void reduce(Text key, Iterable<IntWritable> values, Reducer<Text,
    IntWritable, Text, IntWritable>.Context context)
            throws IOException, InterruptedException {
        int total=0;
        for(IntWritable value : values)
        {
            total+=value.get();
        }
        context.write(key, new IntWritable(total));
    }
```

3.2.4. Results

Feature name	Missing values
HolidaysEvents_date	0
HolidaysEvents_description	0
HolidaysEvents_locale	0
HolidaysEvents_localeName	0
HolidaysEvents_transferred	0
HolidaysEvents_type	0
Oil_dateOil	0
Oil_dcoilwtico	43
Stores_city	0
Stores_cluster	0
Stores_state	0
Stores_storeNbr	0
Stores_type	0
Test_date	0
Test_family	0
Test_id	0
Test_onPromotion	0
Test_storeNbr	0
Train_date	0
Train_family	0
Train_id	0
Train_onPromotion	0
Train_sales	0
Train_storeNbr	0
Transaction_date	0
Transaction_storeNbr	0
Transaction_transactions	0

Table 3.8: Missing values

3.3. General Data analysis

A first analysis was made on the clusters: they represent various groups of similar stores; in short they can represent stores of the same type (i.e. grocery stores, electronics store, etc..).

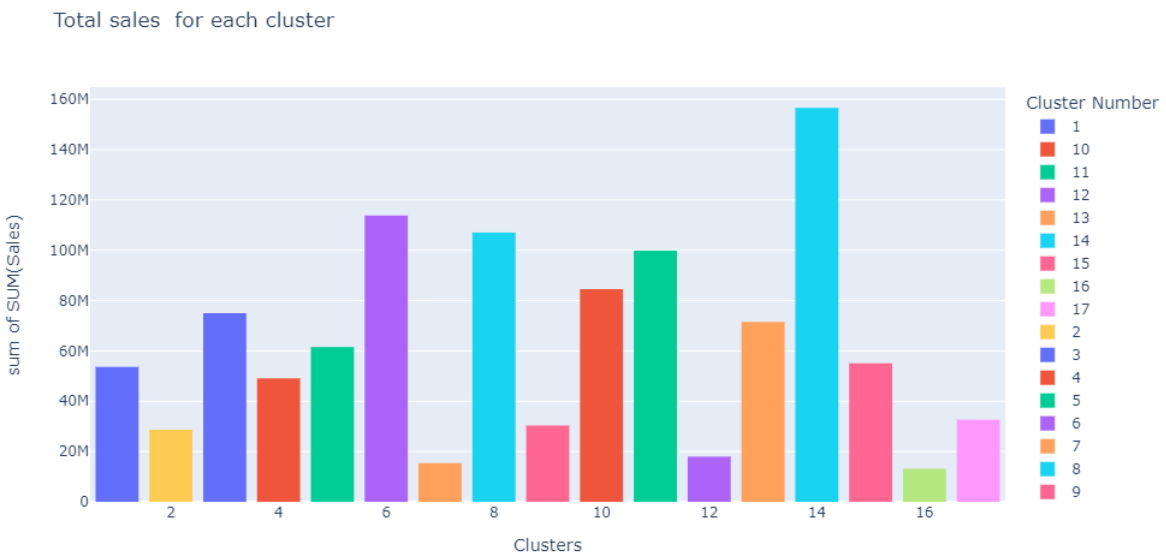

Figure 3.1: Total sales for each cluster

In the plot 3.1 we can see that the cluster of stores with the most total sales through all the years is the cluster number 14; we can assume that this cluster could be a cluster of stores regarding basic necessities as, for example, foods and drinks. The cluster 16 has the less sales of all clusters.

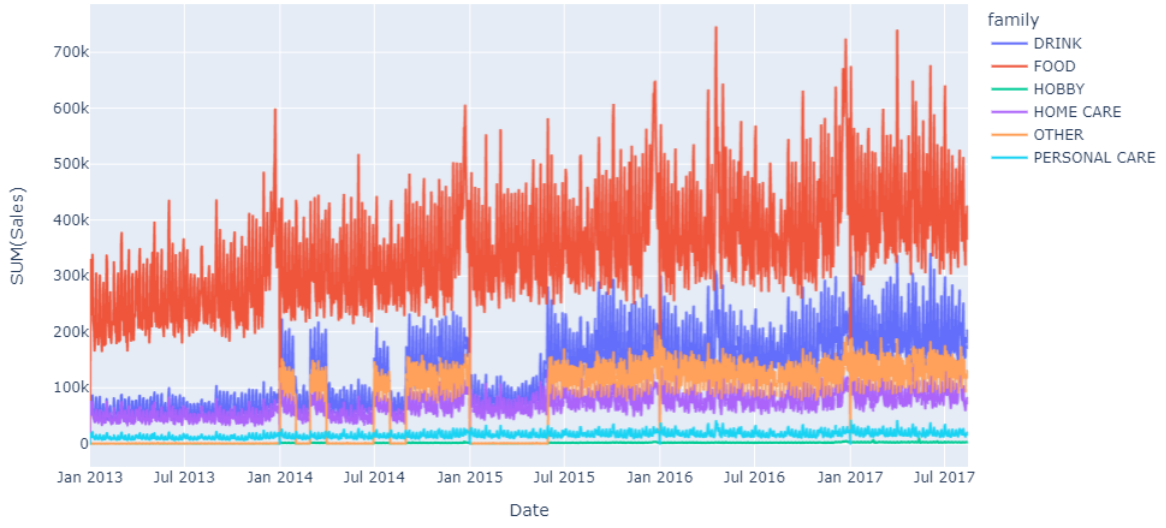

Figure 3.2: Total sales for each family

In order to confirm the previous analysis, we did another one represented in the plot 3.2 analyzing the total sales for each family. Originally there were 33 families that we regrouped in macro families in order to facilitate the various analysis and to avoid the overfitting (too see the implementation, look

the paragraph 4.1); in fact, the families with the most sales are food and drink and this could confirm our theory on the cluster number 14. After this analysis we decided to use the oil as regressor in the model.

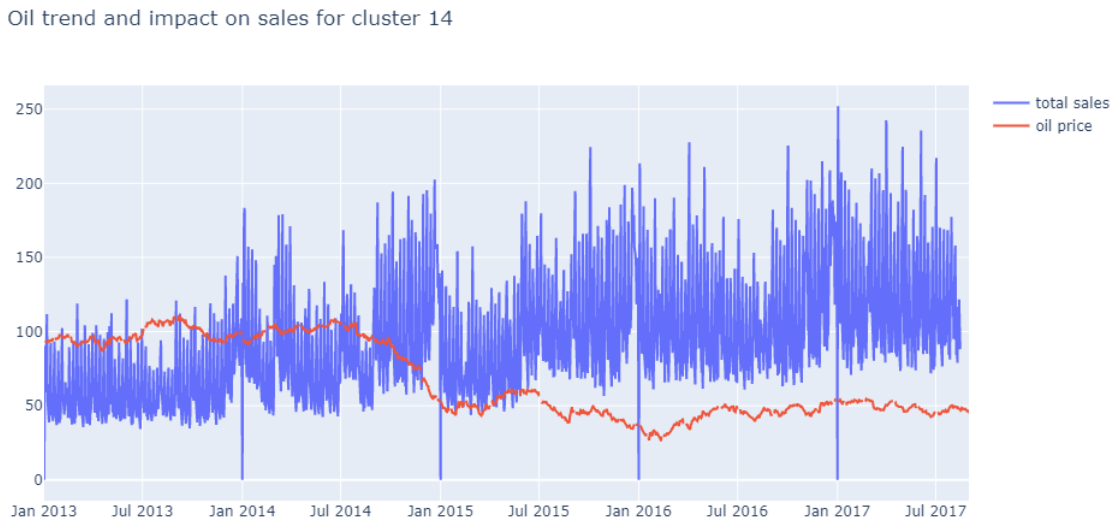

Figure 3.3: Oil trend and impact on sales for cluster 14

Another analysis made on the clusters is the impact of the oil on the sales (plot 3.3); we can observe that when the trend of the oil price went down, from the beginning of the third quarter of 2014, the sales increased; we have this effect because spending lesser in oil, the people is pushed to spend more in other products, in this case the products sold from the stores in the cluster 14.

captionTotal sales for each holiday

We also analyzed the sales during all the holidays events. We can notice that the majority of the

sales take place during the Carnival and the days after and before Christmas, concomitantly in the preparation of the party (Carnival case) or in the purchase of various gift (Christmas); while the days of Christmas and of New Year's Day, the sales are almost zero. We can assume that the sales are zero maybe because the stores are closed in these days.

4

Data Cleaning and Transformation

We decided to do two main prediction: one for the various families of the products and one for the clusters that represents the different kind of the stores. Each prediction is done about the sales during the years, precisely from 2013 to 2017.
Because of that, we did two different Data Cleaning, one for each prediction.

4.1. Product family prediction

The purpose of this task is to predict the forecast of the sum of the sales of various products families. To to this, we must apply some transformation to the original train set. First of all we must aggregate the families in some macro group, to avoid to go in overfitting. So we have aggregated the records and then we grouped them by date and family in order to obtain the sum of sales for each day and family. All of this has been done in MapReduce (see listing 4.1 and listing 4.2).

4.1.1. Mapper

The behaviour of the Mapper is simple. It receive the line of the csv file inside the HDFS.
Once the rows have been read, a check is made on the type of family to which the sale belongs and the key which will be the id of the row and the family to which it belongs. The values will be the rest of the columns.

Listing 4.1: Family Mapper

```
1
2  protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
       Text>.Context context)
3              throws IOException, InterruptedException {
4          String[] fields=value.toString().split(",");
5          String newVal="";
6          if(fields[3].equals("PREPARED FOODS") || fields[3].equals("GROCERY I") ||
               fields[3].equals("BREAD/BAKERY") || fields[3].equals("GROCERY II")
7                || fields[3].equals("SEAFOOD") || fields[3].equals("DELI") || fields
                     [3].equals("POULTRY") || fields[3].equals("FROZEN FOODS")
8                || fields[3].equals("EGGS") || fields[3].equals("MEATS") || fields
                     [3].equals("DAIRY")) {
9              newVal=fields[1].toString()+",FOOD,"+fields[4].toString()+","+fields
                   [5].toString();
10             context.write(new Text(fields[1]+",FOOD"),new Text(newVal));
11         }
12         else if(fields[3].equals("BEVERAGES") || fields[3].equals('"'+"LIQUOR")) {
13                 newVal=fields[1].toString()+",DRINK,"+fields[4].toString()+","+
                       fields[5].toString();
14                 context.write(new Text(fields[1]+",DRINK"),new Text(newVal));
15         }
16         else if(fields[3].equals("BOOKS") || fields[3].equals("CELEBRATION") ||
               fields[3].equals("MAGAZINES")
17                 || fields[3].equals("PLAYERS AND ELECTRONICS") || fields[3].equals
                       ("LAWN AND GARDEN")) {
18             newVal=fields[1].toString()+",HOBBY,"+fields[4].toString()+","+fields
                   [5].toString();
19             context.write(new Text(fields[1]+",HOBBY"),new Text(newVal));
20         }
21         else if(fields[3].equals("BABY CARE") || fields[3].equals("LINGERIE") ||
               fields[3].equals("BEAUTY")
22                 || fields[3].equals("LADIESWEAR") || fields[3].equals("PERSONAL
                       CARE") ) {
23             newVal=fields[1].toString()+",PERSONAL CARE,"+fields[4].toString()+","
                   +fields[5].toString();
24             context.write(new Text(fields[1]+",PERSONAL CARE"),new Text(newVal));
25         }
26         else if(fields[3].equals("HOME AND KITCHEN I") || fields[3].equals("HOME
               AND KITCHEN II") || fields[3].equals("HOME APPLIANCES")
27                 || fields[3].equals("HOME CARE") || fields[3].equals("CLEANING") )
                       {
28             newVal=fields[1].toString()+",HOME CARE,"+fields[4].toString()+","+
                   fields[5].toString();
29             context.write(new Text(fields[1]+",HOME CARE"),new Text(newVal));
30         }
31         else {
32             newVal=fields[1].toString()+",OTHER,"+fields[4].toString()+","+fields
                   [5].toString();
33             context.write(new Text(fields[1]+",OTHER"),new Text(newVal));
34         }
35
36
37     }
```

4.1.2. Reducer

The reducer will receive the new rows inside the object ***values*** built by the mapper, all grouped by the object ***key***, composed by id and family. The aim of the reducer is to make an aggregation by making the sum of the sales and promotion in order to write inside the object ***context*** the new row grouped by the new macro family.

Listing 4.2: Family Reducer

```
protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text,
    NullWritable>.Context context)throws IOException, InterruptedException {
        Float sales=0f;
        Float promotion=0f;
        String s="";
        for(Text value: values) {
            String[] fields=value.toString().split(",");
            promotion+=Float.parseFloat(fields[3].toString());
            sales+=Float.parseFloat(fields[2].toString());
            s=value.toString();
        }
        String[] fields=s.split(",");
        s=fields[0]+","+fields[1]+","+sales+","+promotion;
        context.write(new Text(s), null);

    }
```

4.2. Cluster sales prediction

Another interesting analysis could be to predict the total daily sales for each store cluster. Unlike the first task, here we do not have all the information directly in the train set, therefore, to take the daily sales of each cluster we have to do a join between train set and store informations. After joining, we need to group the records by date and cluster. Also this task is entirely made in MapReduce in the two jobs reported below.

4.2.1. Join Job

In the first Map Reduce of the Cluster prediction, we did a join between the train set and the store dataset that contains various informations about the stores including the "Cluster" column, the date that we used for the analysis.

Job configuration

We did a little modification compared to the canonical and simplest configuration of the job because here we have to read from two different directory inside the HDFS.So each table will have his personal Mapper.

```
Job j = Job.getInstance(conf,"JoinTrainStore");
j.setJarByClass(ClusterJob.class);
j.setReducerClass(CluterReducer.class);
j.setMapOutputKeyClass(Text.class);
j.setMapOutputValueClass(Text.class);
MultipleInputs.addInputPath(j, new Path("store_sales/train"),TextInputFormat.class
    , ClusterMapper.class);
MultipleInputs.addInputPath(j, new Path("store_sales/stores"),TextInputFormat.
    class, StoreMapper.class);
FileOutputFormat.setOutputPath(j, new Path("store_train_joined"));
j.waitForCompletion(true);
```

Store Mapper

The Store Mapper will receive the rows inside the csv file ”Store” and write inside the context object as key the ”Store Number” which will be the parameter of the join and as values the name of the Cluster that we want inside our new table joined and the name of the Table so that the reducer can understand what kind of line it is reading.

```
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
    Text>.Context context)
            throws IOException, InterruptedException {
        String[] fields=value.toString().split(",");
        context.write(new Text(fields[0].toString()),new Text("STORE,"+fields[4].
            toString()));

}
```

Train Mapper

The Train Mapper will receive the rows inside the csv file ”Train” and write inside the context object as key the ”Store Number” which will be the parameter of the join and as values the columns of the table Train that we want inside our new table joined, and the name of the Table so that the reducer can understand what kind of line it is reading.

```
@Override
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
    Text>.Context context)
        throws IOException, InterruptedException {
    String[] fields=value.toString().split(",");
    String val=fields[0].toString()+","+fields[1].toString()+","+fields[4].
        toString()+","+fields[5].toString();
    context.write(new Text(fields[2].toString()),new Text("TRAIN,"+val.toString())
        );

}
```

Reducer

The reducer will receive the new lines from the mappers and first of all read the name of the cluster where the first word of the value is ”Store” and for each store will append inside the rows of the train the name of the cluster. This is a Reduce-side Join.

```
@Override
protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text,
    NullWritable, Text>.Context context)
        throws IOException, InterruptedException {
    ArrayList<Text> store=new ArrayList<Text>();
    ArrayList<Text> train=new ArrayList<Text>();
    for(Text value : values) {
        String[] fields=value.toString().split(",");
        if(fields[0].equals("STORE")) {
            store.add(new Text(fields[1]));
        }
        else
            train.add(new Text(fields[1]+","+fields[2]+","+fields[3]+","+fields
                [4]));
    }
    for(Text st: store) {
        for(Text tr:train) {
            String[] fields=tr.toString().split(",");
            context.write(NullWritable.get(),new Text(tr.toString()+","+st.
                toString()));
        }
    }
}
```

4.2.2. GroupBy Job

In the second Map Reduce, that will be launched immediately after the previous job, we did a very similar work done with Family prediction Map Reduce. Here, we grouped the data by the number of cluster and also by the date doing an aggregation on the sum of the sales and the promotion.

Mapper

The Mapper will receive the lines of the csv inside the hdfs repository, that was built by the previous job map-reduce. Inside this Mapper we write inside the object context as key the date of the sale and the number of the cluster,and as values the sum and the promotion.

```
@Override
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
    Text>.Context context)
        throws IOException, InterruptedException {
    String[] fields=value.toString().split(",");
    String newVal=fields[2].toString()+","+fields[3].toString();
    context.write(new Text(fields[1].toString()+","+fields[4].toString()),new Text
        (newVal.toString()));

}
```

Reducer

The goal of the Reducer is to sum the value of the sales and of the promotion and insert them inside the object context also with the date and the number of the cluster grouped.

```
29 @Override
30 protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text,
       Text>.Context context)
31         throws IOException, InterruptedException {
32     Float sales=0f;
33     Float promotion=0f;
34     String s="";
35     String[] temp=key.toString().split(",");
36     for(Text value: values) {
37         String[] fields=value.toString().split(",");
38         promotion+=Float.parseFloat(fields[1].toString());
39         sales+=Float.parseFloat(fields[0].toString());
40     }
41     //String[] fields=s.split(",");
42     s=temp[0]+","+temp[1]+","+sales+","+promotion;
43     context.write(new Text(s), null);
44 }
```

5

Modelling and Evaluation

5.1. Prophet

Prophet[1] is open source software released by Facebook's Core Data Science team[1].
Prophet is a procedure for forecasting time series data based on an additive regression model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data.

5.2. Prophet on Spark

Spark is based on the concept of DataFrame. In simple words, the Dataframes API was the way from Spark creators to make it easy to work with Data in the framework. They are very similar to Pandas Dataframes or R Dataframes, but with several advantages. The first of course is that they can be distributed across a cluster, so they work with a lot of data, and the second one is that it's optimized.
Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters. Spark can be 100x faster than Hadoop for large scale data processing by exploiting in memory computing and other optimizations.

Since Prophet isn't available in MLlib, to run our algorithm on top of Spark we exploited the Pandas UDFs[2] (user defined functions). Pandas UDFs are user defined functions that are executed by Spark using Arrow to transfer data and Pandas to work with the data. In Spark are available two types of Pandas UDFs:

- **Scalar Pandas UDFs**: that are used for vectorizing scalar operations.
- **Grouped map Pandas UDFs**: these, first splits a Spark Dataframe into groups based on the conditions specified in the groupby operator (e.g. family, store cluster..). Then applies a function (in our case the model training) to each group. After that it will combines and returns the results as a new Spark Dataframe containing the results provided by each model.

5.3. Model tuning

We added to the model two important regressors that they have brought us big improvements:

- **Oil price** - Because Ecuador is an oil-dependent country and it's economical health is highly vulnerable to shocks in oil prices, in fact generally, when the oil goes down, the sales goes up.
- **Promotions** - Because the quantity of products on promotions can affect much the sales.

Prophet takes into account the holidays and how they can impact on sales. For example, during the carnival day, sales increase a lot, while on the contrary the first day of the year they tend to go to zero. So we added all national holidays to the model and, to these, we added the 15th and the last day of each month because wages in the public sector are paid every two weeks.

[1] https://facebook.github.io/prophet/
[2] https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.functions.pandas_udf.html

5.4. Prophet metrics

The statistics used to evaluate our models are:

- Mean Squared Log Error (MSLE)
- Root Mean Squared Log Error (RMSLE)
- Mean Absolute Error (MAE)
- Mean Absolute Percent Error (MAPE)

With a particular focus to the **RMSLE**.

5.5. Results for families model

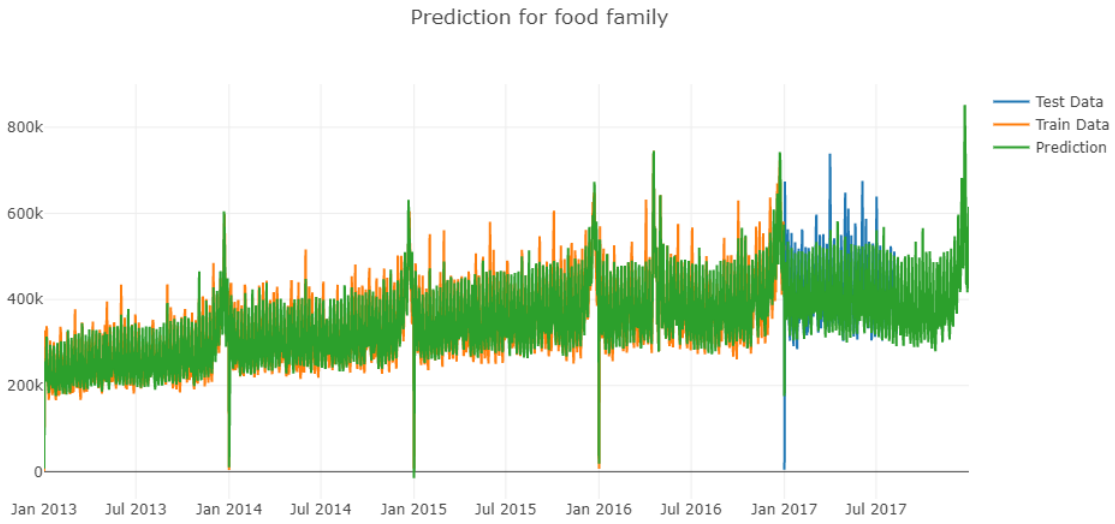

Figure 5.1: Food sales forecast

MSLE	MAE	MAPE	RMSLE
0.066	30395.096	0.227	0.258

Table 5.1: Metrics obtained for food sales forecast

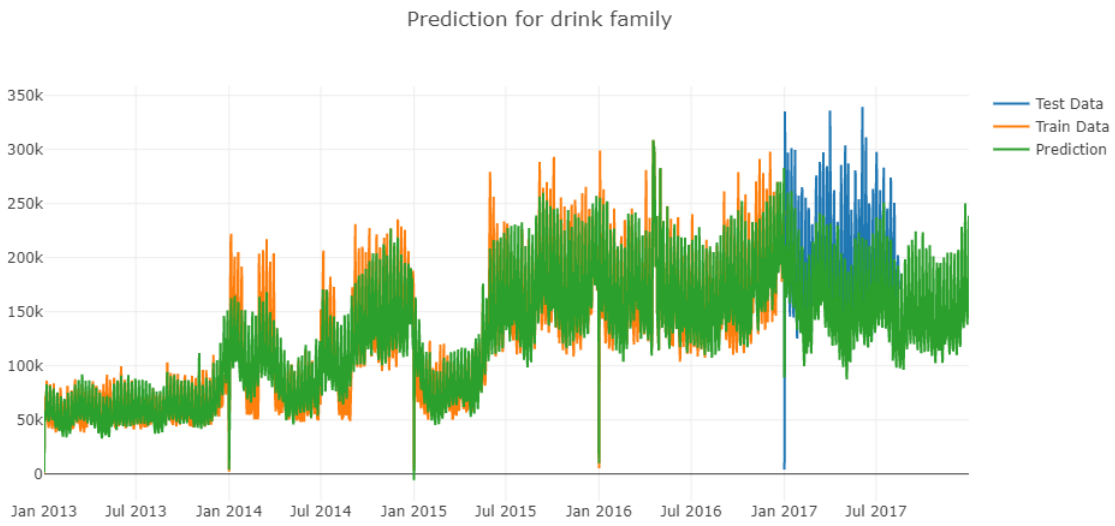

Figure 5.2: Drink sales forecast

MSLE	MAE	MAPE	RMSLE
0.095	35273.065	0.268	0.308

Table 5.2: Metrics obtained for drinks sales forecast

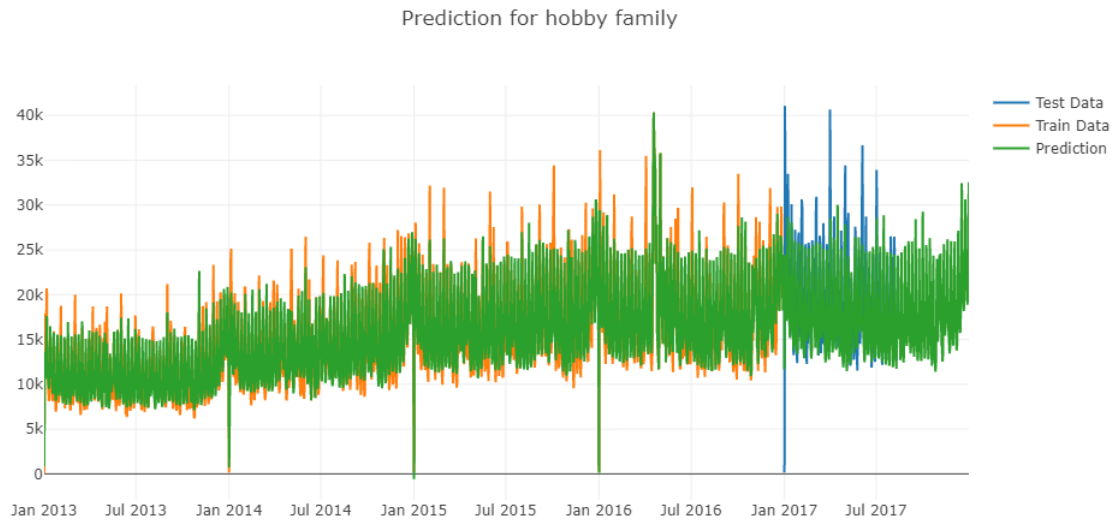

Figure 5.3: Hobby sales forecast

MSLE	MAE	MAPE	RMSLE
0.107	563.367	0.238	0.327

Table 5.3: Metrics obtained for hobbies sales forecast

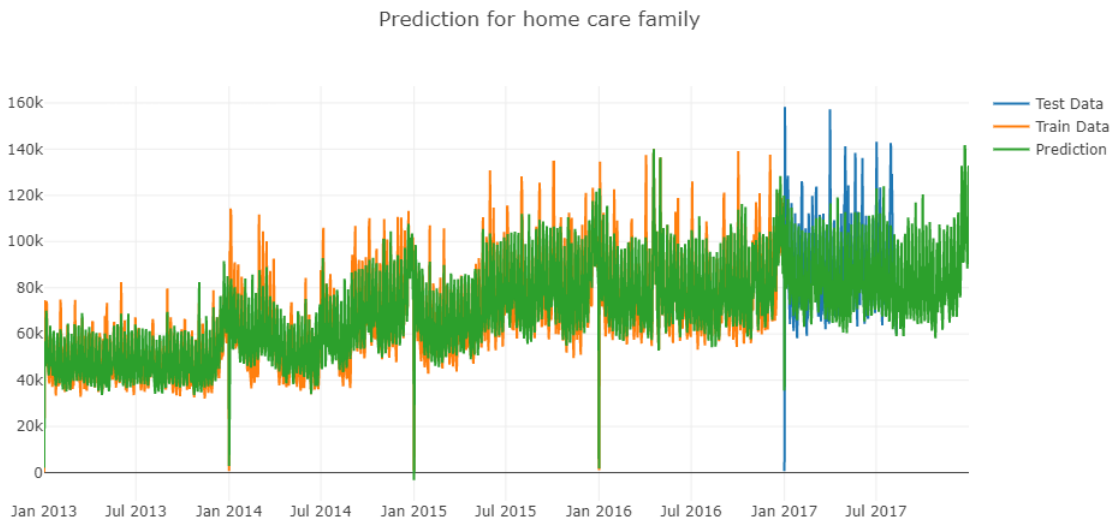

Figure 5.4: Home care sales forecast

MSLE	MAE	MAPE	RMSLE
0.090	8997.128	0.347	0.300

Table 5.4: Metrics obtained for home care sales forecast

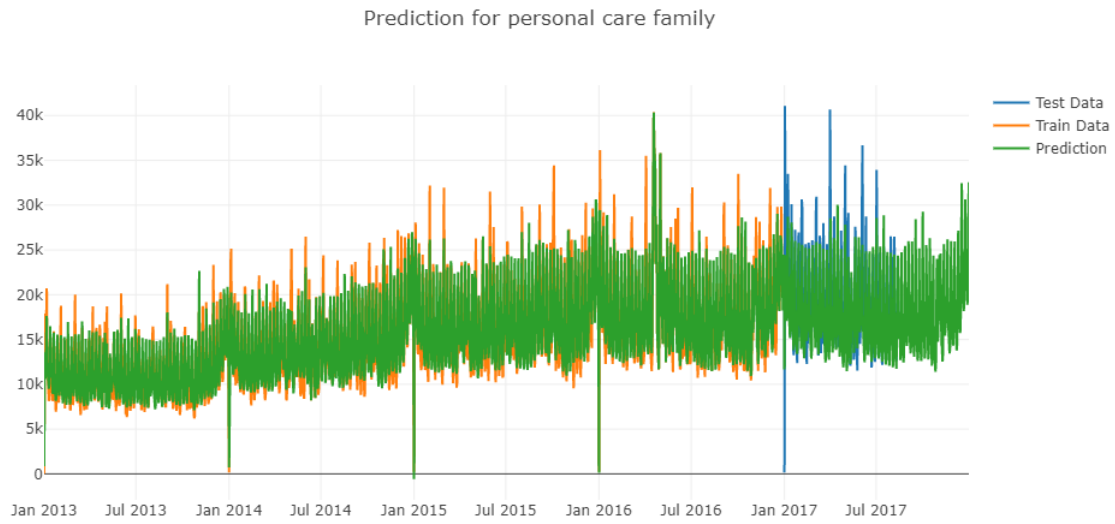

Figure 5.5: Personal care sales forecast

MSLE	MAE	MAPE	RMSLE
0.105	2283.591	0.445	0.324

Table 5.5: Metrics obtained for personal care sales forecast

5.6. Results for clusters model

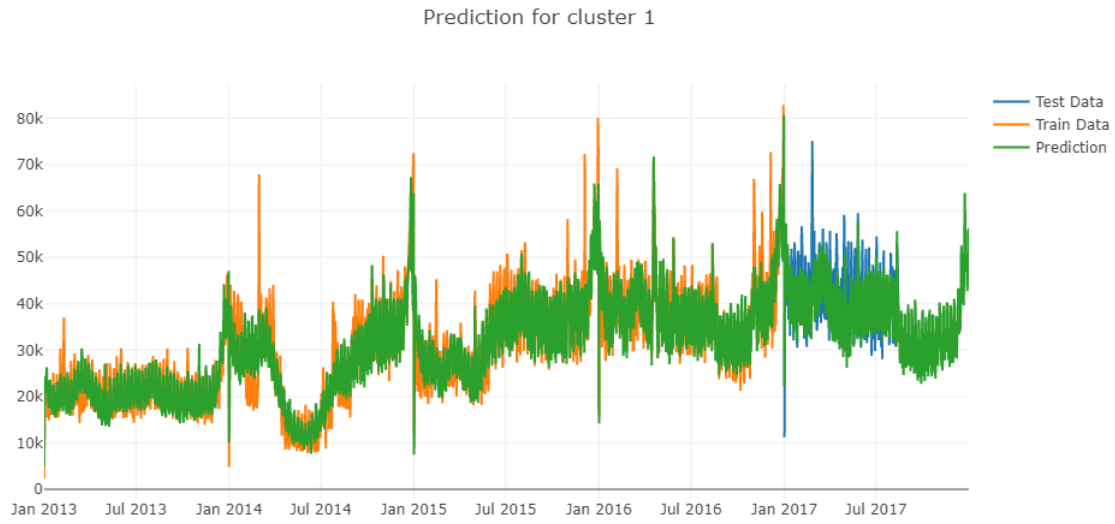

Figure 5.6: Cluster 1 sales forecast

MSLE	MAE	MAPE	RMSLE
0.012	3338.283	0.0793	0.113

Table 5.6: Metrics obtained for cluster 1 sales forecast

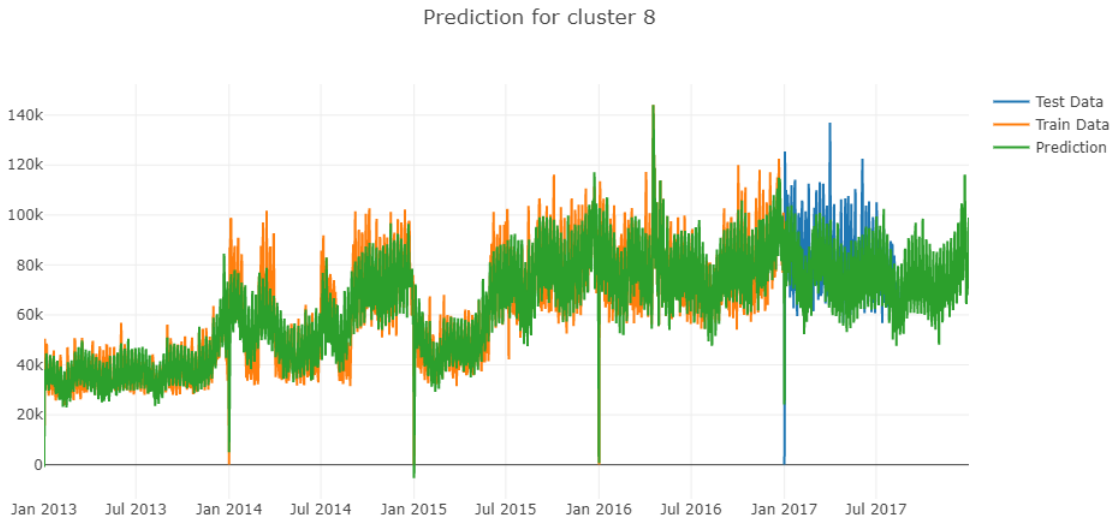

Figure 5.7: Cluster 8 sales forecast

MSLE	MAE	MAPE	RMSLE
0.463	7754.074	4.776	0.680

Table 5.7: Metrics obtained for cluster 8 sales forecast

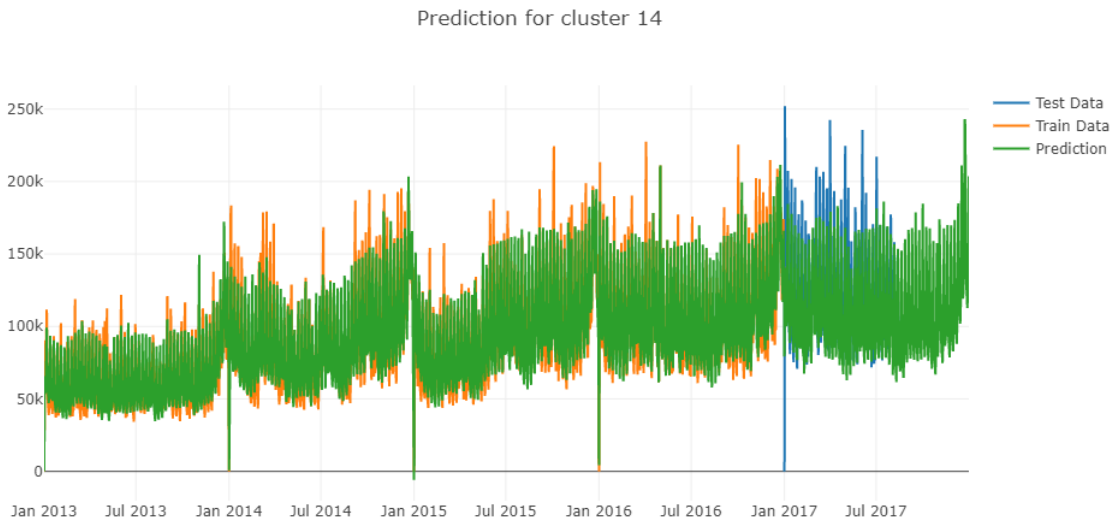

Figure 5.8: Cluster 14 sales forecast

MSLE	MAE	MAPE	RMSLE
0.582	14022.410	1.571	0.763

Table 5.8: Metrics obtained for cluster 14 sales forecast

5.7. Running the model

To run the model on the cluster, we can adopt two different strategy:

- spark-submit
- jupyter-notebook

To run with spark-submit:

```
export PYSPARK_DRIVER_PYTHON=`which python`
export PYSPARK_PYTHON=./environment/bin/python

conda activate bigdata

spark-submit --conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=./environment/bin/python
--master yarn --deploy-mode client --archives bigdata.tar.gz#environment
--num-executors 4 cluster_model.py
```

To run with jupyter-notebook:
In the master

```
conda activate bigdata
jupyter-notebook --no-browser --port 1234
```

In the client

```
ssh -N -f -L localhost:1111:localhost:1234 master
```

The go to localhost:1111 and run the notebook. Is recommended to execute the model on the jupyter-notebook in order to exploit the interactive graphs made with plotly. Or alternatively, run the model with spark submit and then read the model and plot the results via jupyter.

6

Conclusions

Regarding the prediction made on the families, after trained the models we can say in general we are satisfied for the results obtained. Families like FOOD (Figure 5.1) and HOME CARE (Figure 5.4) are suitable to do this prediction task and getting a look on their metrics, in particular the RMSLE that is one of the best metric for time series and that is suggested by Kaggle for this kind of competition, we can notice how they have very good values that are lower than the other families metrics.

Using the same criterion used for the previous analysis, we can comment the predictions made on the clusters sale, in particular the clusters 1, 8 and 14. Some clusters are indicated for this kind of analysis, for example the cluster number 1 which obtained the best RMSLE whit a score of 0.113 (look Table 5.6) and getting a look in depth in the notebook, we can notice that the difference between prediction and real data is minimal.

In general, the analysis made on the dataset could be extended to other kind of prediction, for example it can be done an analysis on the sales under the point of view of a specific city or region, also we can do analysis on the promotion during the holidays in order to understand when is convenient to do purchases and in which kind of store if we look the clusters. All these predictions can be made reusing the model that we used simply changing the type of the grouping.

References

[1] Sean J. Taylor and Benjamin Letham. “Forecasting at Scale”. In: *The American Statistician* 72.1 (Jan. 2018), pp. 37–45. URL: https://peerj.com/preprints/3190v2/.