SAPIENZA
UNIVERSITÀ DI ROMA

DETECTION OF NON-TECHNICAL ENERGY LOSSES IN POWER UTILITIES USING DATA MINING TECHNIQUES AND APACHE SPARK

Facoltà di Ingegneria dell'informazione, Informatica e Statistica

Dipartimento di Scienze Statistiche
Corso di laurea in Data Science

Emmanuele Conti

Matricola 1610955

Relatore

Ioannis Chatzigiannakis

Abstract

In the last years, the problem of detecting non-technical losses (NTL) in energy distribution systems have been investigated by the Electric Companies supported by the Academic research community.

The NTL investigation requires the analysis of the consumption data collected by the energy metering systems, on potentially millions of end users over a timeframe of several months or years. Moreover, it requires the execution of compute-intensive Machine Learning algorithms on the metering data.

The data collected from the metering systems have the characteristics to be considered Big Data, and the NTL problem is one of the toughest issues in the Energy Data Management, suitable to verify if the emerging technologies for Big Data Analysis provide advantages for their application in this field.

This work has experimented Apache Spark: Clustering Computing and MLlib Machine Learning library models to address a real-world NTL problem using a dataset with more than 1.6M customers, demonstrating that a new generation of Energy Data Management solutions can be efficiently implemented on this technology.

Index Terms - Electricity Theft Detection, Clustering, Apache Spark, Data Mining, Unsupervised Learning, Non-Technical Losses

Acknowledgements

IL SIGNORE È MIO PASTORE, NULLA MI MANCA.

SALMO 23,1

Primo su tutti voglio ringraziare il Signore perché veramente non mi ha fatto mancare nulla nella vita sostenendomi attraverso tante persone che voglio cercare di ringraziare tutte, partendo da chi mi è stato messo accanto solo nella parte finale di questo percorso arrivando a chi c'è stato fin dal principio.

Un grazie particolare va dunque a Simone e l'INNAAS e a tutte le persone che ci lavorano perché mi hanno messo a mio agio fin da subito. Un grazie particolare a miei due relatori esterni Francesco e Matteo.

Un grazie al professore Chatzigiannakis per la sua grande disponibilità e pazienza dimostrata per far fronte alle scadenze personali.

Voglio ringraziare tutti i colleghi che ho incontrato e che hanno in qualche modo aggiunto un mattoncino per arrivare a questo traguardo, in particolare Valerio un collega speciale nonché grande amico che mi ha dato l'aiuto più concreto di chiunque altro nell'ambito universitario e inevitabilmente anche personale grazie alla sua umiltà e pazienza.

Voglio ringraziare Dio per tutti i miei fratelli di comunità e le loro esperienze di vita e per la mia seconda famiglia per eccellenza, i miei amici quale fonte inesauribile di forze. Ognuno di loro che è stato fondamentale in questi 5 anni di studio intenso essendoci stati sempre e regalandomi la leggerezza e la semplicità di cui ha veramente bisogno uno studente universitario sotto pressione.

Infine, il ringraziamento senza dubbio più importante va alla base della montagna, la mia famiglia partendo dal nonno, passando per mamma e papà fino ad arrivare alle mie due splendide sorelle e i miei fenomenali 4 fratelli. Questa famiglia è senza dubbio il dono più prezioso e caro per me, una famiglia splendida che mi ha sempre amato e regalalo una fiducia sconfinata che spero abbia ripagato in parte con la mia vita e in parte anche grazie a questa laurea che conclude un ciclo di vita.

Table of Contents

List of Tables

List of Figures

1. Introduction

Our modern society and daily activities strongly depend on the availability of electricity. Electrical power grids allow to distribute and deliver electricity from generation infrastructures such as power plants or solar cells to customers such as residences or factories. Electrical power grids are the backbone of today's society. Losses during generation and distribution cause major problems, including financial losses to electricity providers, and a decrease of stability and reliability. One frequently appearing problem are losses in power grids, namely the difference between the generated or bought energy and the billed ones, can be divided into two distinct categories: technical and non-technical losses.

According also to [1] the former is related with problems in the system through the physical characteristics of the equipment, that is, the technical losses are the energy lost in the transport, the transformation and the equipment of measurement, becoming a very high cost to the electric power companies. The non-technical losses are those associated with the commercialization of the supplied energy to the user, and refer to the delivered and not billed energy resulting in a loss in the profits. They are also defined as the difference between the total losses and the technical losses, being strongly related to illegal connections in the distribution system.

In recent years, the problem of detecting non-technical losses in distribution systems has been paramount. Theft and adulteration of power meters, with the purpose to modify the measurement of the energy consumption, are the main causes that lead to non-technical losses in power companies. Since then to perform periodic inspections to minimize such frauds may be very expensive, it is a hard task to calculate or measure the amount of losses, and in most cases, it is almost impossible to know where they occur. Aimed at reducing fraud and energy theft, several electric power companies have been concerned that the illegal connections should be better profiled. Electric utilities will never be able to eliminate fraud, but more realistically, minimization of such losses may guarantee investments in energy quality programs, as well as enable a reduction in its price to the consumer. Currently, some improvement in this area can be observed with

the use of various artificial intelligence techniques to automatically identify non-technical losses, which are a real application in Smart Grids. Despite the widespread use of machine learning techniques for the identification of non-technical losses in power systems, the problem of selecting the most representative features has not been widely discussed in the context of nontechnical losses.

Technical losses:

- Copper losses those are due to I2R losses that are inherent in all inductors because of the finite resistance of conductors
- Dielectric losses that are losses that resulting from the heating effect on the dielectric material between conductors
- Induction and radiation losses, produced by the electromagnetic fields surrounding conductors. Technical losses computable and controllable, provided the power system in question consists of known quantities of loads. The following are the causes of technical losses:
  - Harmonics distortion
  - Improper earthing at consumer end
  - Long single-phase lines
  - Unbalanced loading
  - Losses due to overloading and low voltage
  - Losses due to a poor standard of equipment.

Non-Technical Losses

- Tampering with meters to ensure the meter recorded a lower consumption reading
- Errors in technical losses computation
- Tapping (hooking) on LT lines
- Arranging false readings by bribing meter readers
- Stealing by bypassing the meter or otherwise making illegal connections
- By just ignoring unpaid bills
- Faulty energy meters or un-metered supply
- Errors and delay in meter reading and billing
- Non-payment by customers.

According to [2] there are also two types of adversaries. The first type is the usual customers who are using the meters, let is call them inside adversaries. Inside adversaries may have some knowledge of smart meters, and thus they can tamper

these meters to lower their electricity bills. Or they may know nothing about smart meters, but they can obtain hacking tools to tamper meters for free [3]. The second type of adversaries are from the outside. Let is call them outside adversaries. Outside adversaries can manipulate meters remotely or manipulate billing messages in communication networks. They could increase the electricity bills as well as decrease them (however increasing the electricity bills is another type of frauds which is out of the scope of this thesis). In message manipulation attacks, the meters are intact. However, the adversaries must intercept the connections between meters and the head-end system to obtain encryption keys. Therefore, the utility company must still locate the meters to replace their keys. Under the aforementioned consideration, a meter is called tampered when it is either message-manipulated or tampered. The inside adversaries could falsify power consumption and attribute it to the neighbors. The prerequisite of this type of attack is the same as message manipulation

Investigations are undertaken by electric utility companies to assess the impact of technical losses in generation, transmission and distribution networks, and the overall performance of power networks [4,5]. Nontechnical losses (NTLs) comprise one of the most important concerns for electricity distribution utilities worldwide. In 2004, Tenaga Nasional Berhad (TNB), the sole electricity provider in peninsular Malaysia recorded revenue losses as high as U.S.$229 million a year as a result of electricity theft, faulty metering, and billing errors [6]. NTLs faced by electric utility companies in the United States was estimated between 0.5% and 3.5% of the gross annual revenue [7], which is relatively low when compared to losses faced by electric utilities in developing countries such as Bangladesh [8], India [9] and Pakistan [10]. Nevertheless, the loss is amounted between U.S.$1 billion and U.S.$10 billion given that utility companies in the U.S. had revenues around U.S.$280 billion in 1998 [7]. Due to the problem associated with NTLs in electric utilities [11] methods for efficient management of NTLs [12], protecting revenue in the distribution industry [13], [14] and detecting fraud electricity consumers [15] have been proposed. The most effective method to reduce NTLs and commercial losses up to date is by using intelligent and smart electronic meters that make fraudulent activities more difficult, and easy to detect [14]. From

an electrical engineering perspective, one method to detect losses is to calculate the energy balance reported in [33], which requires topological information of the network. In emerging economies, which are of particular interest due to their high NTL proportion, this is not realistic for the following reasons: (i) network topology undergoes continuous changes to satisfy the rapidly growing demand of electricity, (ii) infrastructure may break and lead to wrong energy balance calculations and (iii) it requires transformers, feeders and connected meters to be read at the same time.

In order to detect NTLs, inspections of customers are carried out, based on predictions whether there may be an NTL. The inspection results are then used in the learning of algorithms to improve predictions. However, carrying out inspections is expensive, as it requires the physical presence of technicians. It is therefore important to make accurate predictions to reduce the number of false positives.

In recent years, several data mining and research studies on fraud identification and prediction techniques have been carried out in the electricity distribution sector. These include statistical methods [16,17,18,19]; decision trees [20,21]; artificial neural networks (ANNs) [18,22,23]; knowledge discovery in databases (KDD) [23,24,25,26]; clustering techniques [26,27,28]; Support Vector Machine [29]; and multiple classifiers using cross-identification and voting schemes [30]. Among these methods, load profiling is one of the most widely used [31] approaches, which is defined as the pattern of electricity consumption of a customer or group of customers over a period [32].

Detecting NTLs is challenging because of the wide range of possible causes of NTLs, such as different fraudulent types of customers.

The challenge of supervised learning for anomaly detection. It must be noted that most NTL detection methods are supervised. Anomaly detection - a superclass of NTL - is generally challenging to learn in a supervised manner for the reasons stated in [34]: (i) anomaly datasets contain a very small number of positive examples and a large number of negative examples, resulting in imbalanced classes, (ii) it is used for many different kinds of anomalies as it is hard for any algorithm to learn from just a few positive examples what the anomalies might

look like and (iii) there may be also future anomalies which may look completely different to any of the anomalous examples learned so far. In contrast, supervised learning works best for (i) large numbers of both positive and negative examples, (ii) when there are enough positive examples so that the algorithm can get a sense of what positive examples might look like and (iii) future positive examples are likely to be similar to the ones in the training set.

1.1 Literature review

Literature review NTL detection can be treated as a special case of fraud detection, for which a general survey is provided in [35].

One method to detect NTLs is to analyze the customer load profile using artificial intelligence methods, such as machine learning or expert systems. Support Vector Machines (SVM) are used in [29], working on daily average consumption features of the last 24 months for less than 400 highly imbalanced training examples, ignoring the class imbalance in the results reported. That work is combined with fuzzy logic [36] or genetic algorithms [37], focusing on an optimization of the SVM output. A rule-based expert system outperforms a SVM in [38] for an unknown number of customers, focusing on high performance implementations. Fuzzy logic following C-means fuzzy clustering is applied to a dataset of ~20K customers in [26]. Furthermore, neural networks using handcrafted features calculated from the consumption time series plus customer-specific pre-computed attributes are used in [39] for ~1K balanced customers. Applying smart half-hour meter readings of three weeks of ~6K customers are fed into a neural network in [40]. Optimum-path forest are applied to NTL detection in [41] for ~10K customers outperforming different SVMs and a neural network.

1.2 Contribution of this thesis

In this thesis, we focus on a large dataset comprising of ~1M records spanning two years of consumption data and apply different NTL detection methods on this real dataset of a big energy utilities. In this context this thesis presents a proposal for identifying suspect profiles of energy consumption compared to regular

consumption profiles with Apache Spark on Databricks[1]. In this work, we try to formulate a model for an end-to-end Big Data analytics platform based on these technologies, that can ingest data from heterogeneous sources, process it in an efficient way, mine the data to generate insights based on business logic and then present the information using interactive visualizations. The proposed approaches define a framework to determine a list of irregular consumption in order to find any fraud. Thereafter a list of consumers classified as fraudsters is generated to help perform the costly inspections with the main objective being the improvement of the hit rate of the inspections to reduce unnecessary operational cost. As seen before, there are several types of fraud that can occur, but this research only concentrates on scenario when abrupt changes appear in customer load profiles, which indicate possible fraud events.

1.3 Structure of this thesis document

The rest of this thesis is organized as follows: in Chapter 2 will be discussed about Big Data Analysis and platforms and includes the implementation of the mentioned Big Data platform Databricks to perform the analyses on real-life use cases and generate useful insights. Chapter 3 provides really fine details on data structure. In Chapter 4 will be presented the framework used for preparing the data to the analysis, which include: filtering and selection of customers and features, identifying of new features and data normalization. Chapter 5 describes different proposed NTL detection models on the data of different dataset, Chapter 6 presents the results of this work and provides an outreach on future work.

[1] https://databricks.com/

2. Big Data

Big Data describes a massive collection of structured and unstructured information that has been collected over the past few years [46]. This data can be used for analysis and the discovery of information that was not available, or even possible, just a few years ago. This collection of information will transform the way to make business and understand the market by analyzing available data and using that information to improve their business processes. The term Big Data was first used to refer to increasing data volumes in the mid-1990s. In 2001, Doug Laney, then an analyst at consultancy Meta Group Inc., expanded the notion of Big Data to also include increases in the variety of data being generated by organizations and the velocity at which that data was being created and updated. Those three factors Volume, Velocity and Variety became known as the 3Vs of Big Data, a concept Gartner popularized after acquiring Meta Group and hiring Laney in 2005.

- Volume: Is currently know that the exponential growth in the data storage. Is possible to find data in the format of videos, music and large images on our social media channels. It is very common to have Terabytes and Petabytes of the storage system for enterprises. As the database grows, the applications and architecture built to support the data needs to be reevaluated quite often. Sometimes the same data is re-evaluated with multiple angles and even though the original data is the same the new found intelligence creates an explosion of the data. The big volume indeed represents ***Big Data***.
- Velocity: The data growth and social media explosion have changed how the data is looked at. There was a time when was believe that data of yesterday is recent. However, news channels and radios have changed how fast they receive the news. Today, people rely on social media to update them with the latest happenings. On social media sometimes a few seconds is enough to classify a message as old (a tweet, status updates etc.) and then something in which users are interested anymore. The data movement is now almost real time and the update window has reduced to fractions of the seconds. This high-velocity data represents ***Big Data***.

- Variety: Data can be stored in multiple formats. For example: database, Excel, CSV, Access or, related to the context, it can be stored in a simple text file. Sometimes the data is not even in the traditional format as were assumed, it may be in the form of video, SMS, pdf or something different. The organization needs to arrange it and make it meaningful. It will be easy to do so if a user has data in the same format, however, it is not the case in most of the time. The real world has data in many different formats and that is the challenge that needs to be addressed with the *Big Data*. This variety of the data represent ***Big Data***.

Fig 2.1 gives an idea of Big Data with some values and Fig 2.2 shows a more recent description of Big Data, in this image taken from [46], the number of V's is increased up to 8.

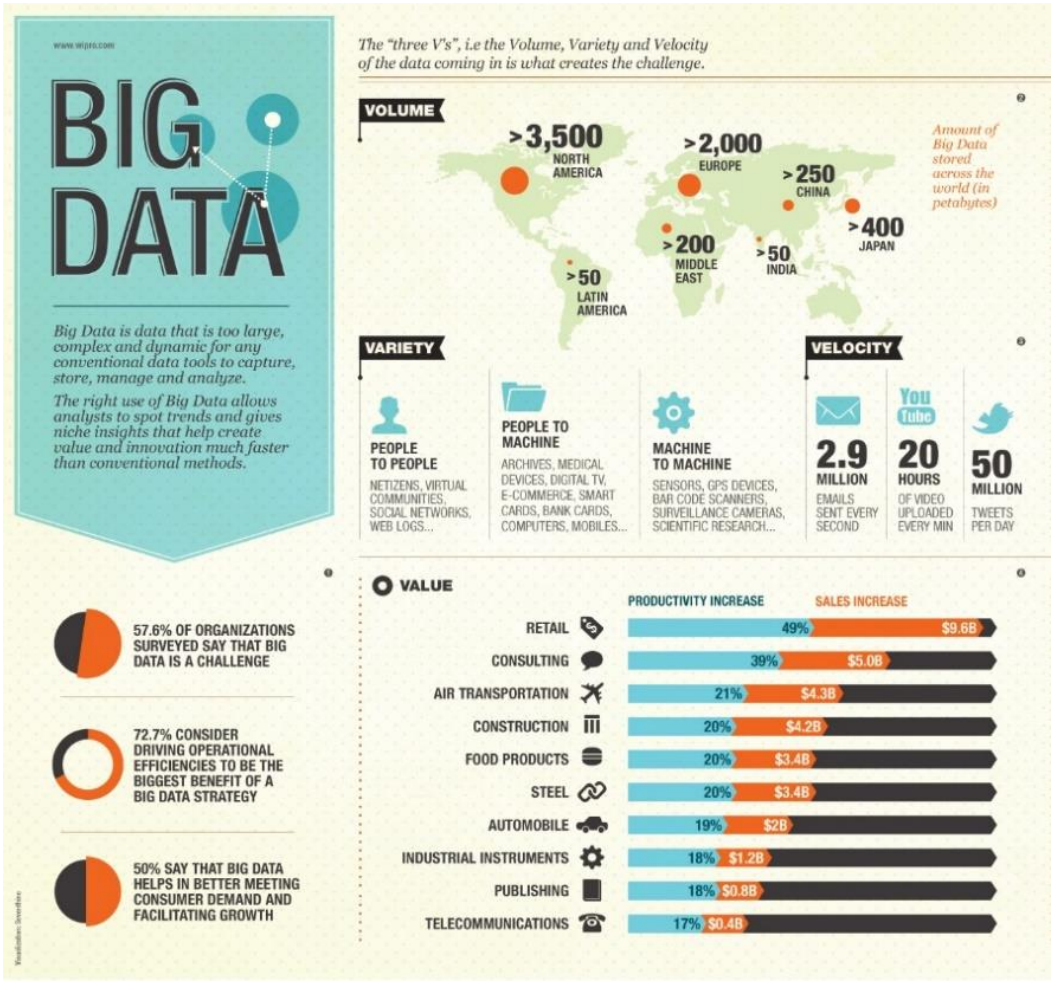


Figure 2-1 The world of Big Data [44]

Figure 2-2 Big Data 8 V' [46]

Separately, the Hadoop[2] distributed processing framework was launched as an Apache open source project in 2006, planting the seeds for a clustered platform built on top of commodity hardware and geared to run Big Data applications. By 2011, Big Data analytics began to take a firm hold in organizations and the public eye, along with Hadoop and various related Big Data technologies that had sprung up around it. Data Analytics talked about many situations where Big Data comes into play. For example Big Data has a strong presence in the food industry, providing businesses with information about customer's activity, likes, and preferences for different items. As instance, McDonald's uses data analytics to figure out what is going on in their stores. They use this information to optimize

[2] http://hadoop.apache.org/

different aspects of their business, such as the drive-thru. McDonald's designs the drive-thru around three different items from Big Data: design, information provided, and the different types of people who order from the drive-thru. By analyzing this information, McDonald's can cater to an even broader crowd then they currently do, making sure that they are doing the right thing not only for their business but also for their customers. Instances like this allow companies to really get the information that they need from customers, rather than playing a guessing game. Big Data will change the way businesses conduct their practices, and it has already begun.

2.1 Big Data Analysis

Big Data analytics is the process of examining Big Data to uncover hidden patterns, unknown correlations, market trends, customer preferences and other useful information that can help organizations make more-informed business decisions.

Driven by specialized analytics systems and software, Big Data analytics can point the way to various business benefits, including new revenue opportunities, more effective marketing, better customer service, improved operational efficiency and competitive advantages over rivals. Big Data analytics applications enable data scientists, predictive modelers, statisticians and other analytics professionals to analyze growing volumes of structured transaction data, plus other forms of data that are often left untapped by conventional business intelligence (BI) and analytics programs. That encompasses a mix of semi-structured and unstructured data, for example, internet clickstream data, web server logs, social media content, text from customer emails and survey responses, mobile-phone call-detail records and machine data captured by sensors connected to the internet of things.

On a broad scale, data analytics technologies and techniques provide a means of analyzing data sets and drawing conclusions about them to help organizations make informed business decisions. Big Data analytics is a form of advanced analytics, which involves complex applications with elements such as predictive

models, statistical algorithms and what-if analyses powered by high-performance analytics systems.

Initially, as the Hadoop ecosystem took shape and started to mature, Big Data applications were primarily used by the largest internet and e-commerce companies, such as Yahoo, Google and Facebook, as well as analytics and marketing services providers. In ensuing years, though, Big Data analytics has increasingly been embraced by retailers, financial services firms, insurers, healthcare organizations, manufacturers, energy companies and other mainstream enterprises.

Unstructured and semi-structured data types typically do not fit well in traditional data warehouses that are based on relational databases oriented to structured data sets. Furthermore, data warehouses may not be able to handle the processing demands posed by sets of Big Data that need to be updated frequently, or even continually, as in the case of real-time data on stock trading, the online activities of website visitors or the performance of mobile applications.

In some cases, Hadoop clusters and NoSQL systems are being used primarily as landing pads and staging areas for data before it gets loaded into a data warehouse or analytical database for analysis, usually in a summarized form that is more conducive to relational structures.

More frequently, however, Big Data analytics users are adopting the concept of a Hadoop data lake that serves as the primary repository for incoming streams of raw data. In such architectures, data can be analyzed directly in a Hadoop cluster or run through a processing engine like Spark. As in data warehousing, sound data management is a crucial first step in the Big Data analytics process. Data stored in the Hadoop Distributed File System must be organized, configured and partitioned properly to get good performance on both extract, transform and load (ETL) integration jobs and analytical queries.

Once the data is ready, it can be analyzed with the software commonly used in advanced analytics processes. That includes tools for data mining, which sift through data sets in search of patterns and relationships; predictive analytics, which build models for forecasting customer behavior and other future

developments; machine learning, which tap algorithms to analyze large data sets; and deep learning, a more advanced offshoot of machine learning. Text mining and statistical analysis software can also play a role in the Big Data analytics process, as can mainstream BI software and data visualization tools. For both ETL and analytics applications, queries can be written in batch-mode MapReduce; programming languages, such as R, Python and Scala; and SQL, the standard language for relational databases that is supported via SQL-on-Hadoop technologies.

Big Data analytics applications often include data from both internal systems and external sources, such as weather data or demographic data on consumers compiled by third-party information services providers. In addition, streaming analytics applications are becoming common in Big Data environments, as users look to do real-time analytics on data fed into Hadoop systems through Spark's Streaming module or other open source stream processing engines, such as Flink and Storm.

Early Big Data systems were mostly deployed on-premises, particularly in large organizations that were collecting, organizing and analyzing massive amounts of data. But cloud platform vendors, such as Amazon Web Services (AWS) and Microsoft, have made it easier to set up and manage Hadoop clusters in the cloud, as have Hadoop suppliers such as Cloudera and Hortonworks, which support their distributions of the Big Data framework on the AWS and Microsoft Azure clouds. Users can now spin up clusters in the cloud, run them for as long as needed and then take them offline, with usage-based pricing that does not require ongoing software licenses.

Potential pitfalls that can trip up organizations on Big Data analytics initiatives include a lack of internal analytics skills and the high cost of hiring experienced data scientists and data engineers to fill the gaps. The amount of data that is typically involved, and its variety, can cause data management issues in areas including data quality, consistency and governance; also, data silos can result from the use of different platforms and data stores in a Big Data architecture. In addition, integrating Hadoop, Spark and other Big Data tools into a cohesive architecture that meets the Big Data analytics necessities of an organization is a

challenging proposition for many IT and analytics teams, which have to identify the right mix of technologies and then put the pieces together.

2.2 Platforms

An essential component of a Big Data platform [43] is the process that enables the ingestion, storage and management of data, and Hadoop is a major open-source framework which helps achieve this, an example of an underlying architecture is showed Fig 2.3.

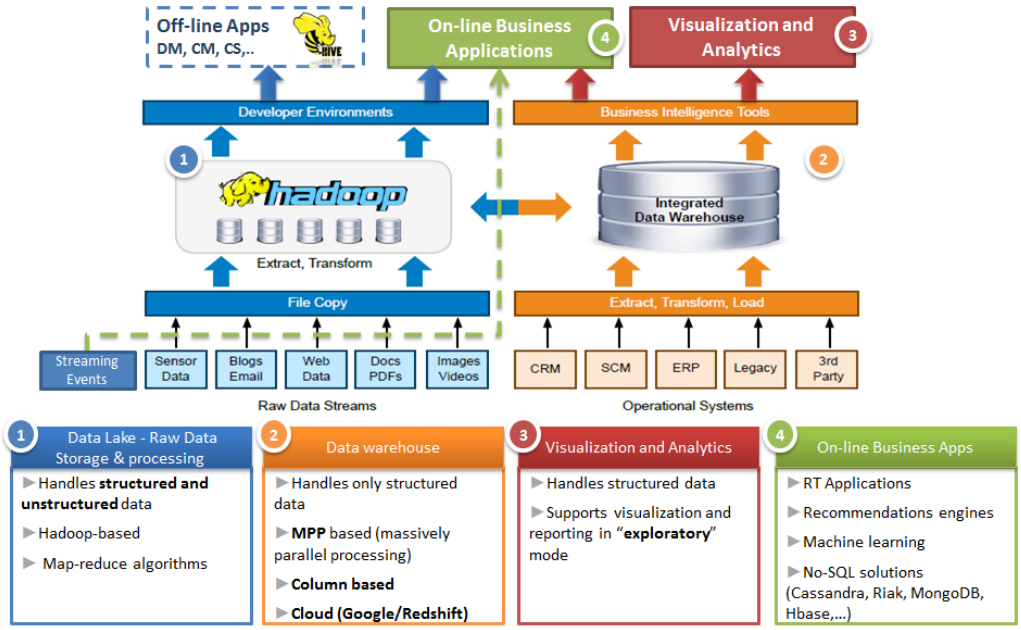


Figure 2-3 Architecture of Hadoop

It supports the processing and storage of extremely large datasets in a distributed computing environment. Hadoop's architecture basically involves cluster planning, i.e. dedicating multi-core CPUs with RAM and HDDs of heavy configurations to facilitate ingression via two main approaches: batch and event-driven. The former is appropriate for file and structured data, while the latter is appropriate for most near-real-time events such as logs, transactional or sensor data. Hadoop's MapReduce does an excellent job in processing batch events, but its efficiency is reduced while processing real-time streams. To compensate for this, it can be used Apache Spark or Storm along with Hadoop, since they are naturally meant for real-time processing. The storage of this processed data is

done in either HDFS or HBase, and both are highly performant databases with fast read and write capabilities.

Since Hadoop processes in a parallel distributed manner, a central infrastructure is required for cross-node synchronization. A ZooKeeper[3] server does that job efficiently, by keeping a copy of the state of the entire system and persisting this information in local log files. Hadoop also provides access control in the form of Information architecture, i.e. a concise access schema that controls tightly who has access to what data and is very helpful when a cluster is shared across departments. On an enterprise level, the continuously generated data far exceeds the limits of our ability to store, process, analyze and transmit it, and this situation is causing stress on all the underlying infrastructure used to generate and process it.

This shortcoming can be taken care of by employing cloud-based large-scale distributed compute and storage infrastructures. It helps enable, either the manual setup of Hadoop and other computing engines like Storm[4] and Spark in a VM or provide these capabilities as services out-of-the-box with automatic scalability of the arrangement as per the usage. These out-of-the-box services have been heavily adopted by SMEs and startups since they provide efficient resource utilization. Azure's HDInsight[5] and Amazon's EMR[6] are such solutions, which provide easy distribution of these technologies as managed clusters with enterprise-level security and monitoring and are the leading players in this domain. Given the current trend of the usage of cloud-based services, it can be stated that is going to see the rise of the information service organizations, the same way the banking industry arose centuries or millennia ago to manage and handle our financial assets.

Since the overall focus of employing a big-data strategy is on gaining business insights, companies are looking forward to developing a comprehensive information management strategy that involves more than simply ingesting Big Data. Specifically, they want to integrate their existing data systems, including the

[3] https://zookeeper.apache.org/
[4] http://storm.apache.org/
[5] https://azure.microsoft.com/it-it/services/hdinsight/
[6] https://aws.amazon.com/it/emr/

relational DBMS, enterprise content management systems, data warehouses, etc. This is where the concept of Data exploration comes into the picture, that describes the data by means of statistical and visualization techniques which help to explore it, to bring its important aspects into focus for further analysis. To achieve comprehensive Data exploration, companies need to do away with traditional analytic techniques and move from hindsight to foresight analytics. If this variable data is the oil, data analysis must be the engine that drives its exploration, and therefore the tools used for this task should be able to harness data from all the given data systems.

In all, can be said that the Big Data technologies have the propensity within them to foster great results for the organizations if combined with efficiently sought-after result-oriented analytics. But for that to happen, organizations must evolve their existing data ingestion architectures. With more data and more potential relationships between data points, businesses will need experts to sift through and pinpoint the signal from the noise, and this is where the role of data scientist comes into the picture. IT departments also need to continue building up a data-driven mindset which includes investing in the back end of data by improving governance policies and data quality.

2.3 Apache Spark, Databricks distribution

Databricks offers a Unified Analytics Platform that unites three realms of experiences together: people, processes and infrastructure (platform). Surrounding, and built atop Apache Spark, are software components that enhance Spark's performance, security, fast IO access, and collaborative workspace environment so data analysts, data engineers and data scientists can work together.

2.3.1 Getting Started with Apache Spark™ on Databricks

In the following paragraphs, it will be possible to familiarize a user with the Spark UI, learn how to create Spark jobs, load data and work with Datasets, get familiar with Spark's DataFrames API and run machine learning algorithms. Instead of worrying about spinning up clusters, maintaining clusters, maintaining code history, or Spark versions, it is possible to start writing Spark queries instantly and

focus on data problems. It will show 4 modules to getting started: An overview on how to use open source Apache Spark and then leverage this knowledge to learn how to use Spark DataFrames with Spark SQL. In time for Spark 2.0, it will be also discussed how to use Datasets and how DataFrames and Datasets are now unified. Each of these modules refers to standalone usage scenarios, including IoT and home sales, with notebooks and datasets.

Apache Spark is a powerful open-source processing engine built around speed, ease of use, and sophisticated analytics, its infrastructure is showed in Fig 2.4.

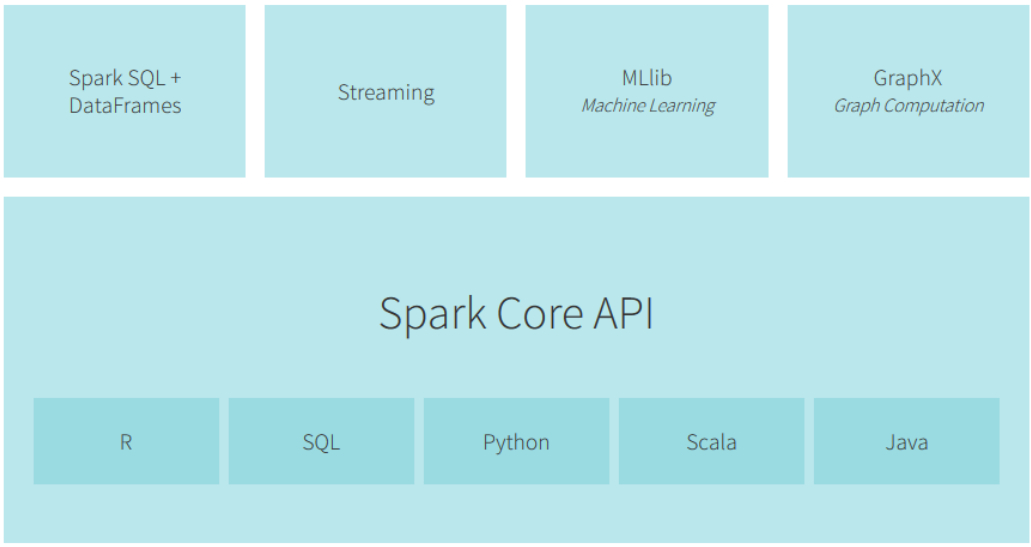


Figure 2-4 Architecture of Spark

The Spark Core is the underlying general execution engine for the Spark platform that all other functionality is built on top of. It provides in-memory computing capabilities to deliver speed, a generalized execution model to support a wide variety of applications, and Java, Scala, and Python APIs for ease of development.

Many data scientists, analysts, and general BI users rely on interactive SQL queries for exploring data. Spark SQL is a Spark module for structured data processing. It provides a programming abstraction called DataFrames and can also act as distributed SQL query engine. It enables unmodified Hadoop Hive queries to run up to 100x faster on existing deployments and data. It also provides powerful integration with the rest of the Spark ecosystem (e.g., integrating SQL query processing with machine learning).

Many applications need the ability to process and analyze not only batch data but also streams of new data in real-time. Running on top of Spark, Spark Streaming enables powerful interactive and analytical applications across both streaming and historical data, while inheriting Spark's ease of use and fault tolerance characteristics. It readily integrates with a wide variety of popular data sources, including HDFS[7], Flume[8], Kafka[9], and Twitter[10]. Machine learning has quickly emerged as a critical piece in mining Big Data for actionable insights. Built on top of Spark, MLlib is a scalable machine learning library that delivers both high-quality algorithms (e.g., multiple iterations to increase accuracy) and blazing speed (up to 100x faster than MapReduce). The library is usable in Java, Scala, and Python as part of Spark applications, so can include it in complete workflows. GraphX is a graph computation engine built on top of Spark that enables users to interactively build, transform and reason about graph-structured data at scale. It comes complete with a library of common algorithms.

> "At Databricks, we're working hard to make Spark easier to use and run than ever, through our efforts on both the Spark codebase and support materials around it. All of our work on Spark is open source and goes directly to Apache."
>
> Matei Zaharia, VP, Apache Spark, Co-founder & Chief Technologist, Databricks

Databricks is a Unified Analytics Platform on top of Apache Spark that accelerates innovation by unifying data science, engineering and business. With our fully managed Spark clusters in the cloud, it can easily provide clusters with just a few clicks. Databricks incorporates an integrated workspace for exploration and visualization so users can learn, work, and collaborate in a single, easy to use environment. It can easily schedule any existing notebook or locally developed Spark code to go from prototype to production without re-engineering.

[7] https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html
[8] https://flume.apache.org/
[9] https://kafka.apache.org/
[10] https://twitter.com/

2.3.2 Overview

This module allows to quickly start using Apache Spark. As this is a quick start, will be discussed the various concepts briefly so a user can complete end-to-end examples.

To write a first Apache Spark Job using Databricks, the user will writes the code in the cells of a Databricks notebook. In this example, it will be used Python. For more information, there is also an available reference on the Apache Spark Quick Start Guide and the Databricks Guide. The purpose of this quick start is to showcase RDD's (Resilient Distributed Datasets) operations so that a user will be able to understand the Spark UI when debugging or trying to understand the tasks being undertaken.

When running this first command, it is reviewed a folder within the Databricks File System (an optimized version of S3) which contains the files.

```
# Look at the file system
%fs ls /databricks-datasets/samples/docs/
```

path
dbfs:/databricks-datasets/samples/docs/README.md

In the next command, it will be used the Spark Context to read the README.md text file.

```
# Setup the textFile RDD to read the README.md file
# Note this is lazy
textFile = sc.textFile("/databricks-datasets/samples/docs/README.md")
```

And then it will be possible to count the lines of this text file by running the command

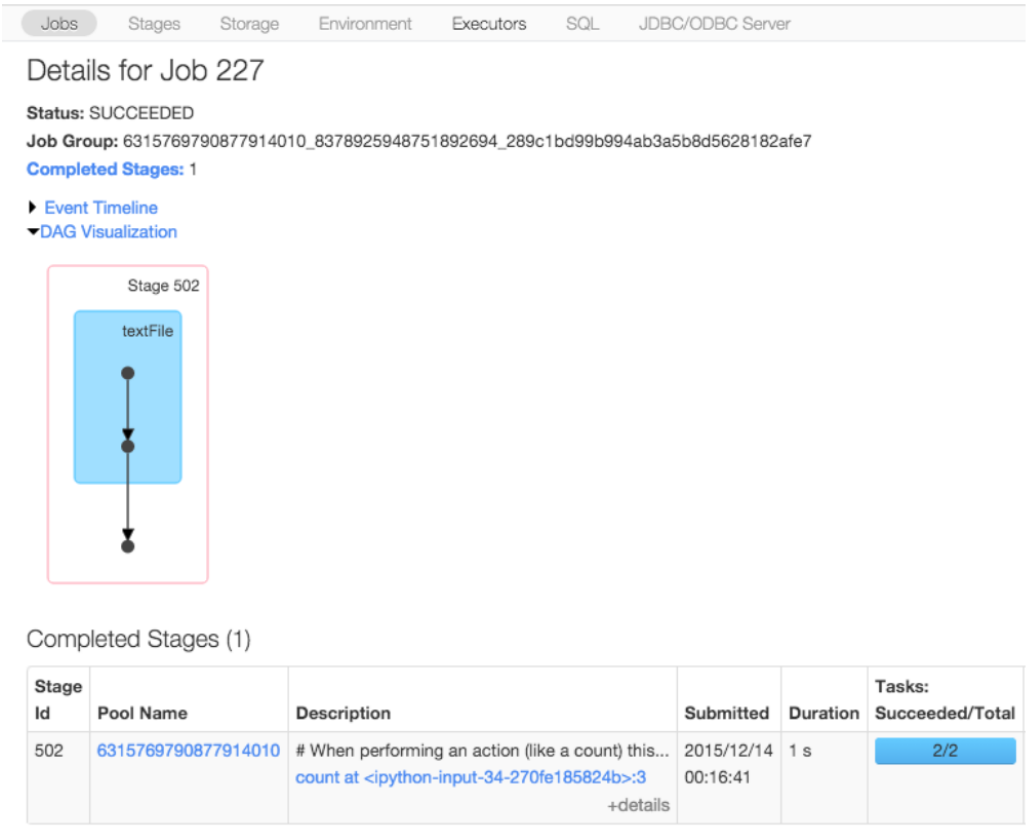
```
# Perform a count against the README.md file
textFile.count()
```

```
> # When performing an action (like a count) this is when the textFile is read and aggregate calculated
  #    Click on [View] to see the stages and executors
  textFile.count()

Out[34]: 82
```

One thing that can be noticed is that the first command, reading the textFile via the Spark Context (sc), did not generate any output while the second command (performing the count) did. The reason for this is because RDDs have *actions* (which returns values) as well as *transformations* (which returns pointers to new

RDDs). The first command was a transformation while the second one was an action. This is important because when Spark performs its calculations, it will not execute any of the transformations until an action occurs. This allows Spark to optimize (e.g. run a filter prior to a join) for performance instead of following the commands serially.

To see what is happening when running the `count()` command, it will be possible to see the jobs and stages within the Spark Web UI. It can access directly from the Databricks notebook, so the user does not need to change the context as is debugging the Spark job. As can be seen from the below **Jobs** view, when performing the *action* `count()` it also includes the previous *transformation* to access the text file.



What is happening under the covers becomes more apparent when reviewing the **Stages** view from the Spark UI (also directly accessible within the Databricks notebook). As it can be seen from the DAG visualization below, prior to the `PythonRDD [1333] count()` step, Spark will perform the task of accessing the file `([1330] textFile)` and running `MapPartitionsRDD [1331] textFile`.

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0.6 s	0.6 s	1 s	1 s	1 s
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Input Size / Records	1967.0 B / 38	1967.0 B / 38	1967.0 B / 44	1967.0 B / 44	1967.0 B / 44

As noted in the previous section, RDDs have actions which return values and transformations which return points to new RDDs. Transformations are lazy and executed when an action is run. Some examples include:

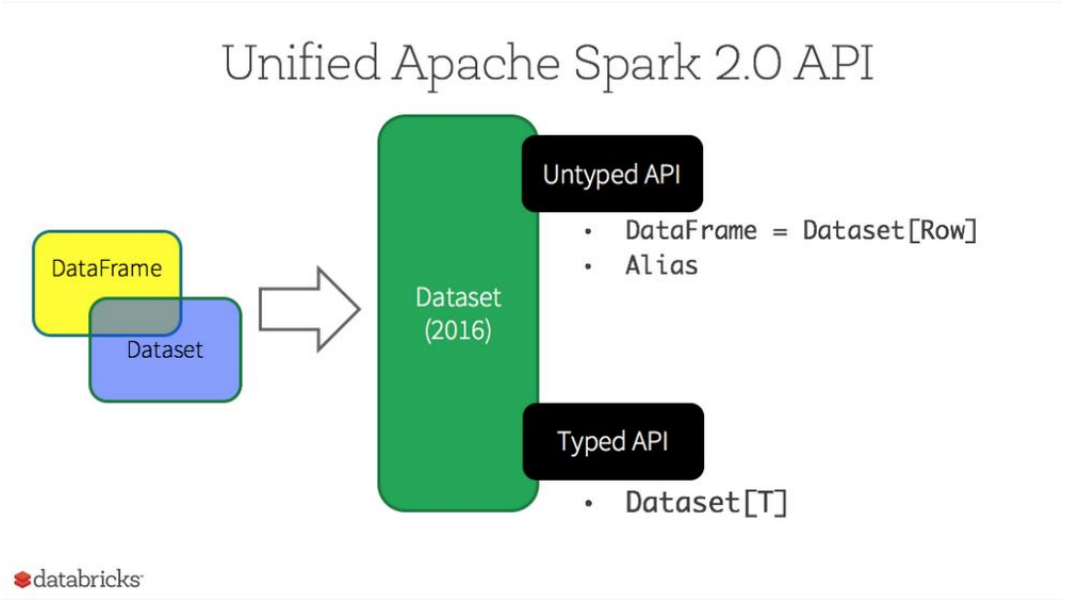
- **Transformations**: *`map()`, `flatMap()`, `filter()`, `mapPartitions()`, `mapPartitionsWithIndex()`, `sample()`, `union()`, `distinct()`, `groupByKey()`, `reduceByKey()`, `sortByKey()`, `join()`, `cogroup()`, `pipe()`, `coalesce()`, `repartition()`, `partitionBy()`, …*
- **Actions**: *`reduce()`, `collect()`, `count()`, `first()`, `take()`, `takeSample()`, `takeOrdered()`, `saveAsTextFile()`, `saveAsSequenceFile()`, `saveAsObjectFile()`, `countByKey()`, `foreach()`, …*

In many scenarios, especially with the performance optimizations embedded in DataFrames and Datasets, it will not be necessary to work with RDDs. But it is important to bring this up because:

- RDDs are the underlying infrastructure that allows Spark to run so fast (in-memory distribution) and provide data lineage.

- If is divided into more advanced components of Spark, it may be necessary to utilize RDDs.
- All the DAG visualizations within the Spark UI reference RDDs.

Saying this, when developing Spark applications, is typically used DataFrames and Datasets. As of Apache Spark 2.0, the DataFrame and Dataset APIs are merged together; a DataFrame is the Dataset Untyped API while what was known as a Dataset is the Dataset Typed API (Fig 2.4)



2.3.3 Datasets

The Apache Spark Dataset API provides a type-safe, object-oriented programming interface. In other words, in Spark 2.0 DataFrame and Datasets are unified as explained in Quick Start about RDDs, DataFrames, and Datasets, and DataFrame is an alias for an untyped Dataset [Row]. Like DataFrames, Datasets take advantage of Spark's Catalyst optimizer[11] by exposing expressions and data fields to a query planner. Beyond Catalyst's optimizer, Datasets also leverage Tungsten's fast in-memory encoding. They extend these benefits with compile-time type safety, meaning production applications can be checked for errors before they are running, and they also allow direct operations over user-defined

[11] At the core of Spark SQL is the Catalyst optimizer, which leverages advanced programming language features (e.g. Scala's pattern matching and quasi-quotes) in a novel way to build an extensible query optimizer.

classes, as it showed in a couple of simple examples below. Lastly, the Dataset API offers a high-level domain specific language operation like *sum(), avg(), join(), select(), groupBy()*, making the code a lot easier to express, read, and write.

In this module, it will be learnt two ways to create Datasets: dynamically creating a data and reading from JSON file using Spark Session. Additionally, through simple and short examples, it will be learnt about Dataset API operations on the Dataset, issue SQL queries and visualize data. For learning purposes, were used a small IoT Device dataset; however, there is no reason why it cannot be used a large dataset[12].

There are two easy ways to have structured data accessible and process it using Dataset APIs within a notebook. First, for primitive types in examples or demos, can be created them within a Scala or Python notebook or in a sample Spark application. For example, here's a way to create a Dataset of 100 integers in a notebook. Note that in Spark 2.0, the `SparkContext` is subsumed by `SparkSession`, a single point of entry, called `Spark`. Going forward, a user can use this handle in the driver or notebook cell, as showed below, in which were created 100 integers as Dataset[Long].

```
// range of 100 numbers to create a Dataset.
val range100 = spark.range(100)
range100.collect()
```

```
▸ (1) Spark Jobs
range100: org.apache.spark.sql.Dataset[Long] = [id: bigint]
res0: Array[Long] = Array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 3
6, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 7
6, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99)
Command took 4.01s
```

Second, the more common way is to read a data file from an external data sources, such as HDFS, S3, NoSQL, RDBMS, or local filesystem. Spark supports multiple formats: JSON, CSV, Text, Parquet, ORC etc. To read a JSON file can be simply used the `SparkSession` handle *spark*.

```
// read a JSON file from a location mounted on a DBFS mount point
// Note that there is used the new entry point in Spark 2.0 called spark
val jsonData = spark.read.json("/databricks-
datasets/data/people/person.json")
```

[12] There are several datasets available in the /databricks-datasets folder which is accessible within the Databricks platform.

At the time of reading the JSON file, Spark does not know the structure of the data on-hand, how a user wants to organize data into a type-specific JVM object. It attempts to infer the schema from the JSON file and creates a `DataFrame = Dataset[Row]` of generic Row objects. Alternatively, to convert the DataFrame into a Dataset reflecting a Scala class object, the user can define a domain specific Scala case class, followed by explicitly converting into that type, as showed below.

```
// First, define a case class that represents our type-specific Scala JVM
Object
case class Person (email: String, iq: Long, name: String)

// Read the JSON file, convert the DataFrames into a type-specific JVM
Scala object Person. Note that at this stage Spark, upon reading JSON,
created a generic
// DataFrame = Dataset[Rows]. By explicitly converting DataFrame into
Dataset
// results in a type-specific rows or collection of objects of type
Person
val ds = spark.read.json("/databricks-
datasets/data/people/person.json").as[Person]
```

In a second example, is done something similar with IoT devices state information captured in a JSON file: define a case class and read the JSON file from the FileStore and convert the `DataFrame = Dataset[DeviceIoTData].`

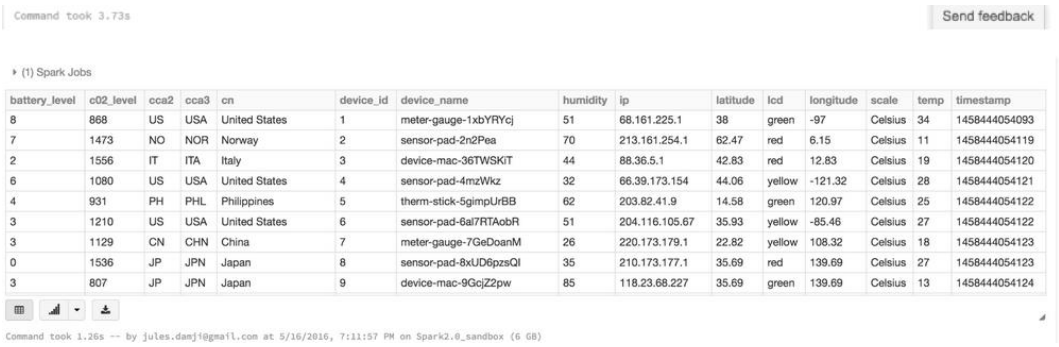
There are a couple of reasons why a user wants to convert a DataFrame into a type-specific JVM objects. First, after an explicit conversion, for all relational and query expressions using Dataset API, it will be possible to get the compile-type safety. For example, if the user applies a filter operation using the wrong data type, Spark will detect mismatch types and issue a compile error rather an execution runtime error, resulting in catching errors earlier. Second, the Dataset API provides high-order methods making code much easier to read and develop.

In the following submodule, Processing and Visualizing a Dataset, it will be noticed how the use of Dataset typed objects make the code much easier to express and read. As above with Person example, below was created a case class that encapsulates our Scala object.

```
// define a case class that represents our Device data.
case class DeviceIoTData (
  battery_level: Long,
  c02_level: Long,
  cca2: String,
  cca3: String,
  cn: String,
  device_id: Long,
  device_name: String,
  humidity: Long,
  ip: String,
  latitude: Double,
  longitude: Double,
  scale: String,
  temp: Long,
  timestamp: Long
)

// fetch the JSON device information uploaded into the Filestore
val jsonFile = "/databricks-datasets/data/iot/iot_devices.json"

// read the json file and create the dataset from the case class
DeviceIoTData
// ds is now a collection of JVM Scala objects DeviceIoTData
val ds = spark.read.json(jsonFile).as[DeviceIoTData]
```

To view this data in a tabular format instead of exporting this data out to a third party tool, can be used the Databricks `display()` command. That is, once is loaded the JSON data and converted into a Dataset for a type-specific collection of JVM objects, the user can view them as it would view a DataFrame, by using either `display()` or using standard Spark commands, such as `take()`, `foreach()`, and `println()` API calls.

```
// display the dataset table just read in from the JSON file
display(ds)
```



Command took 3.73s — Send feedback

▸ (1) Spark Jobs

battery_level	c02_level	cca2	cca3	cn	device_id	device_name	humidity	ip	latitude	lcd	longitude	scale	temp	timestamp
8	868	US	USA	United States	1	meter-gauge-1xbYRYcj	51	68.161.225.1	38	green	-97	Celsius	34	1458444054093
7	1473	NO	NOR	Norway	2	sensor-pad-2n2Pea	70	213.161.254.1	62.47	red	6.15	Celsius	11	1458444054119
2	1556	IT	ITA	Italy	3	device-mac-36TWSKiT	44	88.36.5.1	42.83	red	12.83	Celsius	19	1458444054120
6	1080	US	USA	United States	4	sensor-pad-4mzWkz	32	66.39.173.154	44.06	yellow	-121.32	Celsius	28	1458444054121
4	931	PH	PHL	Philippines	5	therm-stick-5gimpUrBB	62	203.82.41.9	14.58	green	120.97	Celsius	25	1458444054122
3	1210	US	USA	United States	6	sensor-pad-6al7RTAobR	51	204.116.105.67	35.93	yellow	-85.46	Celsius	27	1458444054122
3	1129	CN	CHN	China	7	meter-gauge-7GeDoanM	26	220.173.179.1	22.82	yellow	108.32	Celsius	18	1458444054123
0	1536	JP	JPN	Japan	8	sensor-pad-8xUD6pzsQI	35	210.173.177.1	35.69	red	139.69	Celsius	27	1458444054123
3	807	JP	JPN	Japan	9	device-mac-9GcjZ2pw	85	118.23.68.227	35.69	green	139.69	Celsius	13	1458444054124

Command took 1.26s -- by jules.damji@gmail.com at 5/16/2016, 7:11:57 PM on Spark2.0_sandbox (6 GB)

```
// Using the standard Spark commands, take() and foreach(), print the
first
// 10 rows of the Datasets.
ds.take(10).foreach(println(_))
```

```
▸ (1) Spark Jobs
DeviceIoTData(8,868,US,USA,United States,1,meter-gauge-1xbYRYcj,51,68.161.225.1,38.0,green,-97.0,Celsius,34,1458444054093)
DeviceIoTData(7,1473,NO,NOR,Norway,2,sensor-pad-2n2Pea,70,213.161.254.1,62.47,red,6.15,Celsius,11,1458444054119)
DeviceIoTData(2,1556,IT,ITA,Italy,3,device-mac-36TWSKiT,44,88.36.5.1,42.83,red,12.83,Celsius,19,1458444054120)
DeviceIoTData(6,1080,US,USA,United States,4,sensor-pad-4mzWkz,32,66.39.173.154,44.06,yellow,-121.32,Celsius,28,1458444054121)
DeviceIoTData(4,931,PH,PHL,Philippines,5,therm-stick-5gimpUrBB,62,203.82.41.9,14.58,green,120.97,Celsius,25,1458444054122)
DeviceIoTData(3,1210,US,USA,United States,6,sensor-pad-6al7RTAobR,51,204.116.105.67,35.93,yellow,-85.46,Celsius,27,1458444054122)
DeviceIoTData(3,1129,CN,CHN,China,7,meter-gauge-7GeDoanM,26,220.173.179.1,22.82,yellow,108.32,Celsius,18,1458444054123)
DeviceIoTData(0,1536,JP,JPN,Japan,8,sensor-pad-8xUD6pzsQI,35,210.173.177.1,35.69,red,139.69,Celsius,27,1458444054123)
DeviceIoTData(3,807,JP,JPN,Japan,9,device-mac-9GcjZ2pw,85,118.23.68.227,35.69,green,139.69,Celsius,13,1458444054124)
DeviceIoTData(7,1470,US,USA,United States,10,sensor-pad-10BsywSYUF,56,208.109.163.218,33.61,red,-111.89,Celsius,26,1458444054125)
Command took 1.18s -- by jules.damji@gmail.com at 5/16/2016, 7:12:05 PM on Spark2.0_sandbox (6 GB)
```

An additional benefit of using the Databricks `display()` command is that it can quickly view this data with several embedded visualizations (Fig. 2.5-6). For example, in a new cell, the user can issue SQL queries and click on the map to see the data. But first, is mandatory to save the dataset, ds, as a temporary table.

```
// registering the Dataset as a temporary table to which the user can
issue SQL queries
ds.createOrReplaceTempView("iot_device_data")
```

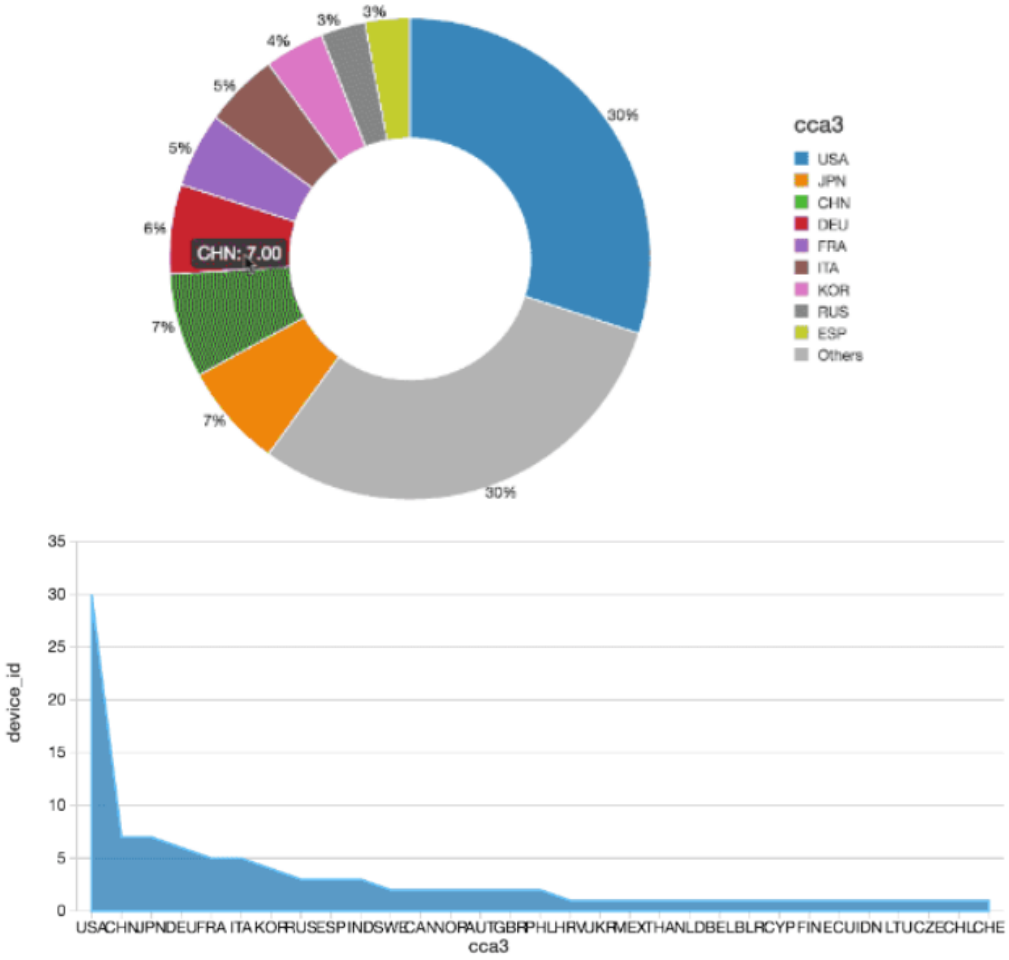


Figure 2-5 Some embedded visualization

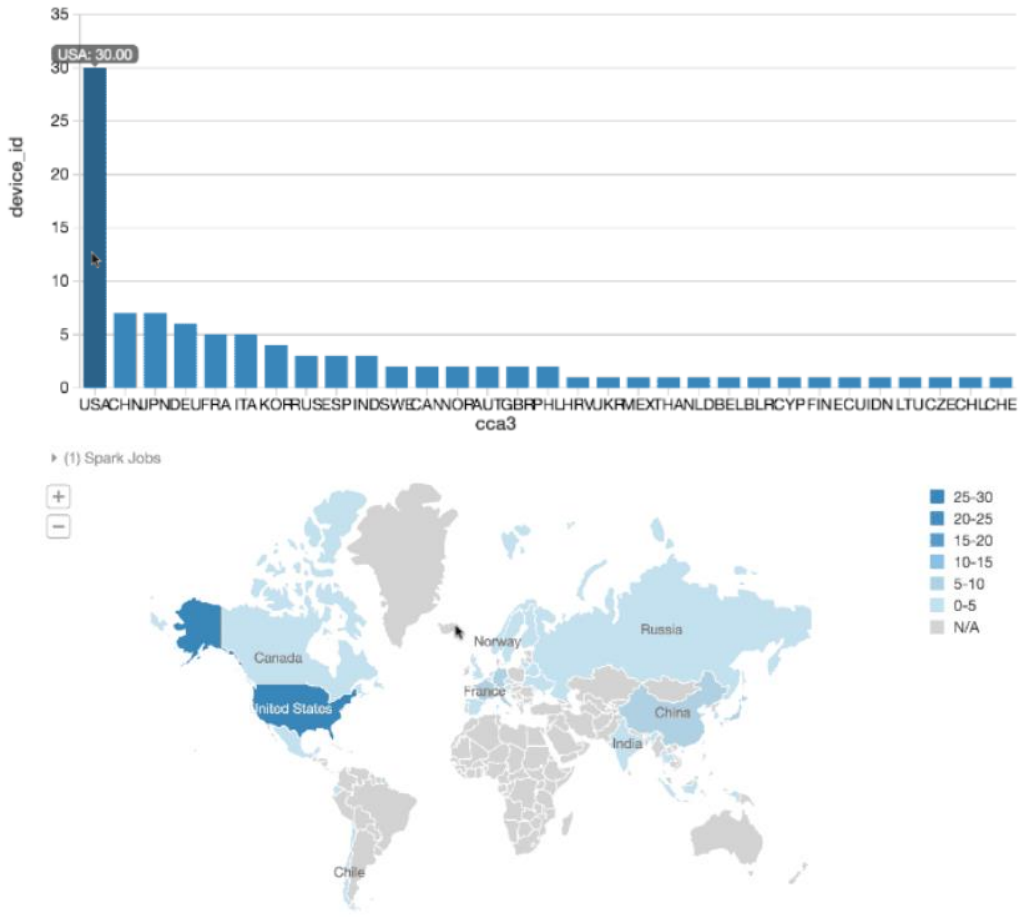


Figure 2-6 Some embedded visualization

Like RDD, Dataset has transformations and actions methods. Most importantly are the high-level domain specific operations such as `sum(), select(), avg(), join(), and union()` that are absent in RDDs. For more information, look at the Scala Dataset API. Let is look at a few handy ones in action.

In the example below, is used `filter(), map(), groupBy()`, and `avg()`, all higher-level methods, to create another Dataset, with only fields that were wishing to view. What is noteworthy is that is accessed the attributes the user wanted to filter by their names as defined in the case class. That is, was used the dot notation to access individual fields. As such, it makes code easy to read and write.

```
// filter out all devices whose temperature exceed 25 degrees and
generate another Dataset with three fields that of interest and then
display the mapped Dataset
val dsTemp = ds.filter(d => d.temp > 25).map(d => (d.temp, d.device_name,
d.cca3)
display(dsTemp)
```

▸ (1) Spark Jobs

_1	_2	_3	_4
34	meter-gauge-1xbYRYcj	1	USA
28	sensor-pad-4mzWkz	4	USA
27	sensor-pad-6al7RTAobR	6	USA
27	sensor-pad-8xUD6pzsQI	8	JPN
26	sensor-pad-10BsywSYUF	10	USA
31	meter-gauge-17zb8Fghhl	17	USA
31	sensor-pad-18XULN9Xv	18	CHN
29	meter-gauge-19eg1BpfCO	19	USA
30	device-mac-21sjz5h	21	AUT

```
// Apply higher-level Dataset API methods such as groupBy() and avg().
// Filter temperatures > 25, along with their corresponding
// devices' humidity, compute averages, groupBy cca3 country codes,
// and display the results, using table and bar charts
val dsAvgTmp = ds.filter(d => {d.temp > 25}).map(d => (d.temp, d.humidity, d.cca3)).groupBy($"_3").avg()

// display averages as a table, grouped by the country
display(dsAvgTmp)
```

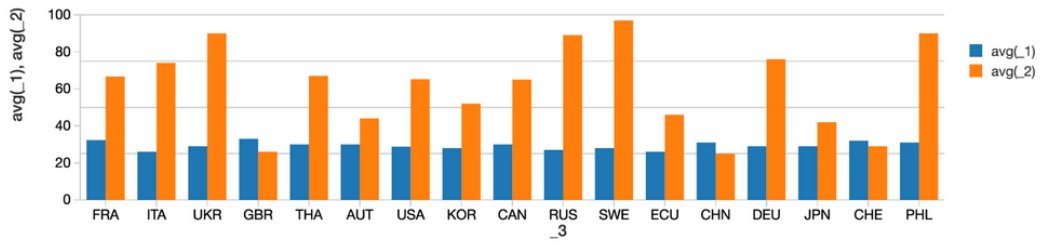
▸ (2) Spark Jobs

_3	avg(_1)	avg(_2)
FRA	32.333333333333336	66.66666666666667
ITA	26	74
UKR	29	90
GBR	33	26
THA	30	67
AUT	30	44
USA	28.76923076923077	65.23076923076923
KOR	28	52
CAN	30	65

Command took 1.05s

```
// display the averages as bar graphs, grouped by the country
display(dsAvgTmp)
```

▸ (2) Spark Jobs



Plot Options...

Command took 1.05s

```
// Select individual fields using the Dataset method select()
// where battery_level is greater than 6. Note this high-level
// domain specific language API reads like a SQL query
display(ds.select($"battery_level", $"c02_level",
$"device_name").where($"battery_level" > 6).sort($"c02_level"))
```

▸ (1) Spark Jobs

battery_level	c02_level	device_name
8	857	sensor-pad-46MiQ33UDaaa
8	868	meter-gauge-1xbYRYcj
8	934	meter-gauge-712JgErD0zVw
7	940	sensor-pad-34F1Jubre3B
9	986	sensor-pad-48jt4eL
7	997	therm-stick-55kEAHLqWn0
7	1131	therm-stick-35Lg804z
7	1155	sensor-pad-20gFNfBgqr
7	1160	meter-gauge-61NehO8Msi

Command took 0.27s

2.3.4 Dataframes

Apache Spark DataFrames were created to run Spark programs faster from both a developer and an execution perspective. With less code to write and less data to read, the Catalyst optimizer solves common problems efficiently and faster using DataFrame functions (e.g. select columns, filtering, joining different data sources, aggregation, etc.). DataFrames also allow to seamlessly intermix operations with custom SQL, Python, Java, R, or Scala code.

The easiest way to work with DataFrames is to access an example dataset[13]. For example, to access the file that compares city population vs. median sale prices of homes, the user can access the file /databricks-datasets/samples/population-vs-price/data_geo.csv.

Was used the spark-csv package from Spark Packages (a community index of packages for Apache Spark) to quickly import the data, specify that a header exists, and infer the schema.

```
# Use the Spark CSV datasource with options specifying:
# - First line of file is a header
# - Automatically infer the schema of the data
data = sqlContext.read.format("csv")
  .option("header", "true")
  .option("inferSchema", "true")
  .load("/databricks-datasets/samples/population-vs-price/data_geo.csv")

data.cache() # Cache data for faster reuse
data = data.dropna() # drop rows with missing values

# Register table so it is accessible via SQL Context
# For Apache Spark = 2.0
data.createOrReplaceTempView("data_geo")
```

[13] There are made several datasets available in the /databricks-datasets folder which is accessible within the Databricks platform

Now that was created the `data` DataFrame, the user can quickly access the data using standard Spark commands such as `take()`. For example, it can use the command `data.take(10)` to view the first ten rows of the `data` DataFrame.

```
> data.take(10)

▸ (1) Spark Jobs
Out[3]:
[Row(2014 rank=101, City=u'Birmingham', State=u'Alabama', State Code=u'AL', 2014 Population estimate=212247, 2015 median sales price=162.9),
 Row(2014 rank=125, City=u'Huntsville', State=u'Alabama', State Code=u'AL', 2014 Population estimate=188226, 2015 median sales price=157.7),
 Row(2014 rank=122, City=u'Mobile', State=u'Alabama', State Code=u'AL', 2014 Population estimate=194675, 2015 median sales price=122.5),
 Row(2014 rank=114, City=u'Montgomery', State=u'Alabama', State Code=u'AL', 2014 Population estimate=200481, 2015 median sales price=129.0),
 Row(2014 rank=64, City=u'Anchorage[19]', State=u'Alaska', State Code=u'AK', 2014 Population estimate=301010, 2015 median sales price=None),
 Row(2014 rank=78, City=u'Chandler', State=u'Arizona', State Code=u'AZ', 2014 Population estimate=254276, 2015 median sales price=None),
 Row(2014 rank=86, City=u'Gilbert[20]', State=u'Arizona', State Code=u'AZ', 2014 Population estimate=239277, 2015 median sales price=None),
 Row(2014 rank=88, City=u'Glendale', State=u'Arizona', State Code=u'AZ', 2014 Population estimate=237517, 2015 median sales price=None),
 Row(2014 rank=38, City=u'Mesa', State=u'Arizona', State Code=u'AZ', 2014 Population estimate=464704, 2015 median sales price=None),
 Row(2014 rank=148, City=u'Peoria', State=u'Arizona', State Code=u'AZ', 2014 Population estimate=166934, 2015 median sales price=None)]
Command took 0.12s
```

To view this data in a tabular format, was used the `display()` command within Databricks.

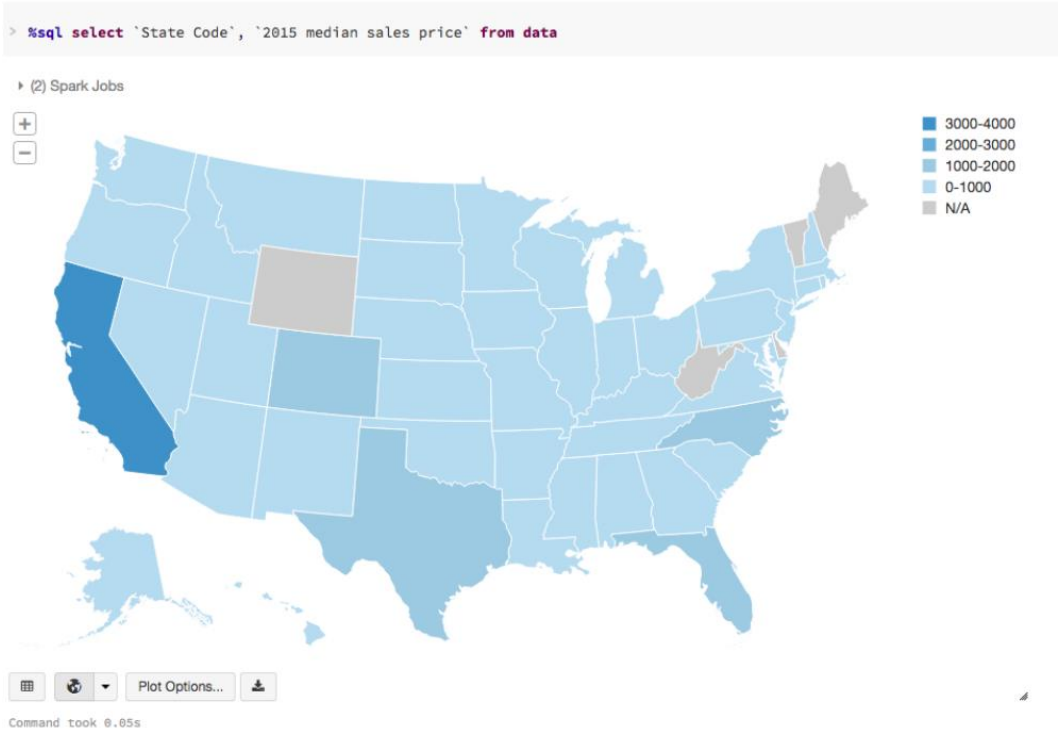
```
> display(data)

▸ (2) Spark Jobs
```

2014 rank	City	State	State Code	2014 Population estimate	2015 median sales price
101	Birmingham	Alabama	AL	212247	162.9
125	Huntsville	Alabama	AL	188226	157.7
122	Mobile	Alabama	AL	194675	122.5
114	Montgomery	Alabama	AL	200481	129
64	Anchorage[19]	Alaska	AK	301010	null
78	Chandler	Arizona	AZ	254276	null
86	Gilbert[20]	Arizona	AZ	239277	null
88	Glendale	Arizona	AZ	237517	null

```
Command took 1.83s
```

An additional benefit of using the Databricks `display()` command is that it can quickly view this data with a number of embedded visualizations. For example, in a new cell, can be specified the following SQL query and click on the map.

```
%sql select `State Code`, `2015 median sales price` from data
```



2.3.5 Machine Learning

As organizations create more diverse and more user-focused data products and services, there is a growing need for machine learning, which can be used to develop personalization, recommendations, and predictive insights. Apache Spark's Machine Learning Library (MLlib) allows data scientists to focus on their data problems and models instead of solving the complexities surrounding distributed data (such as infrastructure, configurations, and so on).

The easiest way to work with DataFrames is to access an example dataset. For example, to access the file that compares city population vs. median sale prices of homes, the user can access the file /databricks-datasets/samples/population-vs-price/data_geo.csv.

Was used the spark-csv package from Spark Packages (a community index of packages for Apache Spark) to quickly import the data, specify that a header exists, and infer the schema.

```
# Use the Spark CSV datasource with options specifying:
# - First line of file is a header
# - Automatically infer the schema of the data
data = sqlContext.read.format("csv")
  .option("header", "true")
  .option("inferSchema", "true")
  .load("/databricks-datasets/samples/population-vs-price/data_geo.csv")

data.cache() # Cache data for faster reuse
data = data.dropna() # drop rows with missing values

# Register table so it is accessible via SQL Context
# For Apache Spark = 2.0
data.createOrReplaceTempView("data_geo")
```

To view this data in a tabular format, was used the `display()` command within Databricks.

```
> display(data)
```

▸ (2) Spark Jobs

2014 rank	City	State	State Code	2014 Population estimate	2015 median sales price
101	Birmingham	Alabama	AL	212247	162.9
125	Huntsville	Alabama	AL	188226	157.7
122	Mobile	Alabama	AL	194675	122.5
114	Montgomery	Alabama	AL	200481	129
64	Anchorage[19]	Alaska	AK	301010	null
78	Chandler	Arizona	AZ	254276	null
86	Gilbert[20]	Arizona	AZ	239277	null
88	Glendale	Arizona	AZ	237517	null

Command took 1.83s

In supervised learning, such as a regression algorithm, the user typically will define a label and a set of features. In our linear regression example, the label is the `2015 median sales price` while the feature is the `2014 Population Estimate`. That is, is was tried to use the *feature* (population) to predict the *label* (sales price). To simplify the creation of features within Python Spark MLlib, was used `LabeledPoint` to convert the feature (population) to a Vector type.

```
# convenience for specifying schema
from pyspark.mllib.regression import LabeledPoint

data = data.select("2014 Population estimate", "2015 median sales price")
  .map(lambda r: LabeledPoint(r[1], [r[0]]))
  .toDF()
display(data)
```

features	label
{"type":1,"size":1,"indices":[],"values":[212247]}	162.9
{"type":1,"size":1,"indices":[],"values":[188226]}	157.7
{"type":1,"size":1,"indices":[],"values":[194675]}	122.5
{"type":1,"size":1,"indices":[],"values":[200481]}	129
{"type":1,"size":1,"indices":[],"values":[1537058]}	206.1
{"type":1,"size":1,"indices":[],"values":[527972]}	178.1

In this section, it will be executed two different linear regression models using different regularization parameters and determine its efficacy. That is, how well do either of these two models predict the sales price (label) based on the population (feature).

```
# Import LinearRegression class
from pyspark.ml.regression import LinearRegression

# Define LinearRegression algorithm
lr = LinearRegression()

# Fit 2 models, using different regularization parameters
modelA = lr.fit(data, {lr.regParam:0.0})
modelB = lr.fit(data, {lr.regParam:100.0})
```

The model can also make predictions by using the `transform()` function which adds a new column of predictions. For example, the code below takes the first model (`modelA`) and shows both the label (original sales price) and prediction (predicted sales price) based on the features (population).

```
# Make predictions
predictionsA = modelA.transform(data)
display(predictionsA)
```

features	label	prediction
{"type":1,"size":1,"indices":[],"values":[212247]}	162.9	199.31676595846622
{"type":1,"size":1,"indices":[],"values":[188226]}	157.7	198.40882267887176
{"type":1,"size":1,"indices":[],"values":[194675]}	122.5	198.65258131548575
{"type":1,"size":1,"indices":[],"values":[200481]}	129	198.8720359044423
{"type":1,"size":1,"indices":[],"values":[1537058]}	206.1	249.39183544694856
{"type":1,"size":1,"indices":[],"values":[527972]}	178.1	211.2505069330287
{"type":1,"size":1,"indices":[],"values":[197706]}	131.8	198.76714674075743
{"type":1,"size":1,"indices":[],"values":[346997]}	685.7	204.41003255541705

To evaluate the regression analysis, it will be calculated the *root mean square error* using the `RegressionEvaluator`. Below is the pySpark code for evaluating the two models and their output.

```
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(metricName="rmse")
RMSE = evaluator.evaluate(predictionsA)
print("ModelA: Root Mean Squared Error = " + str(RMSE))

# ModelA: Root Mean Squared Error = 128.602026843
predictionsB = modelB.transform(data)
RMSE = evaluator.evaluate(predictionsB)
print("ModelB: Root Mean Squared Error = " + str(RMSE))

# ModelB: Root Mean Squared Error = 129.496300193
```

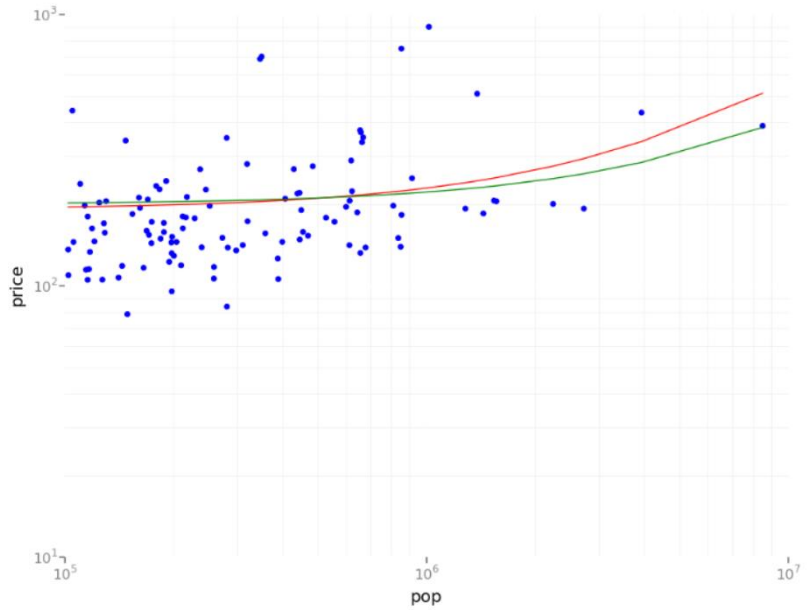
As is typical for many machine learning algorithms, the user will want to visualize the scatterplot. As Databricks supports Python `pandas` and `ggplot`, the code

below creates a linear regression plot using Python Pandas DataFrame (`pydf`) and `ggplot` to display the scatterplot and the two regression models.

```
# Import numpy, pandas, and ggplot
import numpy as np
from pandas import *
from ggplot import *

# Create Python DataFrame
pop = data.map(lambda p: (p.features[0])).collect()
price = data.map(lambda p: (p.label)).collect()
predA = predictionsA.select("prediction").map(lambda r: r[0]).collect()
predB = predictionsB.select("prediction").map(lambda r: r[0]).collect()

pydf = DataFrame({'pop':pop,'price':price,'predA':predA, 'predB':predB})
```

Visualizing the Model :

```
# Create scatter plot and two regression models (scaling exponential)
using ggplot
p = ggplot(pydf, aes('pop','price')) +
geom_point(color='blue') +
geom_line(pydf, aes('pop','predA'), color='red') +
geom_line(pydf, aes('pop','predB'), color='green') +
scale_x_log10() + scale_y_log10()
display(p)
```



3. Data Structure

This section describes the structure of the data used to conduct the analysis on non-technical losses over smart energy grids.

3.1 Datasets provided

The analysis is based on two kinds of datasets:

- A dataset that includes all the information in respect to the consumers, such as their location, the type of customer, etc. These data do not change frequently. This dataset was called: “Master dataset”.
- The second dataset includes information on the energy measured by the metering devices deployed at the customers throughout the year, where measures are sampled over each month. This dataset was called: “Metering dataset”.

3.2 Master Dataset

The master data represent customer reference data, which typically changes infrequently and count more than 1.6 million customers of 2017, in the Table 3.1 the data dictionary.

Field Name	Type	Description
Id_customer	String	POD : Alphanumeric national code that uniquely identify the final customer
year_month	String	The year and month in reference to the single record
market	String	Type of energetic market
meter_type	String	Type of device that measures the amount of electric energy
status_customer	Binary	Indicator if the customer active or not in that month
contracted_power	Double	Indicate the power level in the contract
contracted_power_band	String	Indicate the power level in the contract divided by band
zone_number	Double	Indicate the number of zone where the customer resides

Table 3-1 Data Dictionary of Master Dataset

3.2.1 Id customer

The Point of Delivery (POD) is an alphanumeric national code that uniquely identify the point on the national territory where the electric energy is delivered to supplier and provided for the final customer. This is simple to recognize because start with two letter that identify the country, in the thesis, always IT acronym to Italy, after that can be found three numbers that identify the provider and the letter E, that stand for electric energy. The POD is completed by 8 digit that correspond to the client number directly found on the meter. An example of POD is "IT123E12345678".

This code remains the same also when someone change the supplier, in that case, the meter will be the same and there will be no change on the meter unless the customer demand higher contracted power. For sake of anonymity, the "id_customer" field has been provided masked as an increased number from 0 to about 1.600.000.

3.2.1 Market

Two are the main type of market in the energetic market: the deregulated and the protected ones. In the first one, the economic and contractual conditions for the supply of electricity are agreed between parties and not fixed by the energy Authorities. From the 1° July 2007 the customers can freely choose from which supplier buy the electricity and determine the conditions. In the second one, the economic and contractual conditions for the electric energy supply are fixed by the energy Authorities. The customer cannot decide at which supplier relying on but will follow the established prices that will update every three months based of prices trend of the oil and gas markets. Below in Table 3.2 the list of possible values for this feature.

Market
Deregulated Market
Protected Market
Internal Usage
Other Market

Table 3-2 Market Types

The Internal Usage refer to all the meters that track the acquisition and distribution among the city and Other Market refer to a particular type of customers that require a specific distinction of market.

The market feature presents a very imbalanced distribution given that the deregulated and protected market represent the 98 % of the customers (Fig 3.1). The number of customers with missing market values are 17.854 and are presented in the figures as unlabeled market type.

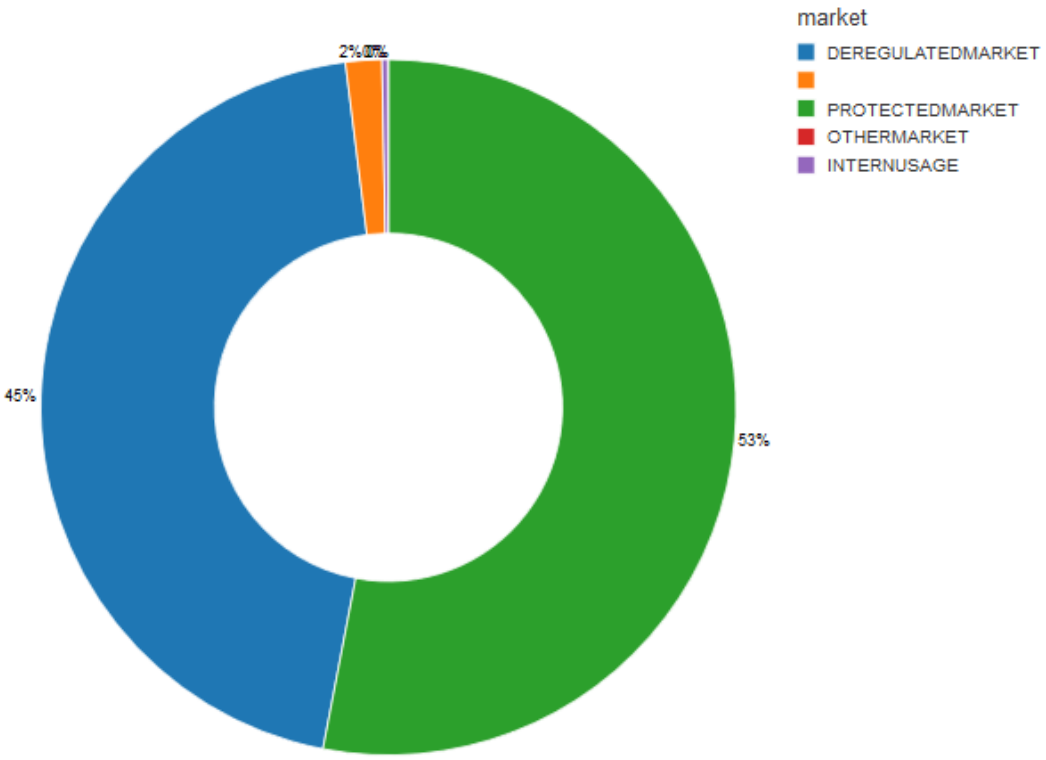


Figure 3-1 Market distribution

Compared with zone number as it can be seen in Fig 3.2 that, aside OTHERMARKET with Zone_1, there is a uniformly distribution of the type of market over the city.

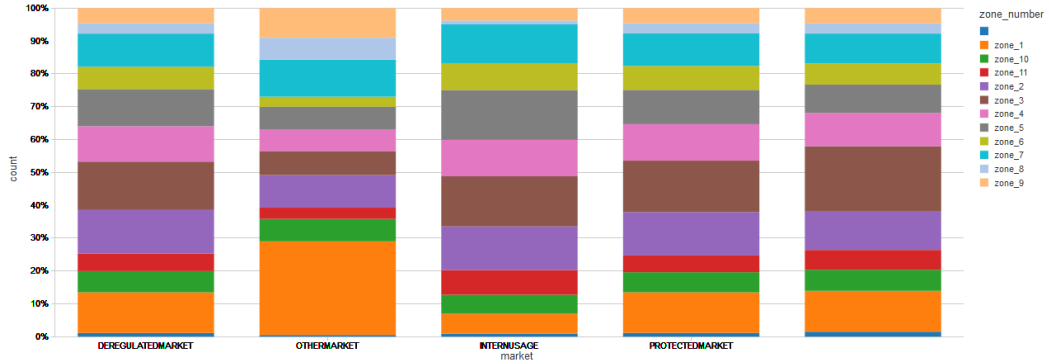


Figure 3-2 Market and Zone comparison

3.2.2 Contracted power values and bands

The contracted power indicates the power level written in the contract and is made available by supplier. The contracted power is defined according to customer necessities in the moment of contract stipulation based on type and number of electrical devices usually utilized. It is measured in kW and the bands according to the National Electric Service [48] are described in Table 3.3.

Band	Description
B1	Customers with contracted power until 1.5 kW
B2	Customers with contracted power greater than 1.5 kWh until 3 kW
B3	Customers with contracted power greater than 3 kW until 6 kW
B4	Customers with contracted power greater than 6 kW until 10 kW
B5	Customers with contracted power greater than 10 kW until 15 kW
B6	Customers with contracted power greater than 15 kW

Table 3-3 Contracted power values and bands

In Italy, the most part of the households is 3 kW and if the user wants a higher power level, until 10 kW, needs to request directly to the supplier. The power contracted level is reported on the bill within the section “Type of contract” or “delivery date**”.**

As it can be seen in Fig 3.3, the contracted power bands also present an unbalanced distribution because the B2 and B3 bands includes respectively the 75% and the 15%. As before the unlabeled band in the figures represent the missing values and the number amounts to 2.110.

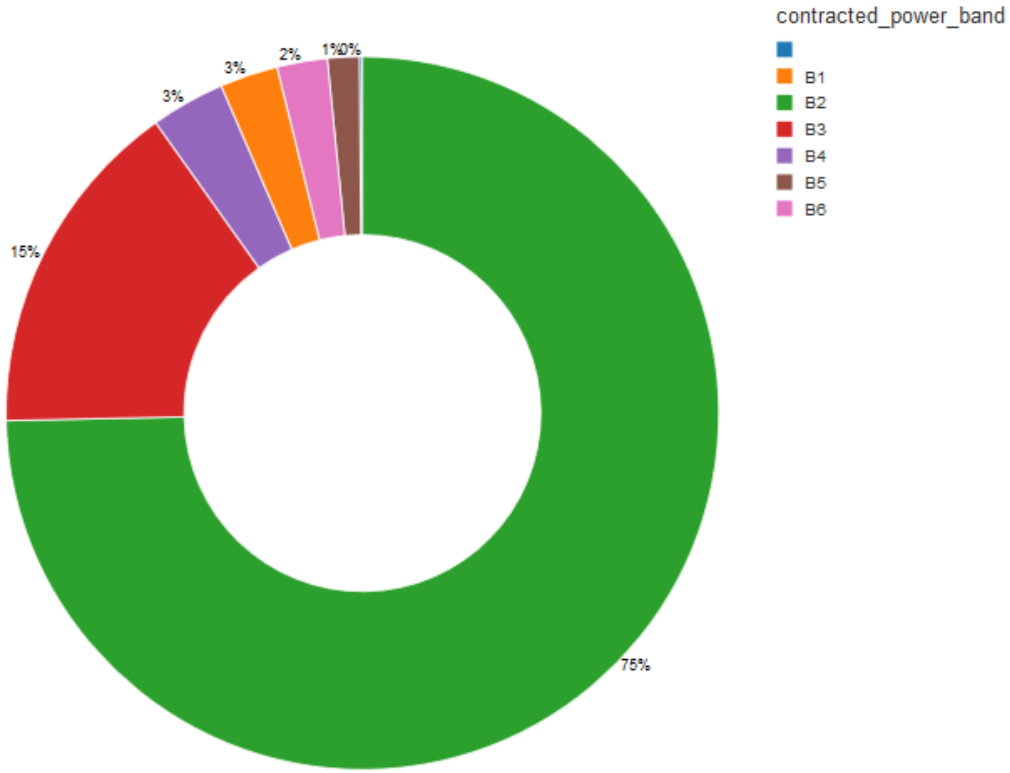


Figure 3-3 Contracted Power distribution

As can be expected the comparison with the zone number reveal a high presence of B2 and B3 for all the zone. In particular, the first one, B2, reach for all the area under observation at least the 55% of the total area, visible in Fig 3.4 below, the figure indicates also the numerosity of each area.

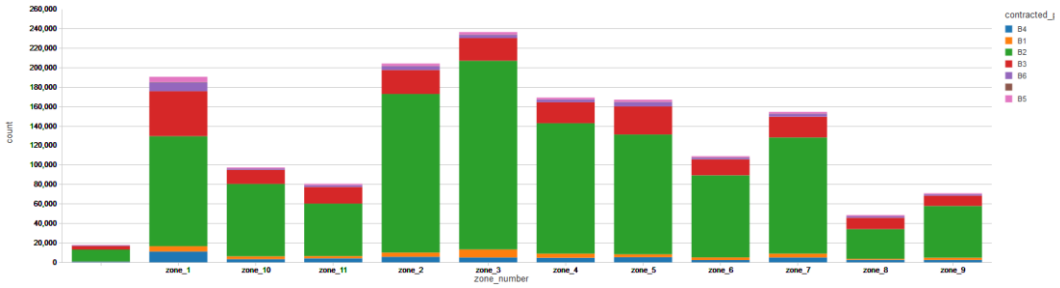


Figure 3-4 Contracted Power and Zone comparison

3.2.3 Number zone

The city under observation is divided into 11 sub-areas and the variable "zone_number" indicate the zone where customer belongs. The city can be imaged divided as in Fig 3.5.

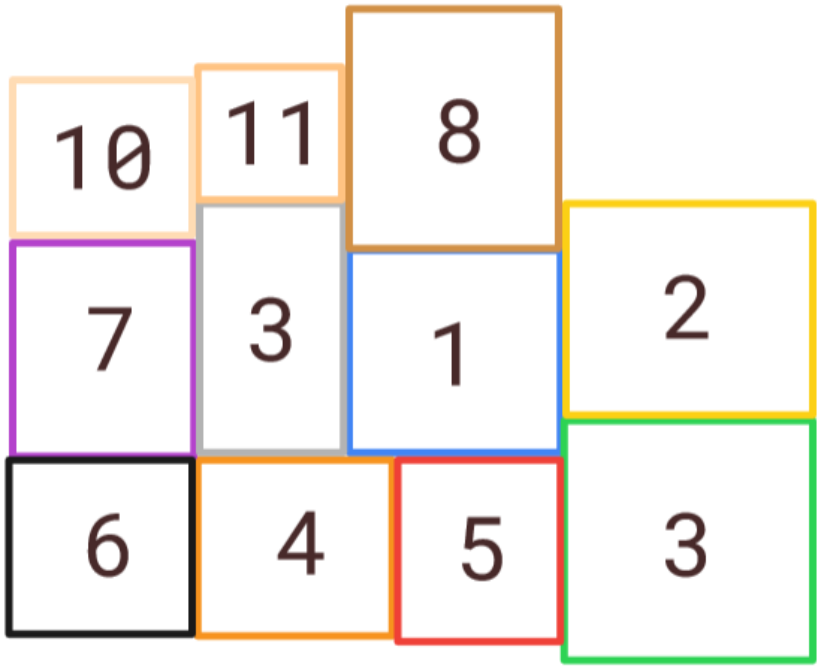


Figure 3-5 Zone map

The zones distribution observable in Fig 3.6, as can be expected from Fig 3.4, show the “zone_3” as the biggest zone with 236.731 costumers. The number of missing value is bigger compared to the other variables and count 26.085 missing values.

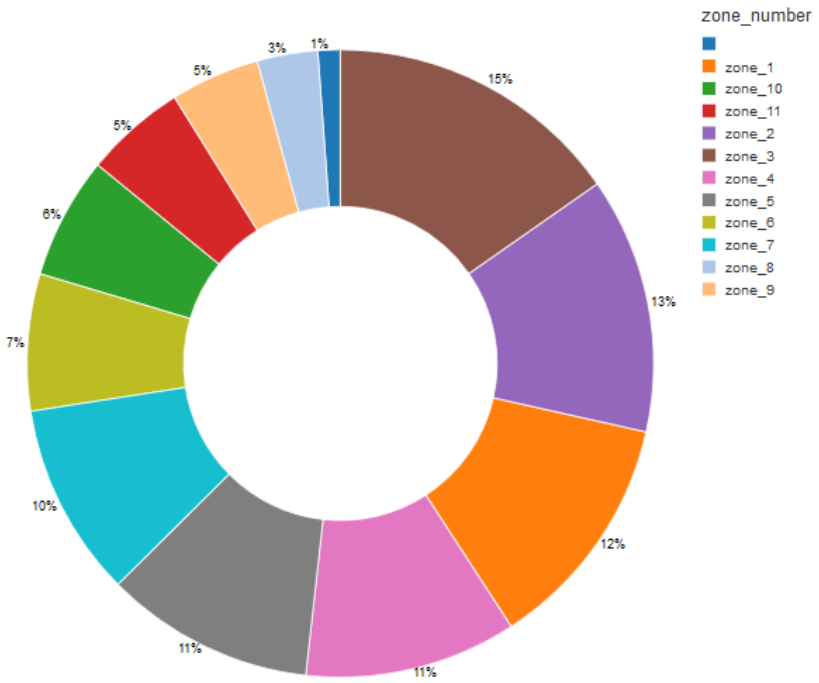


Figure 3-6 Zone distribution

3.2.4 Meter type

The smart meters are installed at almost all the final users. In 2007 the installation of electronic meters for medium voltage users was completed, while in 2011 the installation covered 95% of low voltage users. Electronic meters allow the measurement of active power and reactive power in and out; the user can, therefore, supply energy to the network, and this encourages self-production and the use of renewable sources (in particular photovoltaic systems). These meters also allow the application of different tariffs for the time slots, prompting users to use appliances outside peak hours and rightly reflecting the value of electricity on the exchange of energy.

The LENNT meters (Fig 3.7) is equipped with a circuit breaker with magnetothermal protection and release coil.

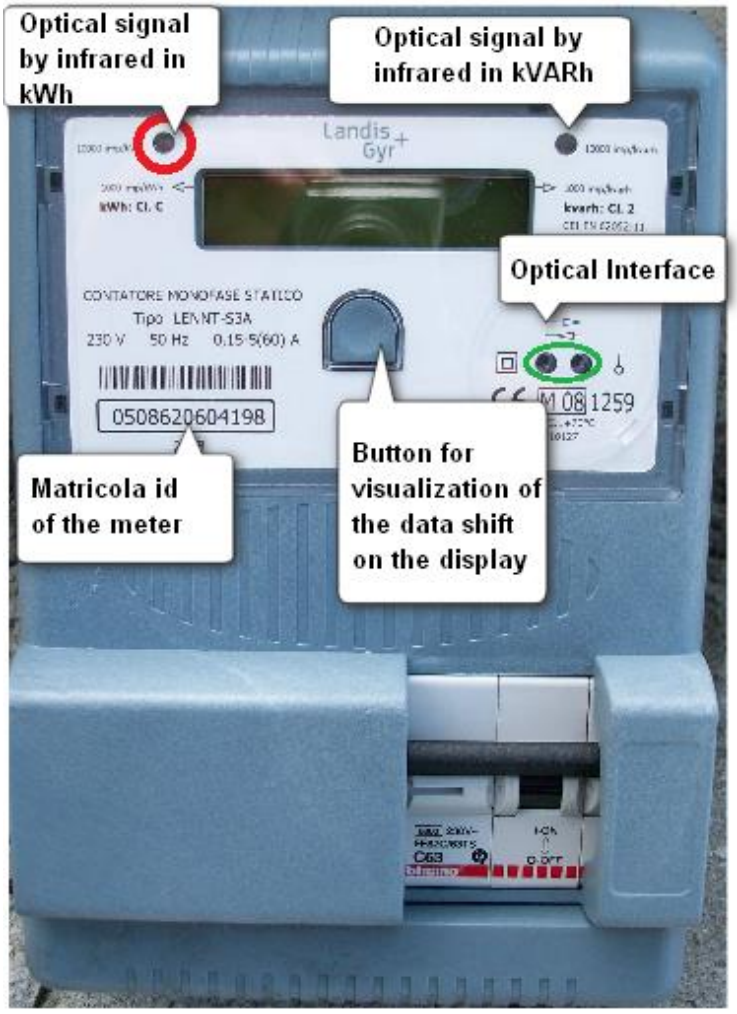


Figure 3-7 LENNT meter [49]

The circuit-breaker can be maneuvered manually at the output and can be controlled by the electronics (locally, in the case of power supercharging, or remotely, on command of the center, in case, and closing of the contract) for opening only. The circuit-breaker can therefore only be closed manually. The possible values for this feature are reported in Table 3.4.

Type of Meter
LENNT_type1
LENNT_type2
GME

Table 3-4 Meter Types

The difference between a three-phase (type2) and a single-phase system (type1) is that in the first case the system is based on three phase cables with the presence of the neutral cable. The single-phase connection is obtained from the three-phase connection using a single phase cable and the neutral one. The production of electricity in the large power stations, as well as transmission and distribution in the territory, take place in three phases for both technical and economic reasons. For domestic systems, a single-phase system is normally used, derived from the three-phase current that arrives at the transformer station. the two types of LENNT is also related to the type of fee, while GME is used for customers powered by high and medium voltage.

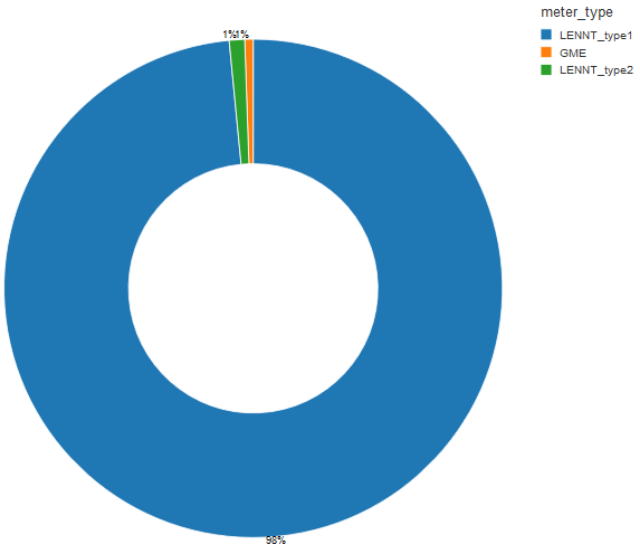


Figure 3-8 Meter type distribution

A global view of the meter type variable (Fig 3.8) show that the 98% are LENNT type 1, this variable is almost a constant for this dataset.

3.2.5 Status customer

The status customer is a simple binary variable that tell us if the customer has an active contract, the dictionary of this variable in Table 3.5

Status Customer	Value
Inactive customer	0
Active customer	1
Unknow status	2

Table 3-5 Status Customer Values

In Fig 3.9 there may be a growing trend of monthly active users among the 2017, every month it can be granted at least 1.287.557 active users. Vice versa the trend of inactive ones decreases until it reaches a value of 32.421 missing entry for the feature in December.

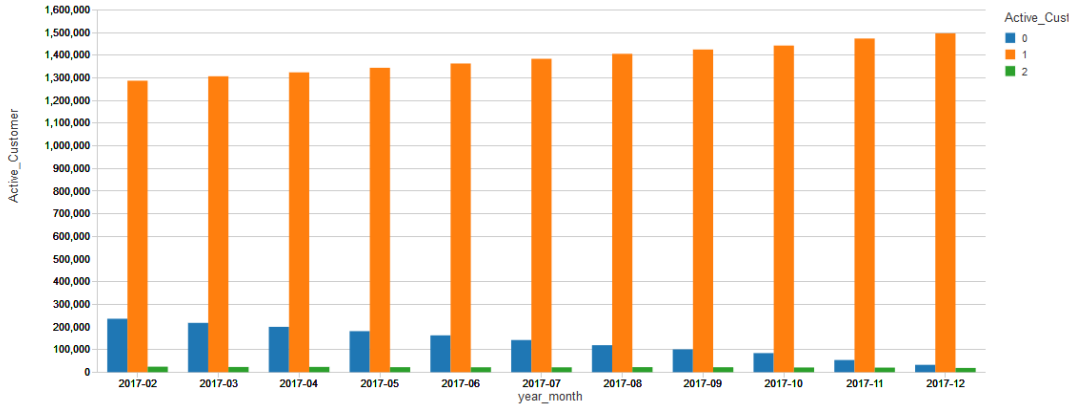


Figure 3-9 Trend of monthly status customer

3.3 Metering Dataset

The dataset (data dictionary in Table 3.6) contain 1.547.939 customers with a monthly records of cumulative energy consumption for a period of 24 months, i.e. from January 2016 to December 2017.

Field Name	Type	Description
Id_customer	String	POD: Alphanumeric national code that uniquely identify the final customer
year_month	String	The year and month in reference to the single record
Tot_active_energy	String	Cumulative consumption until specific year-month

Table 3-6 Data Dictionary on Metering Dataset

According to [50], the mainly system for monitoring energy are Active and Reactive Energy and Power Factor.

Active energy is that which is transformed into work and heat by electrical devices. Devices such as incandescent light bulbs only absorb active energy. The unit of measure is the kWh (kilowatt hour). It is the unit of measurement of electrical energy; it represents the energy absorbed in 1 hour by a device with the power of 1 kW. In the bill, electricity consumption is billed in kWh.

The reactive energy is that portion of energy that instead of being consumed immediately by the user is stored for a few fractions of a second and released into the electric network. The use of reactive energy concerns equipment that needs a magnetic field to work, such as electric motors, fluorescent lamps (neon), electronic devices (television, computers, etc.). The unit of measurement for reactive energy is the varh (Volt-Ampere Reactive hour). This energy is not commercialized; therefore, a moderate consumption of reactive energy is to be considered as physiological. A maximum quantity of reactive energy sampling is tolerated, currently valid only for supplies above 16.5 kW, beyond which a penalty is triggered. The parameter that is normally taken into consideration to check whether the system has too high a reactive energy consumption is the power factor or cos φ. This parameter evaluates the link between active energy and reactive energy and in the case of an ideal load, only resistive and therefore without consumed reactive energy, it is 1. Reactive energy sampling is considered normal until the user has a power factor (cos φ) greater than 0.9. Values below

this limit indicate problems with the system and the simultaneous request for penalties by the electricity distributor with whom the contract was stipulated.

The withdrawal of reactive energy by a user device can be limited or even canceled by means of some simple technological devices installed on the customer's electrical system, in this case, it will need to speak of power factor correction of the electrical system.

3.3.1 Transformation

For the purpose analyze monthly consumption, was decided to pivot the Metering dataset fixing the “Id_customer” as primary key with so creating a feature for each month, every new variable name’s has this pattern “tot_active_energy_yyyy-mm”, where “yyyy-mm” represent the tuple (year, month) the output of this transformation gives a dataframe with 1.547.939 with 24 features. For doing this pivoting was used the function written in PySpark below:

```
##################################################
# FUNCTION NAME: pivot_udf
# version 1.0
##################################################
# Input parameters:

# 1. dataframe, df
# 2. id unique for dataframe, id_df
# 3. the column of time, (e.g. month,year or combination)
# 4. all the columns of your dataframe, *cols
##################################################
# Output parameters:
# 1. pivoted dataframe, df_pivot
##################################################
import pyspark.sql.functions as fn

def pivot_monthly_udf(df,id_df,time_to_pivot,*cols):
    df_pivot = df.select(id_df).drop_duplicates()
    for c in cols:
      df_pivot = df_pivot.join(df.withColumn('combination_cols',fn.concat(fn.lit('{}_'.format(c)),df[time_to_pivot]))\
                               .groupby(id_df).pivot('combination_cols')\
                               .agg(fn.first(c)),id_df)
    return df_pivot
```

Once done the pivoting (code below), the monthly consumption, our target feature, was calculated by the difference between the energy measured on a month minus the energy measured the previous month, an example of display of 2 entries of the dataset is reported in Fig 3.10.

```
Metering_dataset_pivot = pivot_monthly_udf(Metering_dataset,"id_customer","year_month","Tot_active_energy")
```

To execute this command PySpark took 40.48 seconds.

id_customer	tot_consumption_2017-08	tot_consumption_2017-09	tot_consumption_2017-10	tot_consumption_2017-11
16075	2112	3924	4824	4368
16471	null	null	840	624

Figure 3-10 Metering Dataset view

3.3.2 Missing values

In statistics, missing data, or missing values, occur when no data value is stored for the variable in an observation. Missing data are a common occurrence and can have a significant effect on the conclusions that can be drawn from the data.

The amount of missing record for each month is reported below (Fig 3.11), in that Figure is possible see that the minimum number of missing entries over the entire period was about 163.000.

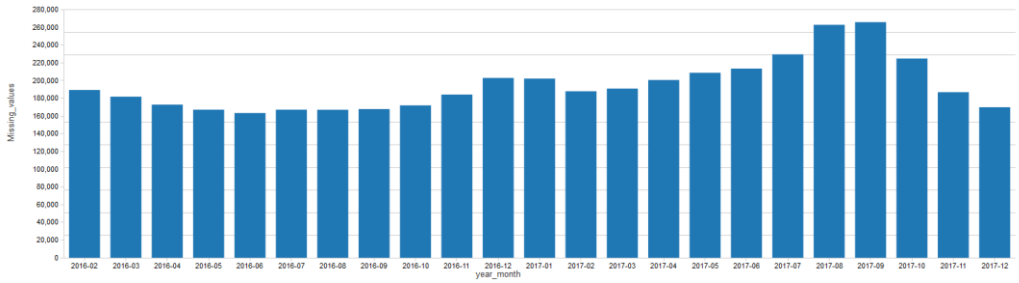


Figure 3-11 Trend of NULL values by month

3.4 Joined Dataset

Before the join, it was decided to remove all new customers registered after the first month, i.e. customer registered after January 2016. This operation was performed by filtering all the costumers with the value 1 in the “status_customer” feature on January 2016, then the feature was dropped.

After that, the Metering dataset has been enriched with 4 features of the Master dataset: “market”, “meter_type”, “contracted_power_band”, “zone_number”.

To execute this command PySpark took 0.07 seconds.

The final result is a dataset with 1.547.939 customers and 28 features: Id_customer, 23 monthly consumptions and the 4 features coming from Master dataset.

4. Data Preprocessing

The setting used on Databricks cluster for the analysis is based on Serverless Pool[14] (beta, R/Python/SQL) cluster type with Databricks Runtime Version 4.0 (includes Apache Spark 2.3.0, Scala 2.11), the Python version is 3 and below the number and type of Driver and Worker:

- 1 Driver r3.xlarge with 30.5 GB Memory, 4 cores, 1DBU
- 3 Worker r3.xlarge with 30.5 GB Memory, 4 cores, 1DBU

4.1 Customer filtering and selection

The main objective of this paragraph are data homogenization and detection of simple anomalies.

Homogenization pretends to group customer who can be compared among each other: customer with similar consumption habits under the period of study.

For this purpose, 4 different operations were performed:

- Given that the meter type LENNT type 1 represent the 98% of the customers, seems like a good choice to keep only customers of this specific type and remove the others. This step filtered out 13.783 customer ids
- Given that the deregulated and protected markets represent the 98% of the customers, only customers of this specific two types of market were retained. This step filtered out 3.750 customer ids
- Look for customers that changed their market along the 2017. Zero found, no customers were filtered out
- Look for customers that changed their contracted band along the 2017. Zero found, no customers were filtered out

The following step performed was the detection of simple abnormalities. This kind of detection pretends to eliminate the customers from the study whose

[14] A serverless pool is self-managed pool of cloud resources that is auto-configured for interactive Spark workloads. It provides the minimum and maximum number of workers and the worker type, and Databricks provisions the computer and local storage based on the user usage.

abnormality is so obvious that they do not need to be further analyzed. There are four kind of obvious abnormality consumption:

- Negative metered energy, zero customer ids found
- Negative consumption, 3.359 customer ids found
- Zero consumption for the whole period, 7.422 customer ids found
- Few number of metered values from the measurements equipment (under 10 from the 23 monthly consumption records), 136.461 customer ids found

The customers successively were merged in order to remove detection id duplication. The number of different customer ids found was respectively 563.963, with the addition of the customers registered after January 2016, was reached a considerable number of 967.324 customer ids remained from the initial number of 1.547.938 of customer population.

4.2 Feature selection and extraction

After all the operation and filter applied, as described in the previous paragraph, the number of missing value for the remaining categorial features “zone_number”, “contracted_power_band” and “market” was respectively 11.528, 14 and 0. All the customers with these Null values in the categorical features were removed, the missing values on the metering data were filled by linear interpolation. At the end of the process the number of selected customers was 955.782.

At this point, the consumption data need to be represented in a normalized scale for further analyze, for M customers {1,2,…,m,…M} over the N month {1,…,h,…,23} a feature matrix is computed, in which element is a daily average kWh consumption feature during that month:

$$x_h^m = \frac{L_h^m}{D_h} \quad (1)$$

Where for customer m, L_h represents the monthly kWh consumption of the month, h and D_h represents the number of days of the current month, a display of the feature matrix is reported in Fig 4.1.

To execute this command PySpark took 0.13 seconds.

The lack of synchronism in the days of month between the readings of a customer make necessary to normalize them, as the meter reading dates affect the monthly kWh consumption recorded for each customer, thus, the daily average kWh consumption values computed using (1) reveals an accurate consumption history of the costumers.

id_customer	cons_dayly_avg_2016-02	cons_dayly_avg_2016-03	cons_dayly_avg_2016-04	cons_dayly_avg_2016-05
3	103.44827586206897	103.35483870967742	87.2	84.38709677419355
4	42.62068965517241	41.806451612903224	41.6	34.45161290322581
5	500.6896551724138	487.35483870967744	348.8	228.7741935483871

Figure 4-1 Daily Consumption dataset view

For the treatment of the categorical features the two algorithms of ETL available on Spark: “StringIndexer” and “OneHotEncoder”. The first one is a label indexer that maps a string column of labels to an ML column of label indices. The indices are in [0, numLabels), ordered by label frequencies. So, the most frequent label gets index 0, this function was applied to all the three features according to Table 4.1 below.

Market	**Market label**	**Zone number**	**Zone number label**	**Contracted power band**	**Contracted power band label**
Protected	0	**Zone_1**	2	B1	3
Deregulated	1	**Zone_2**	1	B2	0
		Zone_3	0	B3	1
		Zone_4	4	B4	2
		Zone_5	3	B5	4
		Zone_6	6	B6	5
		Zone_7	5		
		Zone_8	10		
		Zone_9	9		
		Zone_10	7		
		Zone_11	8		

Table 4-1 Categorical Features and Labels

The function “OneHotEncoder” one maps a column of label indices to a column of binary vectors, with at most a single one-value. This encoding allows algorithms which expect continuous features to use categorical features, this function was applied only to “contracted_power_band” variable.

To execute this command PySpark took 1.20 seconds.

In Fig 4.2 is visible a display of these two functions.

id_customer	zone_number	contracted_power_band	market	contracted_power_band_label	contracted_power_band_encoded	market_label	zone_number_label
4	zone_2	B2	DEREGULATEDMARKET	0	▸[0,5,[0],[1]]	1	1
5	zone_1	B4	DEREGULATEDMARKET	2	▸[0,5,[2],[1]]	1	2
6	zone_1	B2	PROTECTEDMARKET	0	▸[0,5,[0],[1]]	0	2

Figure 4-2 Categorial features view

4.3 Identifying Relevant Attributes

Eleven attributes have been chosen to create a general pattern of power consumption for each client.

- M1, M2, M3, M4 represents the average consumption (in kWh) of a specific client for each semester (where M1 is 1st half 2016 to M4 that is 2nd half 2017)
- MAX represents the maximum consumption (in kWh) of a specific client in the whole period
- DEV1, DEV2, DEV3, DEV4 represent the standard deviation of the consumption of a specific client for each semester
- ZNM represents the average power consumption in the area where the client lives.
- MM represents the average power consumption for the market where the client belongs

The code use is reported below:

```
Consumption_rePiv_enrich.registerTempTable("cs_pivot_selected")
c_max = sqlContext.sql("select id_customer, max(Consumption) as max from cs_pivot_selected group by id_customer")

c_meanMarket = sqlContext.sql("select market_label, mean(Consumption) as meanMarket from cs_pivot_selected group by market_label")
c_meanZone = sqlContext.sql("select zone_number_label, mean(Consumption) as meanZone from cs_pivot_selected group by zone_number_label")

fistband =Consumption_rePiv.filter( (col("year_month") == '2016-02') | (col("year_month") == '2016-03') | (col("year_month") == '2016-04') \
                                  | (col("year_month") == '2016-05') | (col("year_month") == '2016-06'))\
.groupBy("id_customer").mean().withColumnRenamed("avg(Consumption)","M1")
secondband =Consumption_rePiv.filter( (col("year_month") == '2016-07') | (col("year_month") == '2016-08') | (col("year_month") == '2016-09') \
                                   | (col("year_month") == '2016-10') | (col("year_month") == '2016-11')| (col("year_month") == '2016-12'))\
.groupBy("id_customer").mean().withColumnRenamed("avg(Consumption)","M2")
thirdband = Consumption_rePiv.filter( (col("year_month") == '2017-01') | (col("year_month") == '2017-02') | (col("year_month") == '2017-03') \
                                   | (col("year_month") == '2017-04') | (col("year_month") == '2017-05')| (col("year_month") == '2017-06'))\
.groupBy("id_customer").mean().withColumnRenamed("avg(Consumption)","M3")
fourthband = Consumption_rePiv.filter( (col("year_month") == '2017-07') | (col("year_month") == '2017-08') | (col("year_month") == '2017-09') \
                                    | (col("year_month") == '2017-10') | (col("year_month") == '2017-11')| (col("year_month") == '2017-12'))\
.groupBy("id_customer").mean().withColumnRenamed("avg(Consumption)","M4")

fistband1 =Consumption_rePiv.filter( (col("year_month") == '2016-02') | (col("year_month") == '2016-03') | (col("year_month") == '2016-04') \
                                  | (col("year_month") == '2016-05') | (col("year_month") == '2016-06'))\
.groupBy("id_customer").agg(stddev("Consumption")).withColumnRenamed("stddev_samp(Consumption)","Dev1")
secondband2 =Consumption_rePiv.filter( (col("year_month") == '2016-07') | (col("year_month") == '2016-08') | (col("year_month") == '2016-09') \
                                   | (col("year_month") == '2016-10') | (col("year_month") == '2016-11')| (col("year_month") == '2016-12'))\
.groupBy("id_customer").agg(stddev("Consumption")).withColumnRenamed("stddev_samp(Consumption)","Dev2")
thirdband3 = Consumption_rePiv.filter( (col("year_month") == '2017-01') | (col("year_month") == '2017-02') | (col("year_month") == '2017-03') \
                                   | (col("year_month") == '2017-04') | (col("year_month") == '2017-05')| (col("year_month") == '2017-06'))\
.groupBy("id_customer").agg(stddev("Consumption")).withColumnRenamed("stddev_samp(Consumption)","Dev3")
fourthband4 = Consumption_rePiv.filter( (col("year_month") == '2017-07') | (col("year_month") == '2017-08') | (col("year_month") == '2017-09') \
                                    | (col("year_month") == '2017-10') | (col("year_month") == '2017-11')| (col("year_month") == '2017-12'))\
.groupBy("id_customer").agg(stddev("Consumption")).withColumnRenamed("stddev_samp(Consumption)","Dev4")

dataframe = df_final.select(["id_customer"]+df_final.columns[-4:])\
.join(c_max,['id_customer'])\
.join(c_meanMarket,['market_label'])\
.join(c_meanZone,['zone_number_label'])\
.join(fistband,["id_customer"])\
.join(secondband,["id_customer"])\
.join(thirdband,["id_customer"])\
.join(fourthband,["id_customer"])\
.join(fistband1,["id_customer"])\
.join(secondband2,["id_customer"])\
.join(thirdband3,["id_customer"])\
.join(fourthband4,["id_customer"])
```

To execute this command PySpark took 8.34 seconds.

Therefore, each client is represented by a vector with the 11 considered attributes. In other words, this vector represents a pattern of consumption. According to [26] the motivation for choosing them was obtained by analyzing real data from electric utilities and examining some existing fraud detection systems. Two main reasons for using the six-months interval were identified. First, this period was chosen to minimize the effects of adverse events such as vacant properties, season changes, and occupancy increases, among others. Second, it was found that several expert systems have also successfully used this time span to establish relevant rules for the process. Regarding the selected attributes, each one was included to model specific information. Each attribute makes its own contribution to the energy profile. The last two attributes are mutually independent, providing information on the average power consumption in the zone of a particular consumer group and in the type of market. Therefore, each attribute provides different kinds of information to the clustering algorithm (or different perspectives of the available information) and complement each other. By means of a cluster-based classification process, one can mutually validate the intrinsic information carried out by each variable in order to improve the overall system detection rate, a display of a portion of the new dataset is reported in Fig 4.3.

id_customer	max	meanMarket	meanZone	M1	M2	M3	M4
10046	147.8709677419355	101.43908418027668	115.28858546647946	67.11332591768632	76.07956989247312	77.0373271889401	103.5956989247311
10066	122.3225806451613	101.43908418027668	82.17969809438769	81.65659621802001	105.33118279569892	89.73840245775729	91.33763440860214
10193	50.06896551724138	81.37122233561725	76.57677791584895	40.56605116796441	36.380645161290325	34.62657450076805	34.50752688172043

Figure 4-3 General Pattern Consumption dataset view

4.4 Data Normalization

The size of peak load may vary among all costumers. Therefore, all the measured load profiles need to be further normalized to allow their comparison and calculation. A normalize measured load profile is calculated by this formula:

$$z_{\mathrm{h}}^{\mathrm{m}} = \frac{L_h^m - \min(L^m)}{\max(L^m) - \min(L^m)} \quad (2)$$

L_h^m represents the monthly kWh consumption for customer h on the month m and z is the actual value of normalized load profile.

Min-max normalization is a linear normalization technique. It does not change the distribution of the data set and scale the values between 0 and 1.

A display of the new dataset is reported in Fig 4.4.

To execute this command PySpark took 0.24 seconds.

id_customer	cons_dayly_avg_normal_2016-02	cons_dayly_avg_normal_2016-03	cons_dayly_avg_normal_2016-04	cons_dayly_avg_normal_2016-05
10193	1	0.8190839694656488	0.5748346055979644	0.7526717557251907
10066	0.5485688528284878	0.5555555555555556	0.27189542483660134	0
10046	0.40826042202779195	0.4552238805970149	0.07587064676616911	0

Figure 4-4 Daily Consumption Normalized dataset view

4.5 Final datasets

After all the previous step was decided to create three final datasets described below:

- Daily Consumption (DC): 23 daily consumption from the 23 consumptions for month (Fig. 4.1).
- Daily Consumption Normalized (DCN): 23 daily consumption of the previous dataset normalized by the maximum and minimum normalizer and three remained categorical features transformed “market_label”, ”zone_number_label” and “contracted_power_band_encoded” (Fig 4.2-4.4).
- General Pattern Consumption (GPC): Each costumer is defined by the following 11 new features: M1, M2, M3, M4, MAX, DEV1, DEV2, DEV3, DEV4, ZNM and MM (Fig 4.3).

5. NTL Detection methods

This part describes the different methodologies applied with the purpose of find abnormalities or fraud. The strategies are briefly described below:

1. Check-Steps on the consumption pattern on DC
2. Interquartile range on DC
3. Clustering on DNC
4. Clustering on GPC

5.1 Anomalous Consumption

One simple method to detect anomalies consider two different simple check-step, this method marks as suspicious all those customers that have a consumption greater than 2 kWh (for a round matter) and status customer equal to 0, thus inactive. The second ones signal all those customers that present negative consumption. All these ids will give aside as a list of obvious suspicious consumers.

5.2 Interquartile Range Method

The second method encompass the interquartile range. The abbreviated "IQR", is just the width of the box in the box-and-whisker plot. That is, IQR = $Q_3 – Q_1$. The IQR can be used as a measure of how spread-out the values are. Statistics assumes that the values are clustered around some central value. The IQR tells how spread out the "middle" values are; it can also be used to tell when some of the other values are "too far" from the central value. These "too far away" points are called "outliers", because they "lie outside" the range in which had expected them. The IQR is the length of the box in the box-and-whisker plot. An outlier is any value that lies more than one and a half times the length of the box from either end of the box. That is, if a data point is below $Q_1 – 1.5×IQR$ or above $Q_3 + 1.5×IQR$, it is viewed as being too far from the central values to be reasonable. The values for $Q_1 – 1.5×IQR$ and $Q_3 + 1.5×IQR$ are the "fences" that mark off the "reasonable" values from the outlier values, outliers lie outside the fences.

It this paragraph was considered not the outliers but the "extreme values", then the values for $Q_1 - 1.5 \times IQR$ and $Q_3 + 1.5 \times IQR$ are the "inner" fences and the values for $Q_1 - 3 \times IQR$ and $Q_3 + 3 \times IQR$ are the "outer" fences.

The IQR of the consumption was calculated, using the DC dataset, for each customer and thereby compared with the current value of the consumption, this comparison is reported in a new column named “check_filtering_IQR”. The code implemented is:

```
#To automate the process of finding outliers by the IQR method,

Consumption_rePiv.registerTempTable("cs_pivot_selected")

df_quartile_1 = sqlContext.sql("select id_customer, percentile_approx(Consumption,0.25) as quartile_1 from cs_pivot_selected group by id_customer")
df_quartile_2 = sqlContext.sql("select id_customer, percentile_approx(Consumption,0.50) as quartile_2 from cs_pivot_selected group by id_customer")
df_quartile_3 = sqlContext.sql("select id_customer, percentile_approx(Consumption,0.75) as quartile_3 from cs_pivot_selected group by id_customer")

dataframe = Consumption_rePiv.join(df_quartile_1,['id_customer']).join(df_quartile_3,['id_customer']).join(df_quartile_2,['id_customer'])
dataframe = dataframe.withColumn('IQR', col('quartile_3') - col('quartile_1')).withColumn('lower_bound', col('quartile_1') - 3 * col('IQR'))\
                                                                               .withColumn('upper_bound', col('quartile_3') + 3 * col('IQR'))
dataframe = dataframe.withColumn('check_filtering_IQR', when((col('Consumption') < col('lower_bound')) | (col('Consumption') > col('upper_bound')),\
                                                             lit(1)).otherwise(lit(0)))
```

To execute this command PySpark took 1.62 seconds.

A display of the output for a single costumer is showed in Fig 5.1.

id_customer	year_month	Consumption	quartile_1	quartile_3	quartile_2	IQR	lower_bound	upper_bound	check_filt
10096	2016-02	97.65517241379311	75.48387096774194	101.14285714285714	93.6	25.658986175115203	24.16589861751153	152.46082949308754	0
10096	2016-03	96	75.48387096774194	101.14285714285714	93.6	25.658986175115203	24.16589861751153	152.46082949308754	0
10096	2016-04	87.2	75.48387096774194	101.14285714285714	93.6	25.658986175115203	24.16589861751153	152.46082949308754	0
10096	2016-05	85.16129032258064	75.48387096774194	101.14285714285714	93.6	25.658986175115203	24.16589861751153	152.46082949308754	0

Figure 5-1 IQR detection dataframe view

All the consumers that present at least one month with consumption marked as extreme values was selected for the creation of another dataset and then pivoted by “pivot_monthly_udf”. This new dataset contains only the “Id_customer” and “check_filtering_IQR” for each month as a binary vector.

Then by using a user-defined function that count the consecutive one in an array, was calculated, for each costumer, the longest number of consecutive one namely month with extreme values. The costumers with at least 4 consecutive extreme values was labeled as suspicious customers.

The code follows:

```
dataframeToPiv = dataframe.filter(col("check_filtering_IQR")==1)
dataframe_1 = pivot_monthly_udf(dataframeToPiv,"id_customer","year_month","check_filtering_IQR")
vecAssembler = VectorAssembler(inputCols=dataframe_1.columns[1:], outputCol="features")
consAssembler = vecAssembler.transform(dataframe_1.na.fill(0)).select('id_customer', 'features')
dataset_count_ones = consAssembler.rdd.map(lambda x : [x[0],consecutive_one(x[1])] )
dataset_count_ones.cache()
label_2_df = dataset_count_ones.filter(lambda x: int(x[1]) > 4)
label_2_Id = sqlContext.createDataFrame(label_2_df.map(lambda x : Row(x[0])),["id_customer"])
```

To execute this command PySpark took 6.44 minutes.

The following image (Fig 5.2) show the daily consumption trend in average of those suspicious customers.

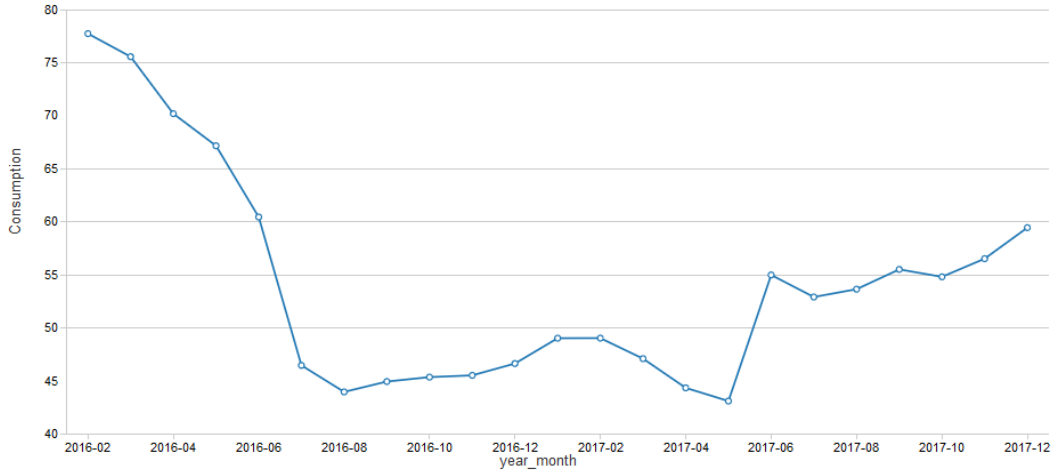


Figure 5-2 DC on average by month of suspicious customer found by IQR Method

As affirm [19], a drastic drop of the consumption can be due to a real slope of the consumers like a change of type of contract or by a different use of the consumed energy. This kind of slope can be due to a failure in the measurement equipment or voluntary alterations of the equipment, both case generate NTLs to the company and a loss of money for it, this kind of detection gives 1.117 customers.

5.3 Clustering

Clustering is the task of assigning entities into groups based on similarities among those entities. The goal is to construct clusters in such a way that entities in one cluster are more closely related, i.e. similar to each other than entities in other clusters. As opposed to classification problems where the goal is to learn based on examples, clustering involves learning based on observation. For this reason, it is a form of unsupervised learning task.

There are many different clustering algorithms and a central notion in all of those is the definition of ’similarity’ between the entities that are being grouped. Different clustering algorithms may have different ways of measuring the similarity. In many clustering algorithms, another common notion is the so-called cluster center, which is a basis to represent the cluster. For example, in K-means

clustering algorithm, the cluster center is the arithmetic mean position of all the points in that cluster.

As an exploratory data analysis tool, clustering has many application areas across various disciplines including social sciences, biology, medicine, business & marketing and computer sciences. Below are some examples [52]:

- Use clustering to group entities based on certain features and then analyze if other features in each group are also close to each other. An example is grouping of tumor DNA microarray data based on gene expressions and then inferring if those groups will imply presence of certain types of cancer. Another example is grouping of patient data based on symptoms and signs and then deducing if those groups will differ from each other with respect to their therapeutic responsiveness or their prognosis.
- Use clustering to group entities into a single cluster only to calculate a center. The center can later be utilized as a representative of all the entities in the group. An example is image compression where an image is first partitioned into small blocks of predetermined size and a cluster center is calculated for the pixels in each block. Then, the image is compressed where each block is replaced by an equally sized block approximated by block's center.
- Use clustering to reduce the amount of data for simplification of analysis. For example, grouping of patient laboratory results based on measured variables (qualitative or quantitative analytes or mathematically derived quantities) may help in understanding how lab data is structured for patients with certain diseases. Another example is segmentation i.e. pixel classification of medical images in order to aid in medical diagnosis.
- Use clustering to solve a classification problem. For example, MRI images of liver belonging to Cirrhosis patients at different stages and non-patients (i.e. free of Cirrhosis) are clustered into two groups, one representing cirrhotic and the other one representing non-cirrhotic cases. Then, MRI image of a new patient is compared to the cluster centers and based on proximity a prediction is made whether the patient is cirrhotic or not.

5.3.1 K-means Algorithm

K-means is among the most popular clustering algorithms [52]. Number of clusters, k, is defined in advance. The centers of clusters and the data points in each cluster are adjusted via an iterative algorithm to finally define k clusters. There is an underlying cost minimization objective where the cost function is so-called Within-Cluster Sum of Squares (WSS). Spark MLlib library provides an implementation for K-means clustering.

5.3.2 Distance intra-cluster Method

The first method that use clustering, in this thesis, sets up the use of distance between points and cluster center, on dataset DCN. In this method was calculated the distances between the points and the cluster centers and fixed a threshold above which every point can be flagged as general errors or potential fraud costumer.

As was already said the k-means algorithm needs to fix the number of cluster k in advance, unfortunately, there is no definitive answer to this problem. The optimal number of clusters is somehow subjective and depends on the method used for measuring similarities and the parameters used for partitioning.

The mainly method to find the right number of cluster: the elbow method and the average silhouette. The first one method looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster does not improve much better the total WSS. The second ones, the Average Silhouette method computes the average silhouette of observations for different values of k. The optimal number of clusters k is the one that maximize the Average Silhouette over a range of possible values for k. Unlikely this second method is not scalable, since it uses pairwise distances, and this will always take O(n^2) time to compute.

The optimal number of clusters in the Elbow method can be defined as follow:

1. Compute clustering algorithm (e.g., k-means clustering) for different values of k.
2. For each k, calculate the total within-cluster sum of square (WSS).
3. Plot the curve of WSS according to the number of clusters k.
4. The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

In this paragraph, the clustering algorithm used was k-means with a range for the number to cluster that goes from 5 to 30, in Fig 5.3 is possible to see the trend of WSS for each k (each iteration used only the 25% of the rows for time related problems).

To execute this command PySpark took 32.05 minutes.

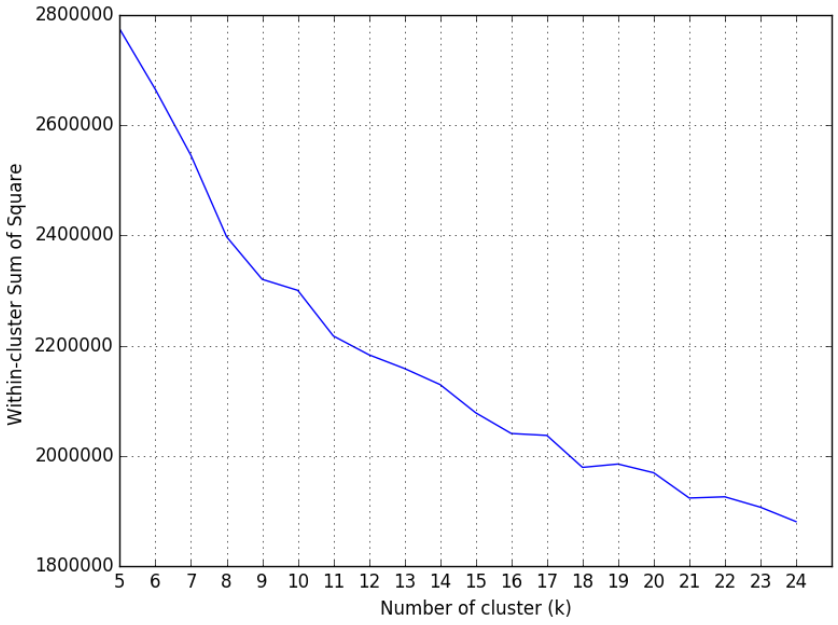


Figure 5-3 Elbow method with K-means

Therefore, for k=11 the WSS tends to change slowly and remain less changing as compared to other k's, so for this dataset 11 should be a good choice for number of clusters.

Hence, at this point, it was used a k-means with k equal to 11, the configuration for the "initMode" parameter that regard the initialization algorithm was set to "k-

means||" with 4 initial step, the convergence tolerance and the maximum number of iteration was respectively set to 0.001 and 100.

The code implemented is showed below:

```
vecAssembler = VectorAssembler(inputCols=DCN.columns[1:], outputCol="features")
consAssembler = vecAssembler.transform(DCN).select('id_customer', 'features')

kmeans_15 = KMeans(featuresCol="features",k=11, seed=1, initMode='k-means||', initSteps=4, tol=0.001, maxIter=100)
model_kmeans_15= kmeans_15.fit(consAssembler)

cons_transformed = model_kmeans_15.transform(consAssembler).select("id_customer","features", "prediction")
```

To execute this command PySpark took 1.05 minutes.

"VectorAssembler" is a transformer that combines a given list of columns into a single vector column. It is useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models.

The Fig 5.4 shows the trend of the daily consumption normalized for each cluster consider for every single point the average of the daily consumption for costumers that belong to that specific cluster in that specific month, in Table 5.1 was reported the cluster's size.

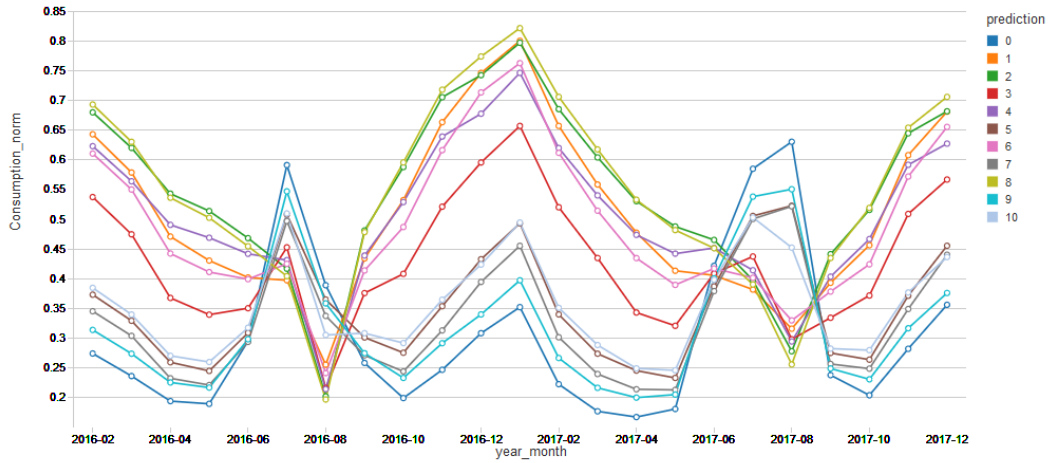


Figure 5-4 DCN on average by month of clusters

Then was calculated all the distances between the customers and the respective center point of each cluster in order to find a good threshold for separate them. Was decided to put the threshold customized for every cluster at 85% of the maximum distance in the current cluster.

Cluster	Size
0	110.580
1	64.841
2	122.222
3	67.447
4	124.334
5	108.273
6	82.230
7	59.040
8	50.816
9	116.965
10	48.222

Table 5-1 Cluster number and sizes

This threshold flagged 2.484 customers and the figure below (Fig.5.5) showed the daily consumption trend in average of those suspicious customers.

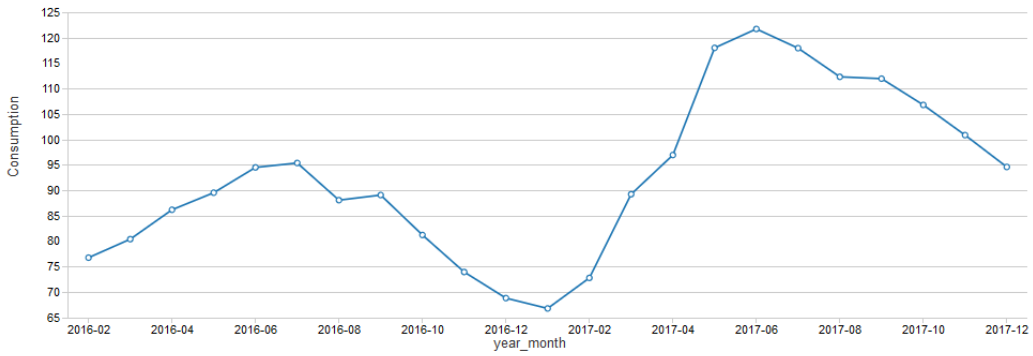


Figure 5-5 DC on average by month of suspicious customer found by First Clustering Method

The performance of the consumption could suggest that the algorithm label as suspicious all those costumers that experiment an abrupt change in the consumption, in this case, an increased consumption starting from January 2017.

The following figures (Fig 5.6-8) will show all the histograms of the distances between the points and the center of the cluster with the threshold highlighted by a vertical blue line. In Table 5.2 is possible see how many anomalous costumers was found by each cluster.

Cluster	Size
0	316
1	140
2	252
3	520
4	130
5	225
6	93
7	200
8	114
9	302
10	192

Table 5-2 Cluster number and sizes of anomalous ids

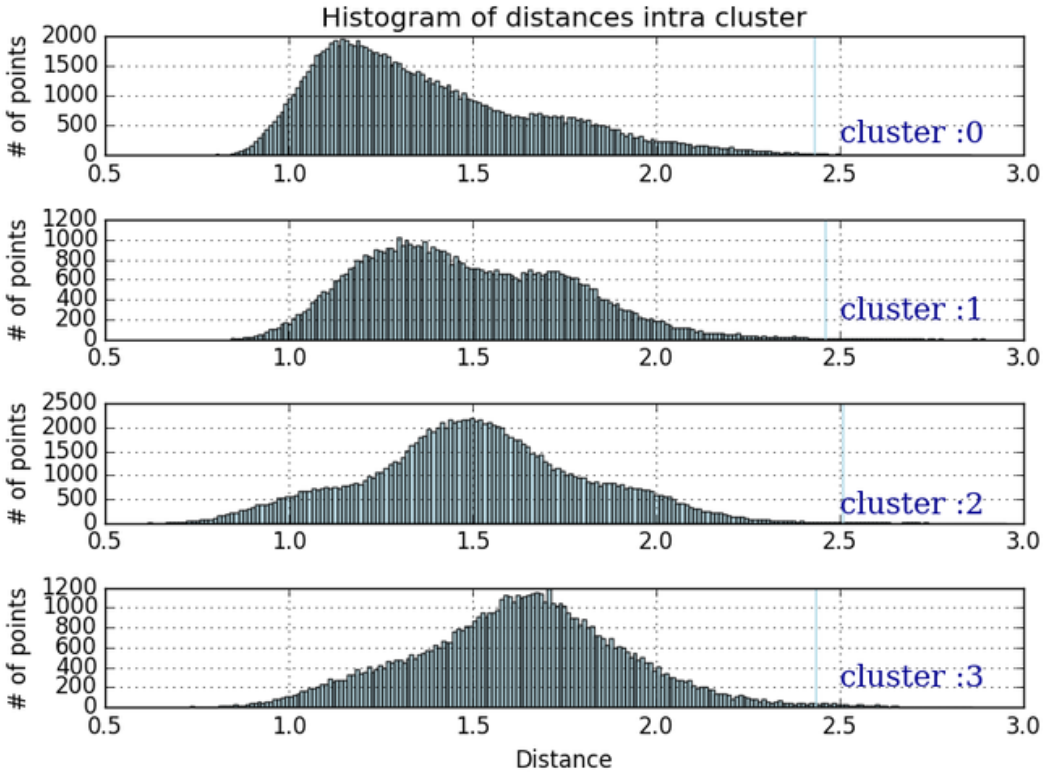


Figure 5-6 Histograms of the distances within cluster 0-3

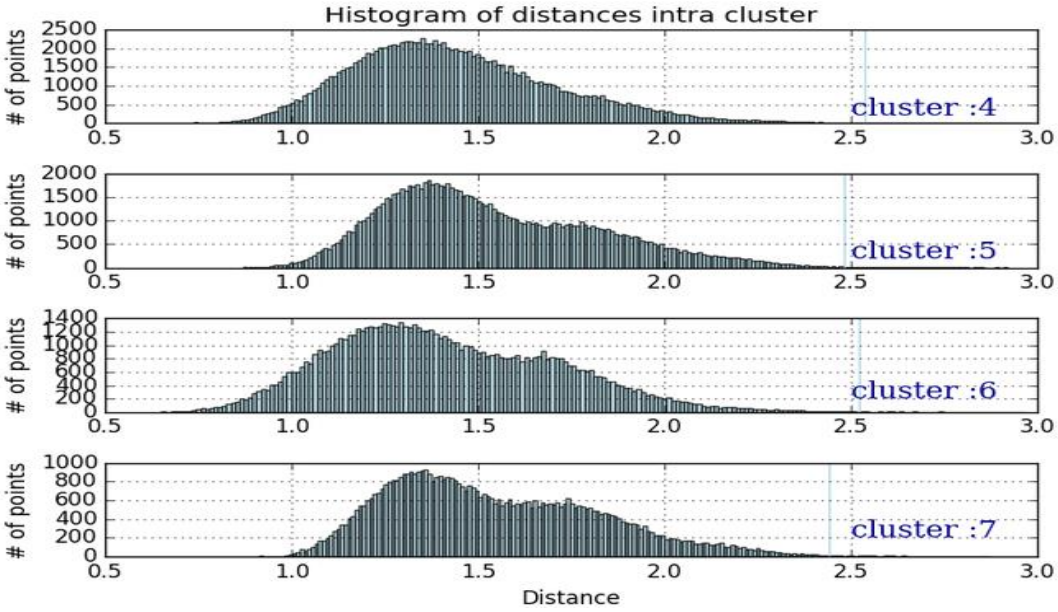


Figure 5-7 Histograms of the distances within cluster 4-7

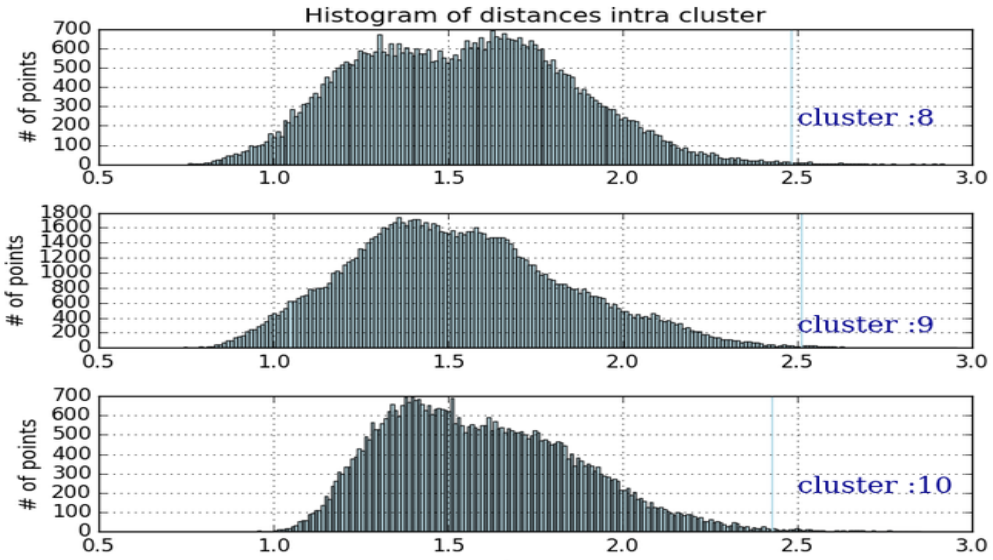


Figure 5-8 Histograms of the distances within cluster 8-10

The next figures (Fig 5,9-11) focus on three cluster in particular:

- 0 and 3 because provide the largest number of anomalous
- 6 because provide the smallest number of anomalous

This focus has as goal to underline the different trend within the cluster for the customers above the threshold and those below signed respectively as 1 and 0.

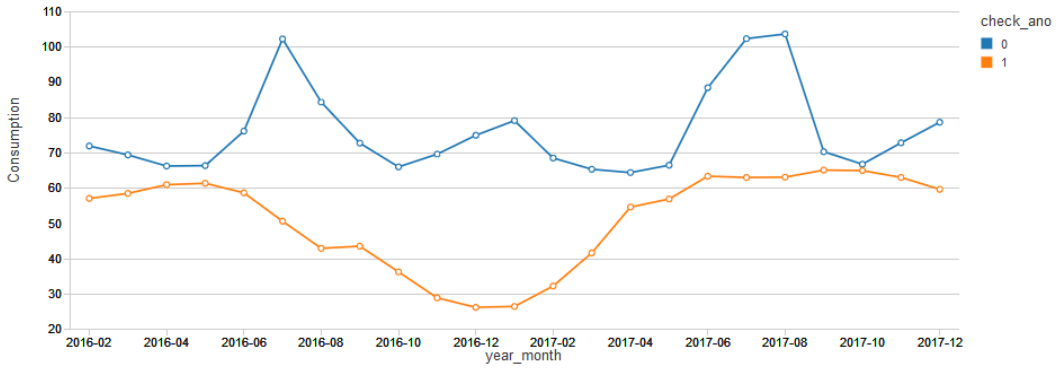


Figure 5-9 Focus on DC of customer belonging to cluster 0

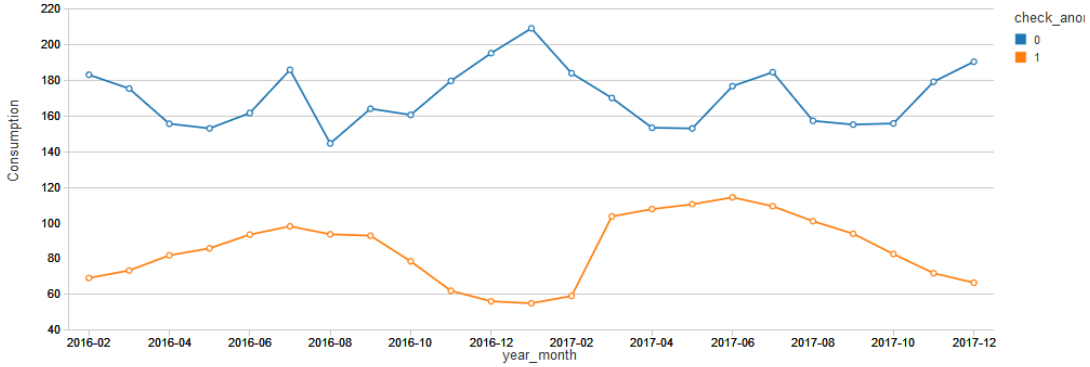


Figure 5-10 Focus on DC of customer belonging to cluster 3

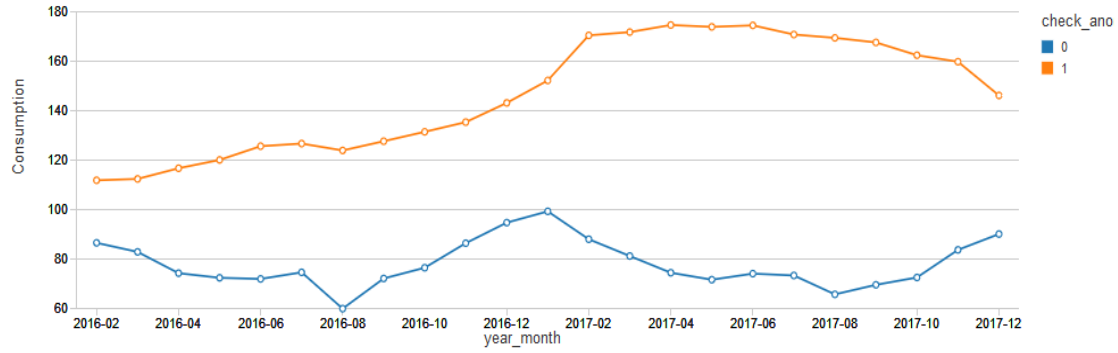


Figure 5-11 Focus on DC of customer belonging to cluster 6

5.3.3 Cluster Less Numerous Method

The second method of clustering was performed on the GCP dataset that comes out by the feature engineering, this method label as errors or potential fraud all those customers belonging to the smallest clusters.

Also, here the clustering algorithm is the k-means and the method to choose the best k is the Elbow method displayed in Fig 5.12.

To execute this command PySpark took 30.52 minutes.

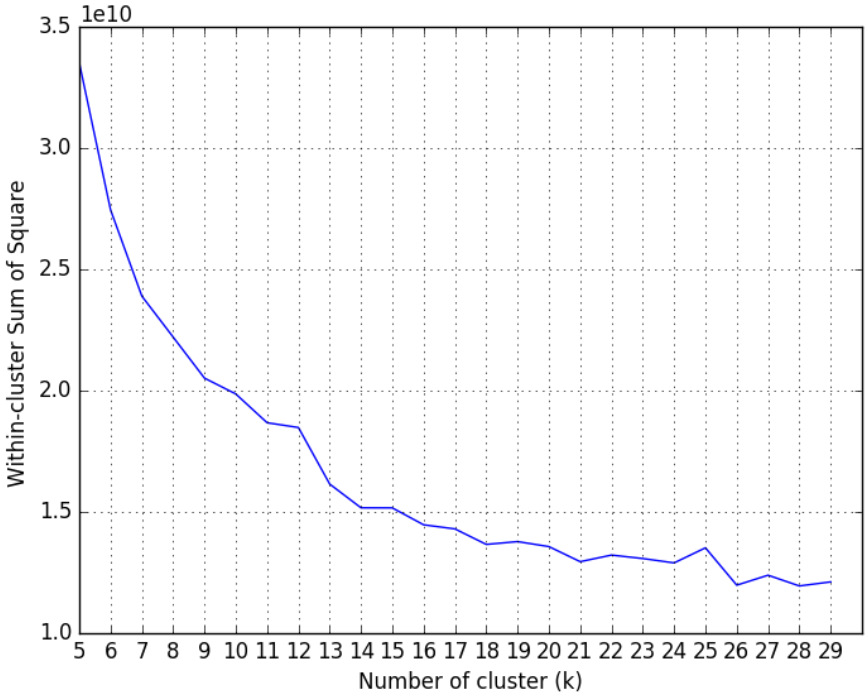


Figure 5-12 Elbow method for K-means

For k equal to 15 the WSS tends to change slowly and remain less changing as compared to other k's but for this type of approach can be better choice a k bigger at a next step along the slope, so for this data 18 should be a good choice.

With the same configuration of the first method, the k-means was ran with a tolerance converge rate of 0.001 and a maximum number of iteration at 100.

To execute this command PySpark took 54.07 seconds.

The following figure (Fig 5.13) shows the trend of the daily consumption normalized for each cluster as in the first method, in Table 5.3 instead, the cluster's size.

What is easy emerge at glance is that the cluster with less number of costumers conveys such a strange consumption trend, in the face of this evidence, was decided to label those customers belonging to the cluster number 4,10,11,12,14 and 17 as suspicious customers.

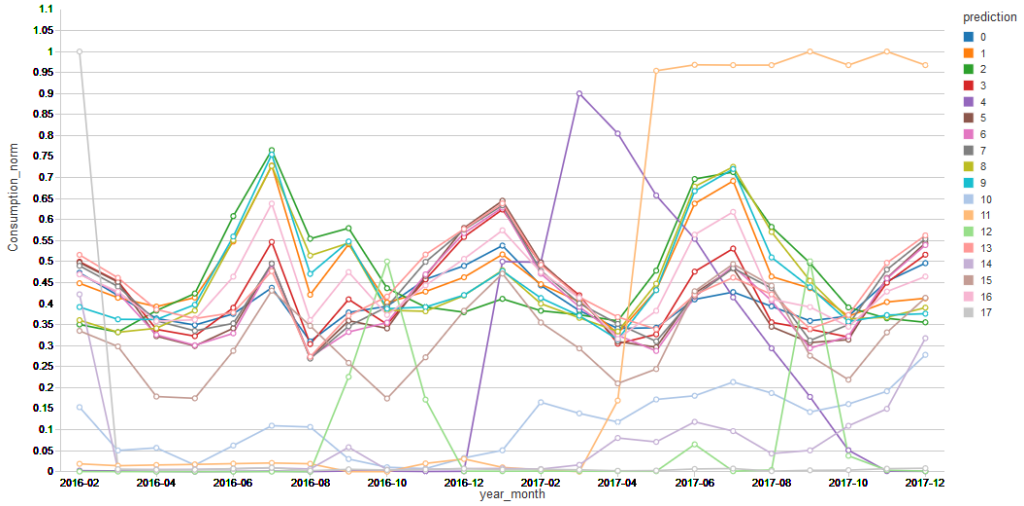


Figure 5-13 DCN on average by month of clusters

Cluster	Size
0	259.358
1	3.604
2	204
3	12.622
4	2
5	29.259
6	84.664
7	218.853
8	766
9	1.831
10	44
11	1
12	2
13	337.557
14	19
15	632
16	6.362
17	2

Table 5-3 Cluster number and sizes

A further support of the method is demonstrated in Fig 5.14 below, indeed removed those customers the cluster is just based on the daily consumption.

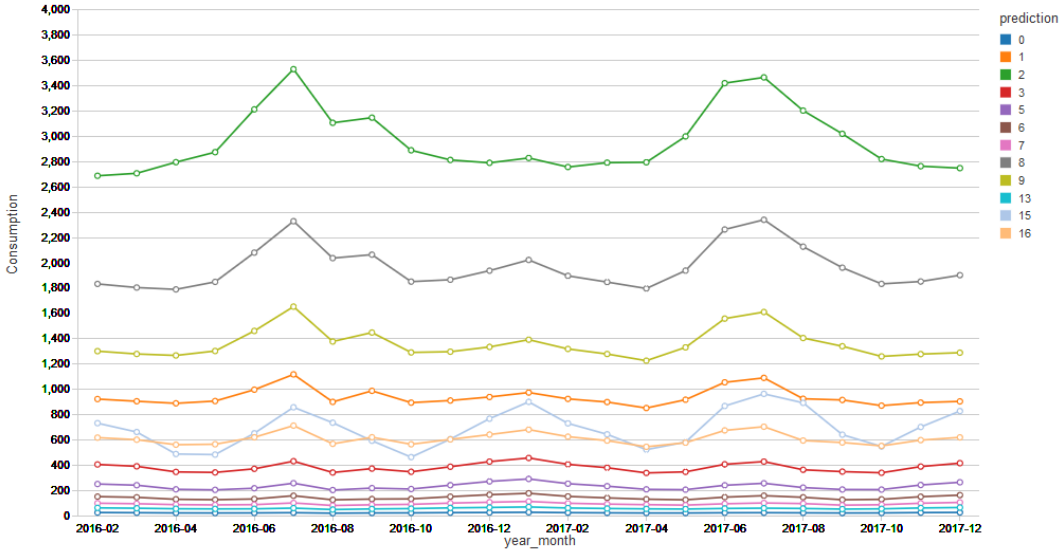


Figure 5-14 DCN on average by month of clusters removed the suspicious clusters

This threshold flagged 70 customers in total and the figure below (Fig. 5.15) shows the daily consumption trend in average of those suspicious customers.

Also, here, in this case, the clustering algorithm highlighted a particular behavior of costumers, a drastic drop during the first five months an sudden increase starting from December 2016.

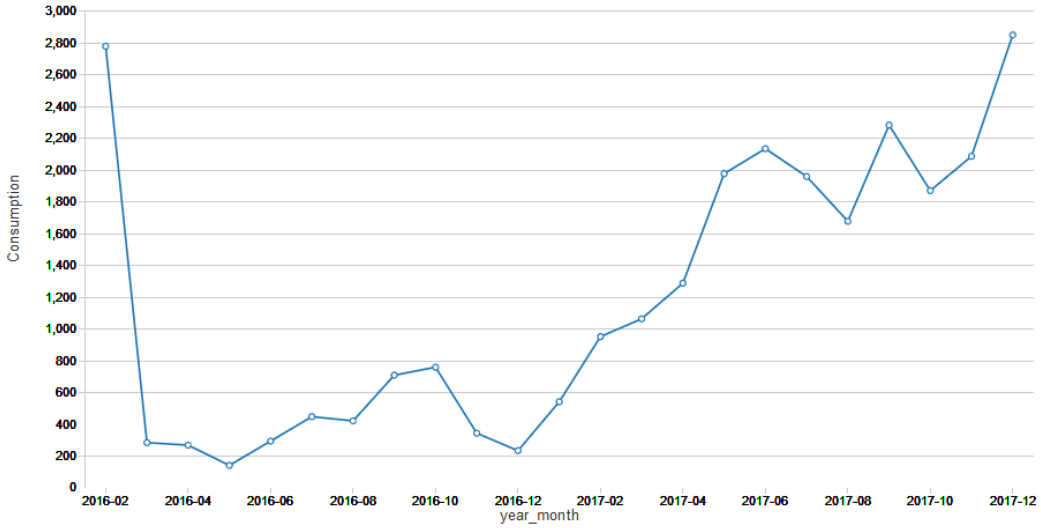


Figure 5-15 DC on average by month of suspicious customer found by Second Clustering Method

5.4 NTL detection methods result

Prior to sending the customers for inspection, the results of all detection methods were cross-checked in order to examine how many customers were matched and to ensure that the different algorithms were not redundant. Thus, after merging the customers detected with each method, the results showed in Table 5.4 were obtained.

Detection method	**Size**
IQR method	1.117
Distance Intra-Cluster method	2.484
Cluster Less Numerous method	70
Total	3.671
Once merged the results of the previous methods	3.615

Table 5-4 Customer selected to be inspected

As is evident from the table only 56 customers from the 3.671 selected customers were detected by more than one methods. Thus, we could deduce that each algorithm detected a type of different patterns of NTL.

Thus, a list of 3.615 customers with an evident and suspicious pattern of consumption with NTL was obtained. These cases of NTL could be due to a drop of electrical demand for their business. In summary, a complete flow chart is showed in Fig. 5.16.

In this diagram, is possible to observe the global scheme and the different steps for the detections of the NTLs. These results are considered very satisfactory considering as the little input information used in the algorithms (basically the evolution of the consumption of the customer and the type of contract)

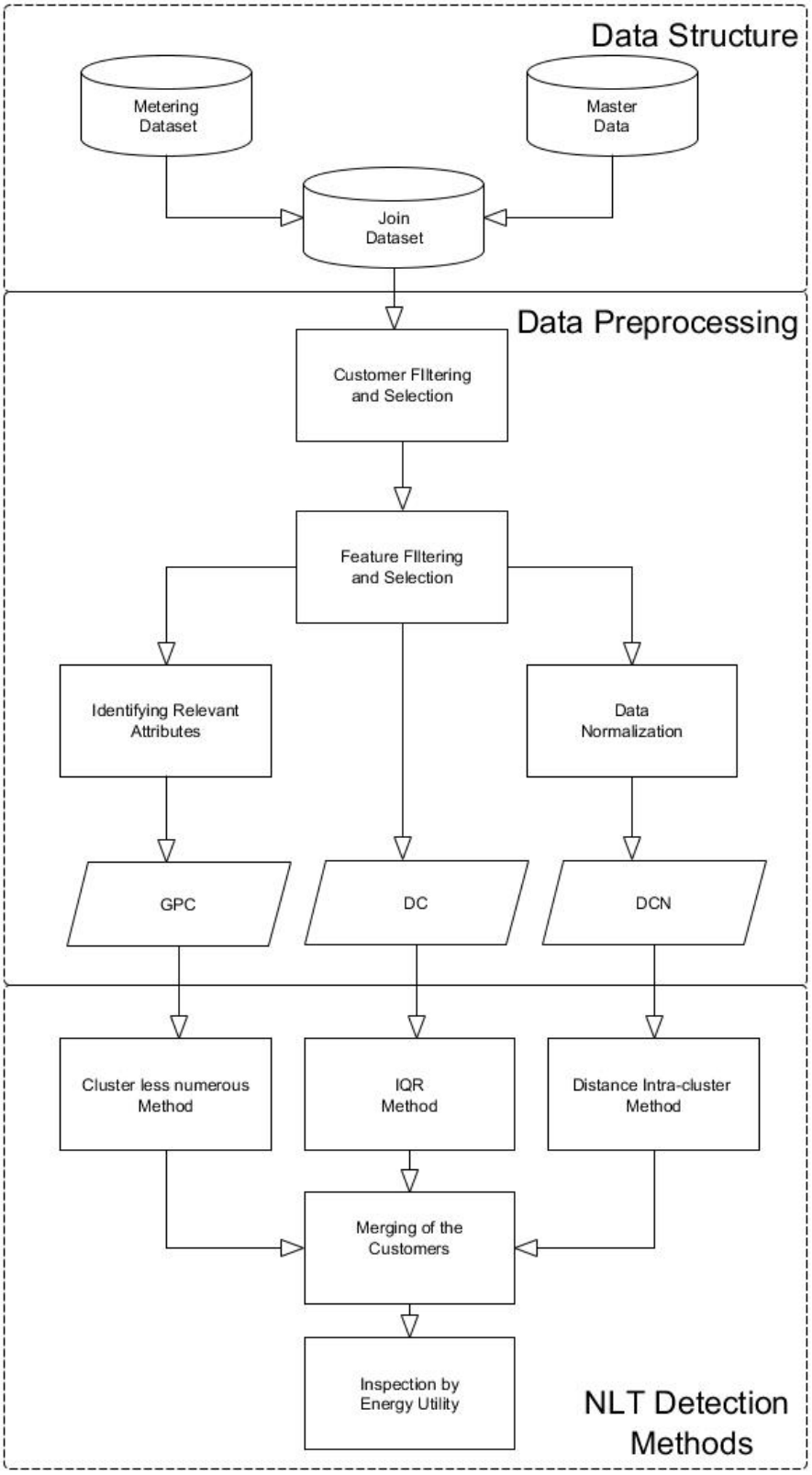


Figure 5-16 Flow Chart of the detection process

6. Conclusion

The work on this thesis supervised by INNAAS[15] was a really valuable experience in several aspects. First of all, as I worked for a company, it taught me what it means to work with qualified people for achieving results, and it taught me how to deal and navigate in the complex environment of Big Data. In addition, it gave me the chance to measure up with a real data problem full of many problems encountered along the work process. The time spent on this elaboration enabled me to learn more in depth the interesting Spark platform and the PySpark programming language, Spark is a milestone for the Big Data Analytics and a very greatly sought skill for a data scientist.

The benefits of the usage of Apache Spark for the elaboration of complex MLlib algorithms on Big Data are multiple. Throughout the thesis was reported the real-time performance to compute the main part of the PySpark code of the project.

Command	**Time**
Pivoting Metering Dataset	40.48 sec
Join Dataset	0.07 sec
Feature Selection and Extraction	0.13 sec
Treatment Categorical Features	1.20 sec
Identifying Relevant Attributes	8.34 sec
Data Normalization	9.24 sec
Creation dataframe 1° Method	1.62 sec
1° Method	6.44 min
Find Best K 2° Method	32.05 min
2° Method	1.05 min
Find Best K 3° Method	30.52 min
3° Method	54.07 sec
Total Time Estimated	71.98 min

Table 6-1 Commands and relative time to computing

[15] http://www.innaas.com/

In Table 6.1 is summarized all the principal steps with the relative time in order to provide an estimation of the time needed to apply this kind of approach with this technology.

The total time requested for this work, considering the about 1.6 millions of customers, is really valid considering its possible future implementation. This thesis was proposed a methodology based on IQR and clustering technique for classifying abnormalities in the profiles of energy consumption.

The main contributions of this paper are as follows:

- Deploy a simple method that requires basically only the historical consumption, which is appropriate for most practical distributions systems.
- Promote Apache Spark as a unified platform for Big Data Analytics
- Some parameter can be adjusted by the user like:
  - The number of the month out of the interquartile distance that labels as suspicious.
  - The threshold distance in the first clustering method.
  - The number of clusters to consider in the second clustering method.
- The utility using this method to guide inspection can actually increase the detection hitrate and the financial return.
- The proposed methods are unsupervised and, therefore does not depend on rules. It can be applied to any distribution utility.

As conclusions, it is necessary to remark that NTL is an important issue in power utilities because it has a high impact on company profits. Despite this, nowadays the methodology of detection of NTLs of the companies is very limited since these companies use detection methods that do not exploit the use of data mining techniques. Different methods to detect NTLs have been developed and tested on a real database supplied by the Energy Company. Concretely, in this thesis, a line of work based on 3 different methods has been presented for the detection of NTLs.

A possible line of work in the future might be the application of different and more complex input parameters or other data mining techniques as well as the

integration of human expert knowledge in these new techniques in order to improve the results.

7. References

1. Sankari, E., Siva, and Rajesh, R., Matheswaran, P., "Detection of Non-Technical Loss in Power Utilities using Data Mining Techniques.", International Journal for Innovative Research in Science & Technology, 1(9). 97-100.

2. Han, W., and Yang X.. "Design a fast Non-Technical Loss fraud detector for smart grid.", Security and Communication Networks , 9(18). 5116-5132.

3. Protalinski, E. (2018). *Smart meter hacking tool released | ZDNet*. [online] ZDNet. Available at: https://www.zdnet.com/article/smart-meter-hacking-tool-released/.

4. Paul, C. R., "System loss in a Metropolitan utility network.", Power Engineering Journal, 1(5). 305-307.

5. Davidson, I. E., Odubiyi, A., Kachienga, M. O., Manshire, B., "Technical loss computation and economic dispatch model for T&D systems in a deregulated ESI.", Power Engineering Journal, 16(2). 55-60.

6. Tenaga Nasional Berhad. Annual Report Tenaga Nasional Berhad 2004. Kuala Lumpur, KL: TNB, 2004.

7. Smith, T. B., "Electricity theft: a comparative analysis.", Energy policy, 32(18). 2067-2076.

8. Alam, M. S., Kabir, E., Rahman, M. M., Chowdhury, M. A. K., "Power sector reform in Bangladesh: Electricity distribution system.", Energy, 29(11). 1773-1783.

9. Saxena, A. K., "Decision priorities and scenarios for minimizing electrical power loss in an Indian power system network.", Electric Power components and systems, 31(8). 717-727.

10. Shrestha, R. M., and Azhar, M., "Environmental and utility planning implications of electricity loss reduction in a developing country: A

comparative study of technical options.", International Journal of Energy Research, 22(1). 47-59.

11. Nizar, A. H., Dong, Z. Y., Wang, Y., "Power utility nontechnical loss analysis with extreme learning machine method.", IEEE Transactions on Power Systems, 23(3). 946-955.

12. Davidson, I. E., "Evaluation and effective management of nontechnical losses in electrical power networks." in Africon Conference, (George, South Africa, 2002), IEEE, Vol. 1

13. Mano, R., Cespedes, R., and Maia, D., "Protecting revenue in the distribution industry: a new approach with the revenue assurance and audit process." in Transmission and Distribution Conference and Exposition, (San Paolo, Brazil, 2004), IEEE

14. Krishna Rao, M., V., and Miller, S., H., "Revenue improvement from intelligent metering systems." in Metering and Tariffs for Energy Supply, (Birmingham, UK ,1999), IET, 218-222.

15. Dick, A. J., "Theft of electricity-how UK electricity companies detect and deter." in Security and Detection, (Brighton, UK, 1995), IET, 90-95.

16. Fourie, J. W., and Calmeyer, J. E., "A statistical method to minimize electrical energy losses in a local electricity distribution network." in AFRICON, 2004. 7th AFRICON Conference in Africa, (Gaborone, Botswana ,2004), IEEE, Vol. 2.

17. Cabral, J. E., and Gontijo, E. M., "Fraud detection in electrical energy consumers using rough sets." in Systems, Man and Cybernetics, 2004 IEEE International Conference on., (The Hague, Netherlands, 2004), IEEE, Vol. 4.

18. Monedero, I., Leon, C., Biscarri, J., and Millan, R. "Midas: Detection of non-technical losses in electrical consumption using neural networks and statistical techniques." in International Conference on Computational Science and Its Applications. Springer, Berlin, 2006.

19. Monedero, I., Leon, C., Biscarri, J., and Millan, R., Guerrero, J., I. "Detection of frauds and other non-technical losses in a power utility using Pearson coefficient, Bayesian networks and decision trees.". International Journal of Electrical Power & Energy Systems, 34(1), 90-98.

20. Gontijo, E. M., Filho, J. R., Delaiba, A. C., Cabral, J. E., Pinto, J. O. P. "Fraud identification in electricity company customers using decision tree." In Systems, Man and Cybernetics, 2004 IEEE International Conference on., (The Hague, Netherlands, 2004), IEEE, Vol. 4.

21. Nizar, A. H., Dong, Z. Y., Zhao, J. H., Zhang, P. "A data mining based NTL analysis method." in Power Engineering Society General Meeting, (Tampa, FL, USA, 2007), IEEE

22. Galvan, J. R., Elices, A., Munoz, A., Czernichow, T., Sanz-Bobi, M. A. "System for detection of abnormalities and fraud in customer consumption." In Proc. of the 12th Conference on the Electric Power Supply Industry, (Pattaya, Thailand ,1998)

23. Costa, B. C., Alberto, B. L. A., Portela, A. M., Maduro, W., Eler, E. O. "Fraud detection in electric power distribution networks using an ANN-based knowledge-discovery process." International Journal of Artificial Intelligence & Applications, 4(6). 17

24. Nizar, A. H., Dong, Z. Y., and Zhao, J. H., "Load profiling and data mining techniques in electricity deregulated market." In Power Engineering Society General Meeting, (Montreal, Que., Canada, 2006), IEEE

25. Nizar, A. H., Dong, Z. Y., Jalaluddin, M., Raffles, M. J., "Load profiling method in detecting non-technical loss activities in a power utility." in Power and Energy Conference PECon'06. IEEE International, (Putra Jaya, Malaysia, 2006). IEEE

26. Angelos, E. W. S., Saavedra, O. R., Carmona Cortes, O. A., de Souza, A. N. "Detection and identification of abnormalities in customer

consumptions in power distribution systems." IEEE Transactions on Power Delivery, 26(4). 2436-2442.

27. Nizar, A. H., Dong, Z. Y. and Zhang, P. "Detection rules for non-technical losses analysis in power utilities." in Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, (Pittsburgh, PA, USA , 2008), IEEE

28. Júnior, L. A. P., Ramos, C. C. O., Rodrigues, D., Pereira, D. R., de Souza, A. N., da Costa, K. A. P., Papa, J. P. "Unsupervised non-technical losses identification through optimum-path forest." Electric Power Systems Research, 140. 413-423.

29. Nagi, J., Yap, K. S., Tiong, S. K., Ahmed, S. K., Mohamad, M. "Nontechnical loss detection for metered customers in power utility using support vector machines.". IEEE transactions on Power Delivery, 25(2). 1162-1171.

30. Jiang, R., Tagaris, H., Lachsz, A., Jeffrey, M. "Wavelet based feature extraction and multiple classifiers for electricity fraud detection." in Transmission and Distribution Conference and Exhibition 2002: Asia Pacific. IEEE/PES, (Yokohama, Japan, Japan, 2002), Vol. 3.

31. Gerbec, D., Gasperic, S., Smon, I., Gubina, F. "Allocation of the load profiles to consumers using probabilistic neural networks." IEEE Transactions on Power Systems, 20(2). 548-555.

32. Allera, S. V., and Horsburgh, A. G. "Load profiling for the energy trading and settlements in the UK electricity markets." In Proc. DistribuTECH Europe DA/DSM Conference, (London, UK, 1998)

33. Méffe, A., Oliveira, C. C. B., Kagan N., Jonathan, S., Caparroz, S., Cavaretti, J. L. "A new method for the computation of technical losses in electrical power distribution systems." in Electricity Distribution, (Amsterdam, Netherlands, 2001), IET

34. Coursera. (2018). *Machine Learning | Coursera*. [online] Available at: https://www.coursera.org/learn/machine-learning.

35. Glauner, P., Meira, J. A., Valtchev, P., State, R., Bettinger, F. "The challenge of non-technical loss detection using artificial intelligence: A survey.", International Journal of Computational Intelligence Systems (IJCIS), 10(1). 760-775.

36. Nagi, J., Yap. K. S., Tiong, S. K., Ahmed, S. K., Nagi, F. "Improving SVM-based nontechnical loss detection in power utility using the fuzzy inference system." IEEE Transactions on power delivery, 26(2). 1284-1285.

37. Nagi, J., Yap. K. S., Tiong, S. K., Ahmed, S. K., Mohammad, A. M. "Detection of abnormalities and electricity theft using genetic support vector machines." in TENCON 2008-2008 IEEE Region 10 Conference, (Hyderabad, India, 2008), IEEE

38. Depuru, S. S. S. R., Wnag, L., Devabhaktuni, V., Green, R. C. "High performance computing for detection of electricity theft." International Journal of Electrical Power & Energy Systems, 47. 21-30.

39. Muniz, C., Figueiredo, K., Vellasco, M., Chavez, G., Pacheco, M. "Irregularity detection on low tension electric installations by neural network ensembles." in Neural Networks, 2009. IJCNN 2009. International Joint Conference on, (Atlanta, GA, USA, 2009), IEEE

40. Ford, V., Siraj, A., and Eberle, W. "Smart grid energy fraud detection using artificial neural networks." In Computational Intelligence Applications in Smart Grid (CIASG), 2014 IEEE Symposium on, (Orlando, FL, USA, 2014), IEEE

41. Ramos, C. C. O., Souza, A. N., Papa, J. P., Falcao, A. X. "Fast non-technical losses identification through optimum-path forest." in Intelligent System Applications to Power Systems, 2009. ISAP'09. 15th International Conference on, (Curitiba, Brazil, 2009), IEEE

42. Glauner, O. P., Boechat, A., Dolberg, L., State, R., Bettinger, F., Rangoni, Y., Duarte, D. "Large-scale detection of non-technical losses in

imbalanced data sets." In Innovative Smart Grid Technologies Conference (ISGT 2016), (Luxembourg, Luxembourg, 2016), Choice Holdings.

43. Chaturvedi, K. (2017). The Characteristics Of Big Data Platforms And Their Importance For Data Exploration | *Linkedin*. [online] Linkedin. Available at: https://www.linkedin.com/pulse/characteristics-big-data-platforms-importance-kamal-chaturvedi/

44. Scanlon, S. (2018). *Big Data*. [online] Available at: http://community.mis.temple.edu/sscanlon/big-data/

45. Blog.sqlauthority.com. (2018). [online] Available at: https://blog.sqlauthority.com/2013/10/02/big-data-what-is-big-data-3-vs-of-big-data-volume-velocity-and-variety-day-2-of-21/

46. M-Brain Market & Media Intelligence Solutions. (2018). *Big Data Technology with 8 V´s - M-Brain Market & Media Intelligence Solutions*. [online] Available at: https://www.m-brain.com/home/technology/big-data-with-8-vs/

47. It.wikipedia.org. (2018). *Point of Delivery*. [online] Available at: https://it.wikipedia.org/wiki/Point_of_Delivery

48. Servizioelettriconazionale.it. (2018). Tariffe usi diversi abitazione | Servizio Elettrico Nazionale. [online] Available at: https://www.servizioelettriconazionale.it/it-IT/tariffe/altri-usi

49. Nuovi contatori elettronici LENNT ®Landis+Gyr MANUALE D'USO (2018). [online] Available at: http://www.amaie.it/impianto%20elettrico/ManualeCE_V2.pdf

50. Zeuslog.com. (2018). Energia Attiva, Reattiva e Fattore di Potenza – ZeusLog. [online] Available at: http://www.zeuslog.com/?page_id=68&lang=it

51. Methods, D. (2018). *Determining The Optimal Number Of Clusters: 3 Must Know Methods - Articles - STHDA*. [online] Sthda.com. Available at: http://www.sthda.com/english/articles/29-cluster-validation-essentials/96-

determining-the-optimal-number-of-clusters-3-must-know-methods/#elbow-method

52. Unyelioglu, K. (2018). *Data Clustering Using Apache Spark - DZone Big Data*. [online] dzone.com. Available at: https://dzone.com/articles/cluster-analysis-using-apache-spark-exploring-colo