University of Zagreb
Faculty of Electrical Engineering and Computing

MASTER'S THESIS no. 2249

Exploratory and Data Mining Analysis in Big Data Environment

Tin Ivan Križ

Zagreb, April 2021

I am extremely grateful to my mentor Damir Pintar for his practical suggestions and helpful advice, and the Faculty of Electrical Engineering and Computing for providing the necessary infrastructure and support. I would like to extend my sincere thanks to my family for their unwavering emotional support and patience.

CONTENTS

1. Introduction

As the amount of generated data in the world is rising, a term *big data* gained popularity as a description of large and diverse datasets that are hard to process. Big data has forced data processing to take a step in a new direction - distributed computing. One of the fastest-growing and most used platforms for distributed data processing at the time of writing is Apache Spark.

This paper demonstrates a workflow of big data processing using Apache Spark in R programming language on a dataset of New York City taxi rides from January 2009 to June 2016. The goal of the analysis is to predict the given tip percentage in a given taxi ride as this research may be useful for taxi companies, taxi drivers and the general study of the tipping system which is an important part of the United States' economy.

The research considers twenty-three taxi ride's features that are related to the ride's location, weather conditions, or are collected by the taximeter. Moreover, the research includes training of linear regression, random forest and decision trees models to predict the tip percentage, as well as a feature importance analysis to determine which features are the most important in predicting tips.

The paper is structured through five more chapters that should extend the understanding of big data analysis. In the second chapter, the growth of data in the world is described with an emphasis on big data and its description and value. It is followed by a chapter that describes the R language and its integration with Apache Spark's infrastructure that can be used to process big data. In other words, the chapter includes an overview of the Apache Spark's architecture as part of the Hadoop ecosystem, and the rationale for using the R language to analyze data. In the fourth chapter, the cross-industry standard process for data mining is described as a popular and effective data mining methodology. Whereas second, third and fourth chapters are mainly theoretical, the fifth chapter presents a practical example of a big data exploratory and data mining analysis using the knowledge from the previous chapters. The analysis uses a dataset of over 1.3 billion NYC taxi rides to predict and understand the received tip ratio. Finally, the last chapter offers a conclusion to the paper and possible future work.

2. Big data

2.1. Data growth

With the rise of technology over the last hundred years, there has also been a rise of generated data. Today, the numbers speak for themselves:

- 4.3 billion people use the internet [1]
- 2.7 billion people use Facebook on a monthly basis [2]
- 2.7 billion people own a smartphone [3]
- 26 billion IoT devices are connected to the internet [4].

Moreover, in one minute:

- 50,000 pictures are posted on Instagram [5]
- 4.5 million YouTube videos are viewed [5]
- 475,000 tweets are posted [5]
- 510,000 comments, 293,000 posts and 136,000 pictures are uploaded to Facebook [2]
- around a quadrillion of bytes[1] of data is generated on average.

The result of this exponential growth of (see Figure 2.1) is the fact that 90% of all of today's global datasphere was generated in the last 2 years [5]. In other words, we are living in the era of data in which the term *big data* stands out.

2.2. What is *big data*?

Big data doesn't have a formal definition and is used in different contexts. The original term is related to big and complicated datasets which are so big that they can not be stored and processed using traditional approaches [7].

[1] A quadrillion bytes is equal to 10^{15} bytes or around 630 terabytes

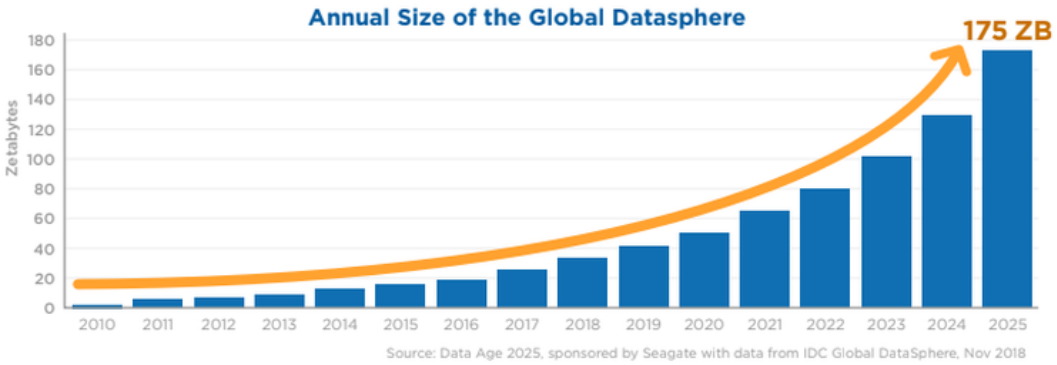

Figure 2.1: Expected growth of the Global Datasphere [6].

The term is most often described using five of its most important properties:

1. Volume - the size of data
2. Velocity - the frequency of arrival of data and the ability to process it in real-time
3. Variety - the different formats and types of data (video, audio, text, pictures...)
4. Veracity - the trustworthiness of data
5. Value - ability to monetize the knowledge out of the data

All five properties are equally important when looking into big data, but, ultimately, one is rising above others - value. Can the data be used to produce value?

2.3. The potential of *big data*

It is often said that "knowledge is power"[2] because the more one knows, the more one will be able to control events. Today, knowledge lies in data.

Growth of data is not only present in Facebook or Google, it is present in all sectors of life such as health, science, retail, politics, insurance, agriculture, etc. Every business wants to advance and grow, and data can help with this task. In other words, the knowledge out of big data may help businesses to reduce costs, increase income, offer new services or products, understand the market better, etc. It is estimated that today's worth of the whole big data market is equal to around 56 billion American dollars (see Figure 2.2).

[2] "Scientia potentia est" is a Latin aphorism which is thought to be first used by Sir Francis Bacon in the 17th century

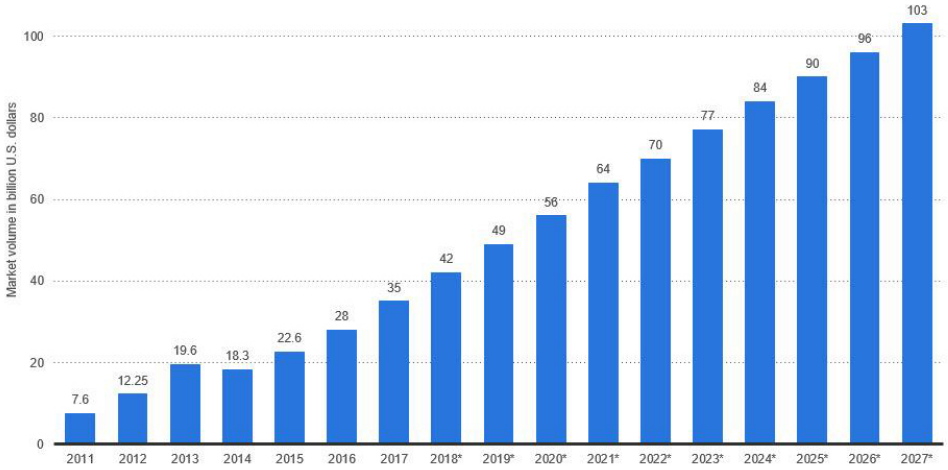

Figure 2.2: Expected worth of the big data market in billions of American dollars ($) [8].

Finally, the question arises: how can we manipulate big data as fast and efficiently as possible? One of the solutions lies in distributed storage and processing which was first offered by the Hadoop ecosystem in 2006 and was afterward used as a foundation to build an even faster and more powerful technology - Apache Spark.

3. Apache Spark and its R integration

3.1. Hadoop ecosystem

Hadoop is an open-source framework that enables the storage and processing of big data in distributed environments. The term *ecosystem* is used because it is comprised out of a large number of technologies which are relying on each other. Figure 3.1 shows a graphic overview of the most popular technologies in the ecosystem and how they relate to each other. All technologies from the ecosystem are a part of the Apache Software Foundation and are open-source which allows them to be rapidly developed and upgraded.

The ecosystem could roughly be divided into the following levels of technologies according to their role:

1. Distributed file systems - data storage
2. Resource managers - mediator between data storage and processing
3. Systems for data processing
4. Peripheral technologies - provide with additional functionalities on top of the existing core systems

Since the focus of this paper is on Apache Spark and not the Hadoop ecosystem itself, only the most popular representatives from each of the abovementioned levels of the ecosystem will be explained.

3.1.1. Hadoop Distributed File System

Distributed file systems

When there is too much data to store it on a single physical computer, it becomes necessary to partition it to multiple machines. Systems that manage data storage in distributed environments are called distributed file systems.

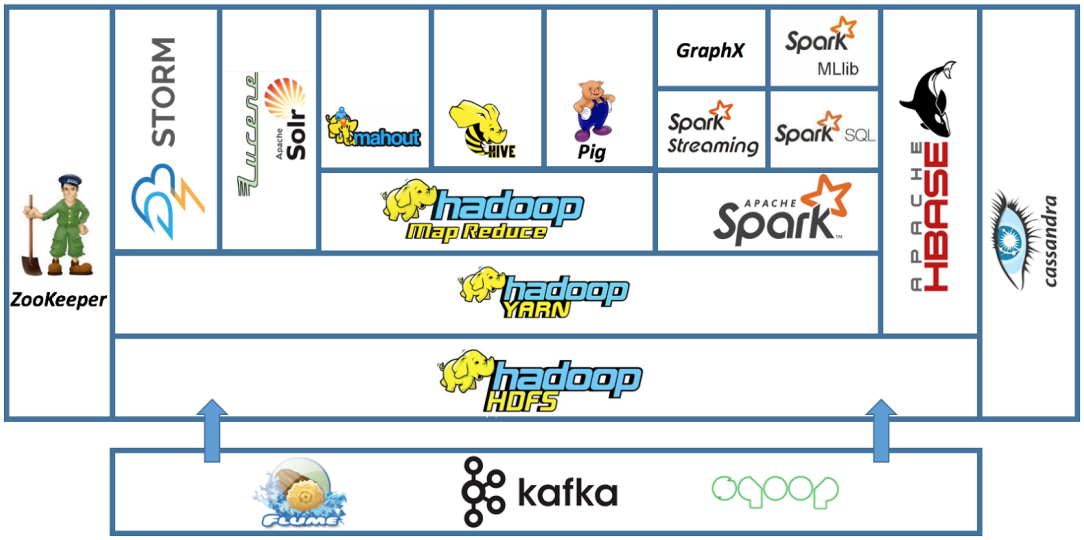

Figure 3.1: Overview of the Hadoop ecosystem [9].

Their task is to enable the client[1] the same management of data as if it was stored on a single machine. Following terms are often used in the context of distributed environments and should be familiar to understand the rest of the chapter:

- Node - A single machine in a network of a distributed file system
- Rack - A group of machines sharing the same power source and network switch
- Cluster - A network of nodes.

Distributed access to big data has a few important conditions:

1. User mobility - possibility for the client to access data from a distant computer[2]
2. Parallel access - possibility to retrieve or process different parts of the same dataset simultaneously
3. Fault tolerance - insurance that there will be no data loss in case of failure of any of the nodes in the cluster
4. Scalability - ability to upgrade the cluster by either adding new nodes (horizontal scalability) or increasing the computational power of existent nodes (vertical scalability)

A distributed file system is representing the first level above the data itself which means that each technology of the higher levels of the ecosystem has to communicate

[1] client can be either the user or an app of a higher level in the ecosystem (see Figure 3.1)
[2] distant computer is a computer which does not have any of the data stored locally

with it. The main task of this level is to store and allow access to the data, while the higher levels use it perform processing. The main technology for this role in the Hadoop ecosystem is the Hadoop Distributed File System (HDFS).

HDFS is adapted for *write once, read many times* pattern which means that it is optimized for fast reading and processing of already written data, but not for changing or retrieving a specific part of the data. HDFS is supported for clusters counting more than a hundred of connected nodes.

Architecture

The architecture of HDFS is based on two types of nodes - named and data nodes.

In a cluster, there exists only one named node and its role is to supervise and control all the other nodes in the cluster and give permissions to clients to access data on the data nodes. Moreover, the named node has access to metadata like the network structure, information on the data nodes, and the log of data changes. However, it does not have access to the data itself. Named nodes are usually machines with better specifications and higher reliability because of their higher importance in the cluster.

Data nodes contain the actual data and execute actions by orders of the named node. They are periodically sending status reports and heartbeats to the named node. It is not necessary for the data nodes to be of high quality, as they usually have average specifications and reliability.

When the client wishes to manipulate data, it first has to communicate with the named node to get access. Afterward, the client is redirected to the corresponding data node which contains the data needed (see Figure 3.2). The data node then communicates directly to the authorized client.

Data blocks

HDFS breaks the data into blocks which it stores on different data nodes in the cluster. The blocks are independent units of data and usually contain 128 MB of data, but this size is configurable depending on the needs of the client.

Breaking the data into blocks fulfills two of the starting conditions for distributed systems: fault tolerance and parallel access. Parallel access is enabled because the blocks are positioned on different nodes which can process data in parallel. On the other hand, fault tolerance is possible because the HDFS copies each block of data on multiple nodes. Named node will always try to store copies of blocks on different racks so that the probability of data loss is the lowest. Additionally, it allows the named node

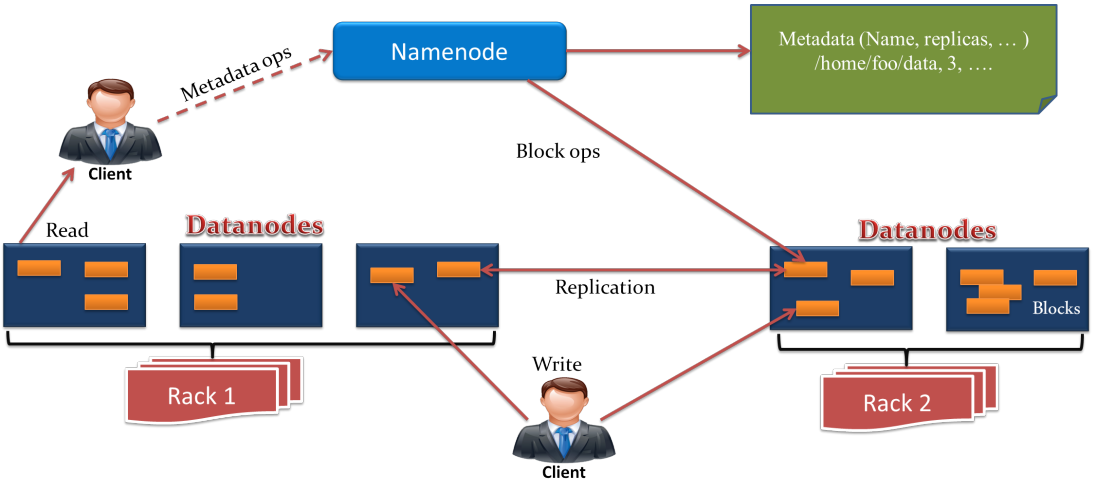

Figure 3.2: Overview of HDFS architecture and how clients can access data [10].

to take the load into consideration by choosing from which data node to give access to data from.

3.1.2. YARN

The resource manager is a mediator between the distributed file systems and the data processing systems (see Figure 3.3). It receives the data manipulation job from the higher levels and then distributes it over the cluster. Separating the data storage and processing in such a manner allows different technologies of the higher levels to not depend on the technologies of the lower level. The level of resource management of the ecosystem is not particularly interesting for the end-users because it is mostly going on in the background and there is little need of changing or manipulating it. Most famous resource managers are YARN and Mesos.

Apache YARN is the most important and most used system for resource management. Although the user will rarely write any code for YARN, the code will be written for technologies of the higher level which will then use the YARN API implicitly. Consequently, this paper will not go any deeper into the architecture of YARN.

YARN enables:

- high compatibility towards the technologies for data processing
- dynamic allocation of resources for more efficient usage
- control of different computational resources such as RAM, CPU, GPU, etc.
- scalability to up to 1000 nodes.

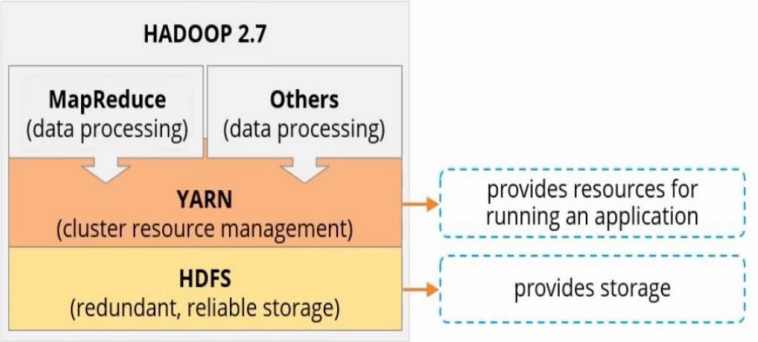

Figure 3.3: Position of YARN in the Hadoop ecosystem [11].

3.1.3. Hadoop MapReduce

In the last two sections, the systems for data storage and resource management have been described, but the most interesting action to perform on data is its processing because it creates the value from data. In the Hadoop ecosystem, there are multiple technologies for processing data but two stand out - MapReduce and Spark.

MapReduce is a simple open-source framework for distributed data processing. It is written in the Java programming language, but it supports any other programming language. The emphasis of the system is on distributed execution and simplicity and it is applicable only for batch[3] data processing in a distributed environment. It is named after its two main components - Map and Reduce.

Model

Model is composed through four main phases (see Figure 3.4):

1. Splitting - the entry data is split into independent parts called splits
2. Mapping - each of the parts is broken down into (key, value) pairs
3. Shuffling and sorting - intermediate results are reorganized for optimization
4. Storing of results

In the first phase, the large input file is split into smaller parts which can be independently processed. The part's size is mostly equal to the size of the block of the distributed files system underneath (128 MB for HDFS), but it can be additionally configured to a custom value. Choosing the size of the split to be the same as the size of the block is the most obvious option because it enables each split to be stored on a

[3]Batch processing is processing static data written on disk.

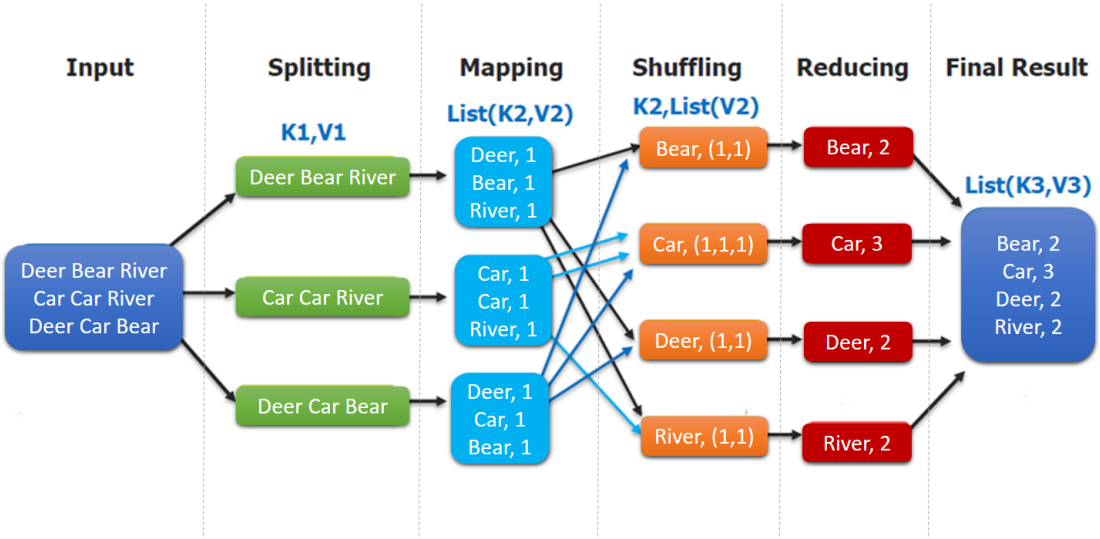

Figure 3.4: The MapReduce model [12].

single node in the cluster, which, consequently, speeds up the processing. Otherwise, the processing of a larger block would have to wait for the rest of the data from the logical split from the other nodes which takes a lot of time.

In the second phase, the user-defined *map()* method is applied to each of the pieces of data. It breaks the data into a list of pairs (key, value) such that they make sense given the task of analysis. E.g. if we want to find out the number of occurrences of words in a text, then the result of this phase would be a list of pairs (word, 1) for each word found in the text. The mapping results are stored on the local disk as temporary intermediate results on the nodes on which the work was performed.

In the shuffling and sorting phase, the mapping results are moved with respect to the keys according to the aggregation jobs. An additional optimization is possible here. Depending on the analysis task, it is possible to sort intermediate results by key or to collect all the intermediate results of the same key in a list.

Furthermore, the edited intermediate results are summarized in the final results of the analysis using the user-defined *reduce()*. These are most often aggregation functions such as sum, minimum, maximum, average, etc. In the end, the final results from each of the aggregation jobs are compiled into a final analysis result file that is written to HDFS.

Note that this way of data processing enables easy parallelization of the work, which is exactly the goal of the model. Although simple, the model proved to be good enough for a wide range of data analysis and processing problems.

Implementation

To fully define the data processing job, the programmer must select the size of the logical block of data and implement the classes *Mapper* and *Reducer*. Moreover, it is possible to also implement the *shuffle and sort* phase through classes of type *Combiner*. In the background, Hadoop will try to organize mapping and reduction jobs so that each job has all the data on the same computer on which it was run.

Hadoop MapReduce has been used to process large amounts of data everywhere and without competition from its inception (2006) until the launch of the next major technology in the world of data processing - Apache Spark (2013).

3.2. Apache Spark

Apache Spark is a unified computing engine and a set of libraries for parallel data processing on computer clusters that was originally created on top of Hadoop. At the time of this writing, it is the most actively developed open-source engine for processing big data. It supports four programming languages: Python, Java, Scala and R. Moreover, it includes libraries for diverse tasks such as SQL querying, streaming, machine learning, and it could run on both a single medium-range laptop and a cluster of thousands of servers.

3.2.1. History

Apache Spark began at UC Berkeley in 2009 as the Spark research project. At the time, Hadoop MapReduce was the dominant parallel programming engine for clusters, being the first open-source system to tackle data-parallel processing on clusters of thousands of nodes. The creators had worked with multiple early Hadoop MapReduce users to understand the benefits and drawbacks of this new programming model, and were, therefore, able to synthesize a list of problems across several use cases and begin designing a more general computing platform. Moreover, the authors had focused on understanding how the Hadoop platform was used for large-scale machine learning using iterative algorithms that need to make multiple passes over data.

Throughout their research, they discovered two things. First, cluster computing has incredible potential. At every organization that used Hadoop MapReduce, brand new applications could be built using the existing data. Second, however, the MapReduce engine was challenging and inefficient for building large applications. The most obvious fallback was the usage in machine learning algorithms where there is a need to pass

through the same dataset ten or twenty times. In MapReduce, each pass would have to be written as a separate job, which would be launched independently on the cluster and would load the data from scratch. Consequently, the Spark team first decided to address this problem by creating an API over a new engine that could perform efficient and in-memory data sharing across computation phases.

Although the first version of Spark supported only batch processing, another interesting use case became clear - real-time processing. The result came in 2011 by simply plugging the Scala interpreter into Spark, it provided a highly usable interactive system for running queries on hundreds of machines. The upgrade was named Shark and has shown that one of the most powerful aspects of Spark will be his additional libraries. Therefore, the project began to follow the *standard library* approach which it still uses to this day.

The project has been contributed to the Apache Software Foundation in 2013 making it long-term and vendor-independent. The project had already been used in more than 30 organizations around the world and had more than 100 contributors at the time. The Apache Spark community released version 1.0 in 2014, 2.0 in 2016, 3.0 in 2019, and has been continuing to make regular releases of the project.

3.2.2. Spark's philosophy

Spark's philosophy is best described through its three components in relation to its definition - Spark is a *unified computing engine and set of libraries for big data* [13].

Unified

Spark is designed to support a wide range of data processing tasks, ranging from simple data loading and SQL queries to machine learning and streaming computation, over the same computation engine. The challenge lies in the fact that these and many more real-world data analytics tasks combine many different processing types and libraries.

Spark's unified nature makes these tasks both easier and more efficient to write. Sparks APIs are composable, consistent, and open-source which allows building any kind of analysis needed. Moreover, the APIs are designed to enable high performance by optimizing across different libraries and functions composed together in a user program.

Computing engine

Although Spark strives for unification, it carefully limits itself to be a computing engine. In other words, Spark handles loading data and its processing, but not the storage itself. It is possible to use Spark with many different distributed file systems such as HDFS (see subsection 3.1.1), Apache Cassandra, Apache Kafka, or even cloud storage systems such as Azure Storage and Amazon S3. The end-user of Spark mostly does not need to know about the underlying storage system as Spark is trying hard to make all these storage systems look similar on the outside. This versatility to data storage systems allows Spark to process data stored anywhere, in difference to Hadoop MapReduce which is more dependent to HDFS.

Libraries

The final part of Spark is its libraries. It supports both internal libraries that are included with the engine and external libraries that are published as third-party packages by the open-source community.

Today, the Apache Sparks core is not changing nearly as much as the libraries growing around it. Spark includes libraries for SQL and structured data (Spark SQL), machine learning (MLlib), stream processing (Spark Streaming), and graph analytics (GraphX). There are hundreds of additional libraries ranging from connectors for various underlying storage systems to new state-of-the-art machine learning algorithms.

3.2.3. Basic architecture

Spark's architecture is composed of three main parts (see Figure 3.5): a driver node, worker nodes and the cluster manager. The cluster manager is exactly the same as the resource manager that was detailed in Subsection 3.3. The worker nodes (also referred to as executors) execute the compute tasks over partitioned data and communicate the intermediate results to other workers or back to the driver node. The driver node is running the actual *main* function and is maintaining information about the Spark application, responding to the user's input and analyzing, distributing and scheduling work across the worker nodes. Usually, the driver node is just one and has much more compute resources than the worker nodes.

However, while the worker and driver nodes are simply just machines, the actual computation is performed by the Spark context. Spark context is the main entry point for Spark functionality since it is tasked with scheduling tasks, managing stor-

age, tracking execution status, specifying access configuration settings, canceling jobs, etc. In the worker nodes, the actual computation is done in a spark executor which is a Spark component tasked with executing subtasks against a specific data partition.

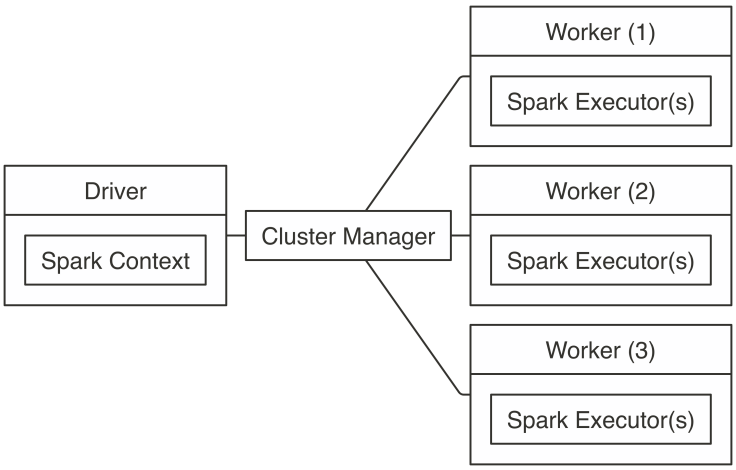

Figure 3.5: Basic Spark architecture [14].

3.3. The R interface for Apache Spark

3.3.1. About the R language

R is a language and environment for statistical computing and graphics. According to the kaggle's annual survey [15], R is the seconds most popular language for data analysis on that data science website, after Python.

R prides itself to be designed by statisticians for statisticians, meaning that this is one of the few programming languages designed for non-programmers, resulting in a more natural programming feeling. Moreover, R allows the data scientist to focus more on understanding the data and less on the particulars of computer science and engineering.

The second biggest advantage of the R programming language is its large community. The R community is active, welcoming and always willing to help new users succeed. It provides its users with a rich package archive named Comprehensive R Archive Network (CRAN) which allows the installation of ready-to-use packages for many tasks such as high-quality data manipulation, visualization and statistical models, many of which are only available in R. Some of the most downloaded and most relevant packages for the topic of this paper are *dplyr* to manipulate data, *ggplot2* to

visualize data and *sparklyr* to provide an R interface to Spark.

3.3.2. Package *sparkly*

Package *sparklyr* is an R package available through CRAN that enables its users with an R interface to Apache Spark. In other words, it combines the R's ease of use and Spark's computation power into a powerful analytic tool for big data. It works like any other CRAN package, meaning that it is easy to install (see snippet 3.1) and it is compatible with other packages and practices from the R community.

Code snippet 3.1: Loading the *sparklyr* package in R

```
install.packages("sparklyr")
library(sparklyr)
```

Rather than being created by the Spark community, the package was created within the R community by the programmers from RStudio (the popular R IDE) which means that it is created by R users for R users. In other words, the package provides an R-first approach to Spark by hiding the common underlying Spark concepts with its own simplified ideas. These Spark concepts are commonly unfamiliar and potentially irrelevant to a typical R user and are, therefore, allowing the user to completely focus on analyzing data and less on the Spark's infrastructure. For instance, code snippet 3.2 shows how simple it is to connect to a local Spark cluster using *sparklyr* and to load a CSV file into a Spark DataFrame from an underlying HDFS.

Code snippet 3.2: Connecting to Spark using *sparklyr*

```
sc <- spark_connect(master = "local")
data <- spark_read_csv(sc,
    name = "data_name",
    path = "path/to/data.csv"
)
```

Except for CSV, only JSON and Parquet formats are supported.

Moreover, *sparklyr* is based on the popular *dplyr* package for manipulating structured data which enables the users to use the same code they once used on a local R's *data.frame* object to now use on a distributed Spark DataFrame. In short, scaling up requires no code changes. However, this simplification comes at a price. Some of Spark's built-in functions and APIs may not be accessible unless dplyr also supports

them. For instance, there is a limited number of functions available for data aggregations using the *dplyr*'s *summarise* function such as sum, min, max, mean, count, etc. If the user needs some other aggregation, such as count of missing values or manipulation of timestamp data, he could write it himself using a custom function.

Another alternative is to execute arbitrary SQL code against the cluster using the *DBI* library which provides a lower-level SQL interface to Spark (see snippet 3.3).

Code snippet 3.3: Executing SQL queries against the cluster

```
library(DBI)
allTables <- dbGetQuery(sc, "SHOW TABLES")
```

Furthermore, *sparklyr* supports some of the core machine learning algorithms for clustering, classification and regression. Some of the most relevant models will be later discussed in more detail in Subsection 5.4.4.

3.3.3. RStudio with Spark

RStudio is the most popular integrated development environment (IDE) for R programming language. Not only does it offer great support for day-to-day R programmer's needs but it also supplies with additional functionalities for integration with Apache Spark. For instance, it supports connecting to the cluster from a user interface (UI), an overview of data frames loaded into yarn, quick access to Spark's and YARN's web UI, etc. Moreover, it is possible to install RStudio on a cluster to enable clients to connect using web browsers and communicate to the cluster using HTTP protocol as is shown in Figure 3.6.

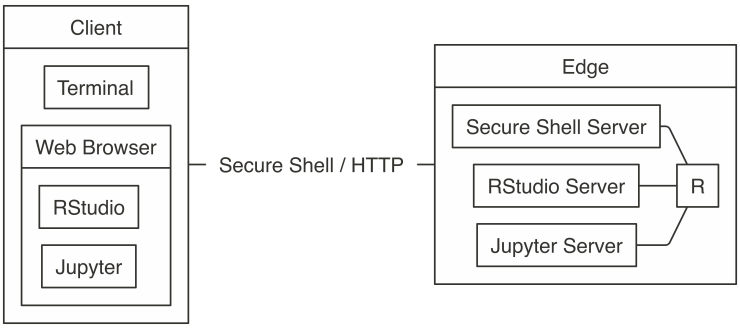

Figure 3.6: Connecting to Spark using a web browser [14].

4. Data mining methodology

This paper will be using the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology. In this chapter, the general methodology will be described at a basic level. At the time of writing, CRISP-DM is the most widely used methodology in the world of data mining [16]. It is a robust and well-proven methodology that provides a structured approach to planning a data mining project.

Phases of the model are shown in Figure 4.1 and are:

1. Business understanding
2. Data understanding
3. Data preparation
4. Modeling
5. Evaluation
6. Deployment

The model proposes an idealized sequence of events. In practice, many of the tasks can be performed in a different order and it will often be necessary to backtrack to previous tasks and repeat certain actions. The model does not try to capture all possible routes through the data mining process.

4.1. Business understanding

The first stage of the CRISP-DM methodology is to get familiar with the domain of the analysis, to set business or scientific goals and a basic plan of achieving those goals. Neglecting this step can mean that a great deal of effort is put into producing the right answers to the wrong questions. It includes:

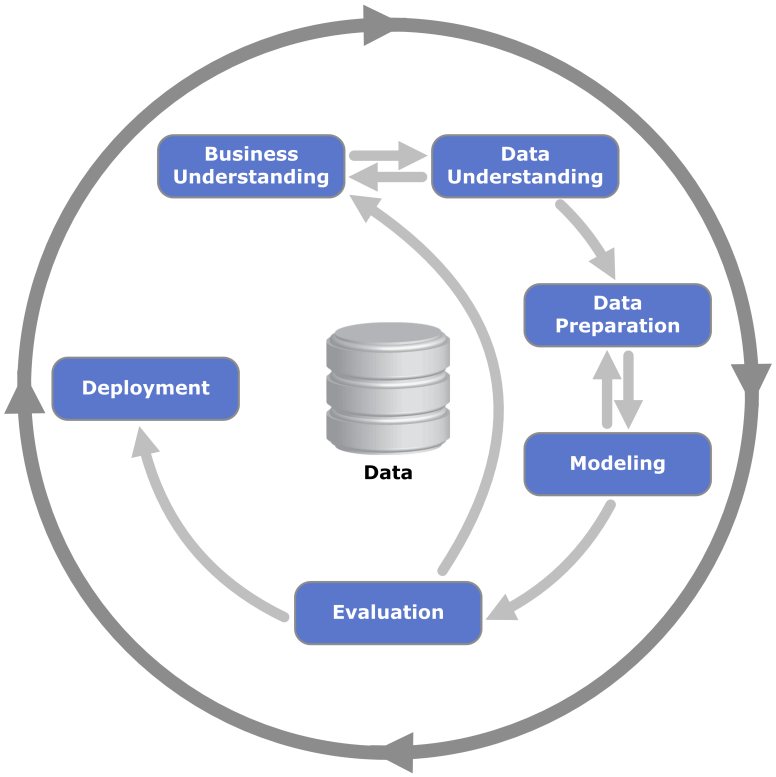

Figure 4.1: CRISP-DM model [17].

1. Domain research - Getting familiar with the basic concepts and the vocabulary of the subject of analysis or talking with an expert in the field.

2. Resources inventory - A list of the resources available to the project such as other personnel, relevant datasets, hardware and software resources.

3. Requirements, assumptions and constraints collection - A list of all requirements of the project including the schedule of completion, the required comprehensibility and quality of results, and any data security concerns as well as any legal issues.

4. Risk evaluation - A list of the risks or events that might delay the project or cause it to fail, and a list of the corresponding contingency plans - what action could be taken if these risks or events take place?

5. Costs and benefits collection - A cost-benefit analysis for the project which compares the costs of the project with the potential benefits to the business if it is successful.

6. Success criteria - A description of the intended outputs of the project that enable the achievement of the business or scientific objectives.

7. Project plan - A description of the intended plan for achieving the data mining goals and thereby achieving the business or scientific goals.

4.2. Data understanding

The second stage of the CRISP-DM process requires acquiring, loading and diving into the data listed in the project resources. Data understanding (stage 2) and preparation (stage 3) are what is widely and commonly understood as Exploratory data analysis (EDA). Data understanding includes:

1. Initial data collection report - List of the data sources acquired together with their locations, the methods used to acquire them and any problems encountered.

2. Data description report - Description of the data that has been acquired including its format, its dimensions, the identities of the fields and any other surface features which have been discovered.

3. Data quality report - List of the results of the data quality verification and their possible solutions. Most typically includes a check of missing values and a check of the cardinality of categorical features and univariate analysis of continuous and discrete features.

4. Data wrangling - Mapping of the raw data into a usable format for analysis. May include transformations of feature types, filtering impossible and plausible anomalies, removing useless features and any other actions that would result in a cleaner dataset.

5. Data exploration report - A result of data exploration, including first findings or initial hypothesis and their impact on the remainder of the project. Usually includes examining:
    - univariate distributions of key features
    - relationships between pairs of a small number of attributes
    - results of simple aggregations
    - properties of significant sub-populations
    - simple statistical analyses.

4.3. Data preparation

This is the stage of the project where a learning-ready dataset is created. The criteria used to make decisions include the relevance of the data to the data mining goals, the quality of the data, and also technical constraints such as limits on data volume or data types. The data selection covers a selection of attributes (columns) as well as a selection of records (rows) in a table. The stage includes:

1. Inclusion/exclusion rationale - List of the data to be included/excluded and the reasons for these decisions

2. Feature engineering - Use of domain knowledge to derive new features from raw data. It may include simple operations from existing features or binning continuous into categorical or discrete features.

3. Data integration - Use of multiple datasets relevant to the topic to enrich the primary dataset. Other datasets could be merged from external resources or aggregated from the primary dataset.

4.4. Modeling

Modeling is the stage of the process where all the knowledge on the dataset and the domain from the previous stages should result in a functional and goal-relevant model. It includes:

1. Modeling technique - Documenting the actual modeling technique that is to be used.

2. Test design - Description of the intended plan for training, testing and evaluating the models.

3. Parameter settings - List of all the parameters and their chosen values, along with the rationale for the choice of parameter settings.

4. Model training - Building of the aforementioned models.

5. Model descriptions - Description of the resulting models, report on the interpretation of the models and document any difficulties encountered with their meanings.

6. Model assessment - Summary of the results for this task, list the qualities of the generated models (e.g.in terms of accuracy) and rank their quality in relation to each other.

7. Revised parameter settings - Revision of the parameter settings and tuning them for the next modeling run.

4.5. Evaluation

The fifth stage of the process where the built models are evaluated from a business perspective and all the relevant insights on the problem are listed and summarised. It gives a verdict on the success of the process and proposes actions for further improvement. It includes:

1. Data mining results assessment - Summary of assessment results in terms of business success criteria, including a final statement regarding whether the project already meets the initial business objectives.

2. Process review - Summary of the process review and highlight of the activities that have been missed and those that should be repeated.

3. Future improvements - List of the potential further actions, along with the reasons for and against each option.

4.6. Deployment

The final stage of the process is deploying the model into the business and creating the report of the whole process. This paper will not be deploying the resulting model because that is not its purpose, but it will be reporting on the final results and experience. The stage of deployment generally includes:

1. Deployment plan - Summary of the deployment strategy including the necessary steps and how to perform them.

2. Monitoring and maintenance plan - Summary of the monitoring and maintenance strategy, including the necessary steps and how to perform them

3. Final report - The final written report of the data mining engagement. It includes all of the previous insights, summarizing and organizing the results.

5. A practical example of big data analysis

5.1. Introduction to the analysis

This paper presents an analysis of tips in taxi rides in New York City (NYC) from January 2009 to June 2016 using Apache Spark and its R interface - *sparklyr*.

The dataset, which is described in more detail later (see Subsection 5.3.1), is 32.6 GB in size and represents big data. It is loaded into HDFS on a cluster of eight nodes where each node has two gigabytes of RAM and four virtual cores available for processing. YARN is the resource manager used as a mediator between the HDFS and the Apache Spark. Hadoop version *3.0.0-cdh6.3.1* and Spark version *2.4.0* is used. Moreover, the cluster has pre-installed RStudio IDE which enables simple execution of R scripts on the cluster and connection through a web browser, as was described in Section 3.3.3.

5.2. Use case: Tipping system in the United States

5.2.1. Role of tipping in the United States' economy

A tip (formally called *gratuity*) is a sum of money customarily given by a client or customer to certain service sector workers for the service they have performed, in addition to the basic price of the service. Tipping by definition is voluntary and at the discretion of the customer. It is an area of increasing interest in economics and social sciences and represents a significant portion of the United States' economy.

It is believed that tipping in the U.S. dates back to the American Civil War in 1865 and has become the norm by the end of the nineteenth century [18]. Moreover, tipping is formally accepted as a significant part of the U.S. economy in 1966 when there has been a new amendment on the U.S. Federal Law stating that employers can pay their

employee's minimum wage down to $2.13 per hour if the employee receives at least $30 per month in tips [19]. This law has been accepted in forty-three out of fifty states in the United States (U.S.).

The tipped workers in the U.S. include waiters in bars and restaurants, hotel staff, taxi drivers, barbers, hairdressers, food deliverers, baggage porters, gaming dealers and many other professions that offer personal service.

5.2.2. Main reasons for tipping

According to Micheal Lynn (a researcher of tips), there are five main motivations for tipping in general [20]:

1. Showing off - Giving a larger tip is a sign of wealth and generosity.
2. To supplement the server's income - As mentioned earlier (see Subsection 5.2.1), the servers in the U.S. may legally be paid as low as $2.13 per hour by their employers. Furthermore, tipped employees have three times the poverty rate than the non-tipped ones [21]. As a result, the general public believes that the people working in tipped occupations rely on tips to survive.
3. Getting better service in the future - Customers may want to ensure quality and swiftness for the next time they need the service by giving a larger tip.
4. Avoiding disapproval - Many tipped employees tend to show their disapproval of an insufficient tip which many people would rather just avoid.
5. Sense of duty - Customers often tip simply because everyone is doing it.

One of the most common misconceptions is that tipping serves as a reward for exceptional service, but this theory has been disproved for professions regarding services in bars and restaurants by Lynn and Graves [22] showing that there is no significant correlation between tip amounts and quality of service. On the other hand, there is a significant amount of research showing that customer's and employee's traits are influencing the tip amount such as gender [23], race [24], age [25] and psychological traits such as extraversion [26].

5.2.3. Role of tips for taxi drivers in NYC

Generally speaking, taxi drivers in NYC do not rely on tips as much as other tipping employees because their income is mostly based on basic fare amounts. According

to the official Taxi & Limousine Commission (TLC) Factbook from 2016 [27], the average hourly income of NYC taxi drivers was $30.41 in 2015. By taxi driver's experience [28], half of the money earned is pure profit. Therefore, taxi drivers should earn around $15 of profit per hour on fare amounts only. Tips represent additional pure profit.

5.2.4. Value of the research

The value of this research has two sides: economic and scientific.

On one hand, the research may be of economic value to taxi companies to get a better estimate of the amount of tips their drivers get in cash, as tips are not formally registered for cash-paid rides. Additionally, the models may also be useful for identifying or analyzing anomalies for all types of rides. Moreover, an insight into people's tipping behavior may be profitable for taxi drivers because they could use the knowledge to make a better choice of where or when to drive to get more tips.

Another economical plus side is that the taxi companies could profit by applying the same modeling techniques on their datasets and, thus, get more insight into their customer's tipping habits and their driver's income. In 2019 alone, the revenue of the taxi and limousine services in the United States was 31 billion dollars [29]. With the premise that the customers tipped 15% on average and the fact that these services are customarily tipped in the United States, it would mean that approximately 4.65 billion dollars have been generated in tips. Therefore, if a better understanding of tipping patterns increases the average customers' tip by just 0.1%, it would be worth 4.65 million dollars for the taxi and limousine services in the United States alone.

On the other hand, the research may be useful for the general psychological and economical scientific research of the tipping system which is one of the foundation stones of the United States' economy (see Subsection 5.2.1).

5.3. Exploratory data analysis

5.3.1. Dataset

This paper will be using the official *TLC Trip record* data from January 2009 to June 2016 for yellow and green taxis, but downloaded from [30] in parquet format. The dataset contains information on 1,382,375,998 rides and has 32.6 GB in size in parquet format. All 24 features in the dataset are described in Table 5.1.

Feature name	Feature description	Data type in Spark
dropoff_latitude	Geographical latitude of the dropoff location.	double
dropoff_longitude	Geographical longitude of the dropoff location.	double
pickup_latitude	Geographical latitude of the pickup location.	double
pickup_longitude	Geographical latitude of the pickup location.	double
dropoff_taxizone_id	Identifier of the specific dropoff taxi zone.	integer
pickup_taxizone_id	Identifier of the specific pickup taxi zone.	integer
ehail_fee	Fee for hailing a taxi using an app.	double
extra	Miscellaneous extras and surcharges.	double
fare_amount	The time-and-distance fare calculated by the meter.	double
improvement_surcharge	$0.30 improvement surcharge for funding an increase of wheelchair accessible vehicles that began being levied in 2015.	double
mta_tax	$0.50 MTA tax that is automatically triggered based on the metered rate in use.	double
passenger_count	The number of passengers in the vehicle. This is a driver-entered value.	integer
payment_type	A code signifying how the passenger paid for the trip.	string
rate_code_id	The final rate code in effect at the end of the trip.	string
store_and_fwd_flag	This flag indicates whether the trip record was held in the vehicle's memory before sending to the vendor, aka "store and forward," because the vehicle did not have a connection to the server.	string
tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.	double
tolls_amount	Total amount of all tolls paid in trip.	double
total_amount	The total amount charged to passengers. Does not include cash tips.	double
trip_distance	The elapsed trip distance in miles reported by the taximeter.	double
trip_type	Indicates the type of the taxi trip (yellow, green, uber...).	string
vendor_id	A code indicating the vendor that provided the record.	string
trip_id	Identifier of the trip.	integer
dropoff_datetime	The date and time when the meter was disengaged.	timestamp
pickup_datetime	The date and time when the meter was engaged.	timestamp

Table 5.1: Description of the original raw dataset.

Importing the dataset

Firstly, the dataset has been downloaded from [30] and uploaded to the cluster from the local computer via File Transfer Protocol (FTP). Afterward, it has been loaded into HDFS by executing the following command in the shell:

Code snippet 5.1: Loading the dataset in the HDFS

```
hadoop fs -put /path/to/data/directory
```

Subsequently, the data is loaded into a Spark DataFrame using the *sparklyr* package as was previously described in Subsection 3.3.2.

5.3.2. Data quality & Data wrangling

Missing values

As it has already been mentioned in Section 3.3.2, the package *dplyr* supports a very limited number of functions for data manipulation. Consequently, at the time of writing, the *dplyr* package does not offer a function for quick and simple calculation of missing values. The custom-made function is shown in code snippet 5.2 and the result of applying it on the raw dataset is shown in Table 5.2.

Code snippet 5.2: Custom-made function for calculating missing values

```
sdf_count_na <- function(tbl){
    df <- map_df(set_names(colnames(tbl)), function(x){
        sym_x <- sym(x)
        count <- tbl %>%
            filter(is.na({{sym_x}})) %>%
            sdf_nrow()

        perc = count / sdf_nrow(tbl)

        tibble::tibble(variable = x,
                       count_na = count,
                       perc_na = perc)
    })
    return(df)
}
```

Feature	Count of missing values	Ratio of missing values
dropoff_latitude	87,938,516	6.37%
dropoff_longitude	87,938,516	6.37%
pickup_latitude	83,396,504	6.03%
pickup_longitude	83,396,504	6.03%
dropoff_taxizone_id	48,721,937	3.52%
pickup_taxizone_id	28,460,209	2.06%
ehail_fee	1,382,375,998	100%
extra	18,804,806	1.36%
fare_amount	18,804,806	1.36%
improvement_surcharge	106,9478,672	77.37%
mta_tax	159,845,375	11.56%
passenger_count	1,382,375,998	100%
payment_type	0	0.00%
rate_code_id	1,382,375,998	100%
store_and_fwd_flag	537,641,570	38.89%
tip_amount	18,804,806	1.36%
tolls_amount	18,804,806	1.36%
total_amount	18,804,806	1.36%
trip_distance	18,804,806	1.36%
trip_type	0	0.00%
vendor_id	0	0.00%
trip_id	0	0.00%
dropoff_datetime	0	0.00%
pickup_datetime	0	0.00%

Table 5.2: Missing values in the raw dataset.

As a result of the high number of missing values shown in Table 5.2, features *passenger_count*, *ehail_fee*, *rate_code_id* and *improvement_surcharge* have been removed from the dataset.

Defining the goal feature - tip ratio

The goal of the analysis is to predict the tip ratio defined as the voluntary amount divided by the obligatory amount paid to the driver.

Generally speaking, the obligatory amount of the taxi ride's price can be calculated as the sum of six other charges: the fare amount, improvement surcharge, MTA tax, tolls amount and the extra charges. However, there are 3.5% of the rides in which that is not the case and around 75% of the rides in which at least one of the mentioned features is missing. Therefore, this approach to calculating the obligatory price is not reliable.

On the other hand, the amount listed as the *total_amount* is the amount directly taken from the customer's credit card and is, therefore, a reliable metric. It is composed out of the tip (voluntary part) and the obligatory part of the ride's price. Consequently, in reference to the dataset, the goal feature (tip ratio) is defined as:

$$obligatory\_amount = total\_amount - tip\_amount \tag{5.1}$$

$$tip\_ratio = tip\_amount / obligatory\_amount \tag{5.2}$$

Moreover, the tip ratio has been additionally rounded up to 3 decimal points. For instance, possible values would be 0.150, 0.067 or 0.123 for 15.0%, 6.7% and 12.3% tips. This rounding has been introduced because the tip amount given to the driver is discrete where the smallest increment is 1 cent ($0.01). With this rounding, the tip percentage for a customer that paid a $2.5 tip for a $10 bill and for another customer that paid $1.94 for a $7.77 bill will both have the same labeled tip ratio in our dataset (25.0%) which describes the same intention, even though the exact ratios would not be the same (25% and 24.967%).

Analyzing tip amounts per type of payment showed that tips are only registered for rides paid by credit cards or by rare *unknown* sources in the original dataset (see Table 5.3). Consequently, further analysis will be limited to rides paid by credit cards. This limitation has deduced the dataset from 1,382,375,998 to 659,597,797 rides (47.71% of the original dataset).

In order to further analyze the quality of data, four new features have been extracted from the existing features in the dataset:

Payment type	Count	Average tip amount ($)
Credit Card	659,597,797	2.4709462133
Cash	698,713,510	0.0008309549
Dispute	1,005,882	0.0046593437
Unspecified	18,804,806	Not a value
No Charge	3,141,837	0.0068077911
Unknown	1,112,166	2.1835603650

Table 5.3: Average tips per payment type in the original dataset.

1. Trip duration (in seconds)

$$trip\_duration = dropoff\_datetime - pickup\_datetime$$

2. Average speed (in miles per hour)

$$avg\_speed = (trip\_distance/trip\_duration) * 3600 \tag{5.3}$$

3. Direct distance (in miles) - Represents the geographical distance (commonly known as as-the-crow-flies distance) between the pickup and the dropoff location. The formula used is derived from projecting a spherical Earth to a plane and it is the simplest approximation of the distance [31]:

$$direct\_distance = R\sqrt{(\Delta\phi)^2 + (cos(\phi_m)\Delta\lambda)^2} \tag{5.4}$$

   where R is the radius of the Earth in miles (3958.761) and $\Delta\phi$ and $\Delta\lambda$ represent the difference in latitude and longitude expressed in radians, respectively. ϕ_m is the "mean latitude" and is calculated as $\phi_m = \frac{\phi_1+\phi_2}{2}$.

4. Ineffectiveness - shows how "ineffective" the trip was compared to traveling via the direct aerial line. Additionally, it may indicate rides in which the taxi driver intentionally took longer routes to the desired destination. This feature will also be used for filtering out rides with improbable values (far less than 1 or higher than 10).

$$ineffectiveness = trip\_distance/direct\_distance \tag{5.5}$$

Rides containing missing values for key features listed above (total amount, tip amount, trip distance, dropoff and pickup time, dropoff and pickup location) have been removed from the dataset. This filter has removed additional 59,189,527 rides and has truncated the dataset to 600,408,270 rows.

Filtering rows

Table 5.4 shows that the raw dataset had contained negative values for rides' tip amounts, trip distances and total price. These negative values were the first to be filtered out after the rows containing missing values for the key features mentioned in the previous section. As a result, the dataset has been reduced by another 736 rows.

In the second and final stage of filtering data, rows containing improbable and extreme values of key features have been removed from the dataset. Firstly, the analysis of quantile values (see Table 5.4) shows that there are records with negative trip durations which are a result of sensor malfunctions. Secondly, all rides which seem to be a result of taximeter starting by accident are removed. Those are the rides that had either total amount, trip distance or trip duration equal to zero. Thirdly, the highly improbable values are removed. For instance, rides that have the *ineffectiveness* feature less than one would mean that the driver traveled from pickup to dropoff in a faster route than the aerial distance which is impossible. However, due to the precision of the calculation of the *direct distance* feature (simplest approximation, see 5.3.2) and the precision of the sensors gathering the data, all rides that had the feature's value less than 0.75 were considered highly improbable. Moreover, all rides that were driven by the average speed of more than fifty miles per hour in an urban environment such as NYC have also been removed from the dataset as highly improbable. Table 5.4 shows all the other applied domain limitations on key features that have finally truncated the dataset to 576,229,728 rows.

Filtering	**Quantile** / **Feature**	**0.00 (min)**	**0.01**	**0.25**	**0.50**	**0.75**	**0.99**	**1.00 (max)**
No filter (raw dataset)	tip_amount	-1677720.1	0.0	0.0	0.0	1.9	10.0	3950588.8
	total_amount	-21474830.00	3.80	7.40	10.30	15.30	61.83	3950611.50
= 1,382,375,998 rows	trip_distance	-40840124.00	0.10	1.01	1.75	3.20	18.09	198623008.00
With filtering:	tip_amount	0.00	0.00	1.11	2.00	2.86	11.47	3950588.75
* rides paid by credit card	total_amount	0.0	5.2	8.9	12.0	17.8	67.7	3950611.5
* removing missing values	trip_distance	0.00	0.30	1.18	1.99	3.57	18.30	59016608.00
* 0 <= tip_amount	tip_ratio	0.000	0.000	0.147	0.197	0.200	0.429	173652.253
* 0 <= total_amount	trip_duration	-180533880	118	420	668	1053	3000	44756446
* 0 <= trip_distance	avg_speed	-1183.19996	2.46790	8.55298	11.60083	15.75000	36.59668	3079127373.91304
	direct_distance	0.00000000	0.02280578	0.93771282	1.58740120	2.84285206	14.01972258	30.18191168
= 600,407,534 rows	ineffectiveness	0.0000000	0.5193803	1.1061478	1.2328695	1.3788128	3.5065314	544170711.8685976
With final additional filtering:	tip_amount	0.00	0.00	1.10	1.96	2.77	10.40	250.07
* 0 <total_amount	total_amount	0.01	5.30	8.90	12.00	17.50	62.83	2001.06
* 0 <trip_distance <= 40	trip_distance	0.01	0.40	1.20	2.00	3.51	17.65	40.00
* 0 <trip_duration <= 3000	tip_ratio	0.000	0.000	0.147	0.197	0.200	0.377	0.500
* tip_ratio <= 0.5	trip_duration	1	120	420	668	1036	2520	3000
* 0.75 <= ineffectiveness <= 10	avg_speed	0.012500	3.613636	8.640000	11.650072	15.763637	36.266667	50.000000
* 0 <avg_speed <= 50	direct_distance	0.001054282	0.302758713	0.957199890	1.594586822	2.812685062	13.623567687	30.128790985
= 576,229,728 rows	ineffectiveness	0.7500000	0.8827423	1.1104510	1.2341012	1.3778628	2.8247819	9.9999880

Table 5.4: Quantile values for key features in the dataset in different filtering phases

Data wrangling using *sparklyr*

Most of the abovementioned process was done using the R's *dplyr* package, as if with any other local dataset, with addition of using the *sparklyr*'s function *sdf_quantile* for calculating quantile values shown in Table 5.4. For example, code snippet 5.3 shows how to create a new feature, calculate some of its important quantile values and then filter the dataset by the feature's value. The same principle had been used for all of the features in this section.

Code snippet 5.3: Example of a data wrangling process using *sparklyr* and *dplyr*

```
# Create feature
nyc_taxi_data <- nyc_taxi_data %>%
  dplyr::mutate(
    avg_speed = (trip_distance / trip_duration) * 3600
)

# Explore feature
sdf_quantile(
    nyc_taxi_data,
    column = "avg_speed",
    probabilities = c(0, 0.01, 0.25, 0.50, 0.75, 0.99, 1)
)

# Filter improbable values
nyc_taxi_data <- nyc_taxi_data %>%
  dplyr::filter(
    avg_speed <= 50
)
```

5.3.3. Data exploration and feature engineering

In the previous section (5.3.2), a group of key features used for filtering has been created and described in Table 5.4 which helped to create the final version of the dataset. In this section, the goal is to look into the possible influence of those existing features to the average tip ratio and to add and examine new features to identify those that could be most beneficial for inclusion in the optimal model.

Before further exploration of the data, the dataset was partitioned into the train and

test datasets as will be described in more detail in Section 5.4. All further exploration was done using the training set to reduce research bias.

The first big step in data exploration is bivariate analysis - analyzing the relationship between two features. The approach to the bivariate analysis is changed when analyzing big data. It is generally done using scatter or jitter plots for continuous features, or boxplots for discrete features. However, as these operations are so computationally difficult for datasets as large as the one used in this paper, the bivariate analysis mostly came down to simply analyzing how a change of one feature influences the mean of another. Therefore, the effect of change of independent key features to the tip ratio is examined by examining the change of average tip ratio with the change of the independent feature's value. Furthermore, if the independent feature is continuous, then it is categorized before plotting to decrease the computation time. Moreover, every plot is accompanied by an additional visualization of the univariate distribution categorized independent feature in question. An explanation of how the visualizations have been made using *sparklyr* and *ggplot2* can be found at the end of this section.

The explored features are categorized by their source as:

1. Features from the original dataset
2. Features derived from the original dataset
3. Features integrated from location data
4. Features integrated from weather data.

Features from the original dataset

All the features from the original dataset have been listed and described in Table 5.1. Most of the features can not be directly used for modeling because they either contain too many missing values (improvement surcharge, e-hail fee, rate code id, store and forward flag, passenger count), or they are not useful in raw format (dropoff latitude, dropoff longitude, pickup latitude, pickup longitude, dropoff time, pickup time), or they contain the goal feature (tip amount, total amount), or they have no interpretability (trip id, vendor id). The only features that could be useful in raw format are the taxi ride's fare amount, MTA tax, tolls amount, extra charges, trip distance and trip type.

From these features, the first four are all a part of the obligatory part of the fare which was discussed and added to the dataset in Subsection 5.3.2 and which will be further explored in Subsection 5.3.3. Therefore, only the features *trip_type* and *trip_distance* from the original dataset could be further explored.

Firstly, as is already described in Table 5.1, the *trip_type* refers to whether the ride is taken with a yellow or a green taxi in NYC. Figure 5.1 shows that yellow taxis make around 96.5% of the dataset and get 1.5% higher tips on average. That is expected because yellow taxis are localized for Manhattan costumers and green taxis are supposed to only pick up passengers from the other four boroughs. The difference between the two types of taxis is big enough for including the feature in the future model.

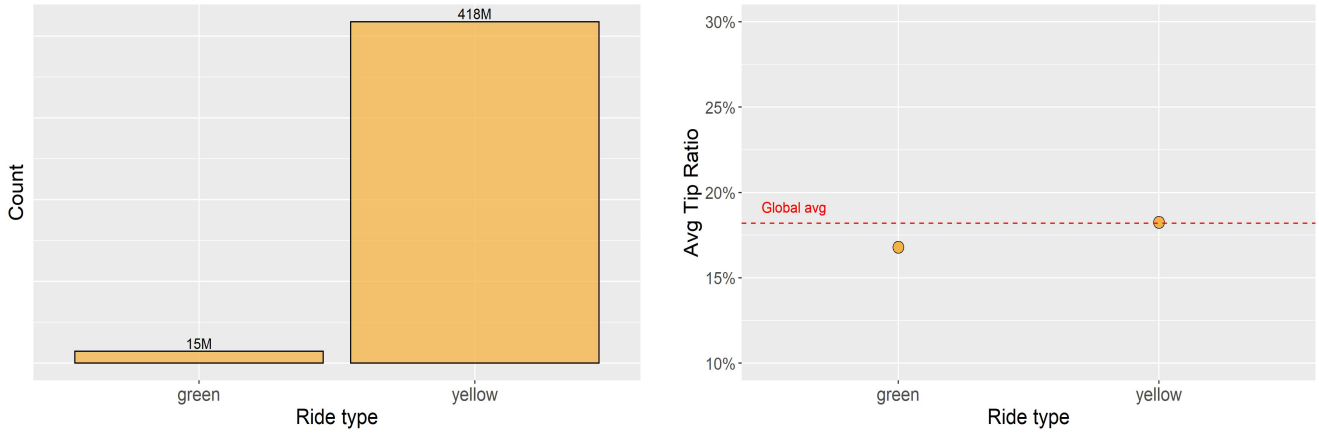

Figure 5.1: Average tip ratio for different ride types in NYC taxi rides from January 2009 to June 2016, and their count in millions.

Secondly, the *trip_distance* is simply the number of miles driven by the taxi. Figure 5.2 shows that the tip is generally smaller for longer rides. Moreover, the average of tip ratios on rides below one mile is 2.52% higher than the average of tip ratios on rides above ten miles. However, exceptions to the trend are taxi rides shorter than 0.2 miles (380 meters) which make only 0.07% of the dataset and can be seen as anomalies in this case. There is also an interesting small local maximum at the rides between nine and ten miles long which will not be examined further in this paper. This feature seems like it could bring predictive value to the later models but it is finally not included in the model because this negative trend is also later noticed with *obligatory_part* and *ride_duration* features. As these features are highly correlated, it is was decided to reduce redundancy by not including the ride's distance in the modeling phase.

Features derived from the original dataset

The first features that were derived from the original dataset were the obligatory part of the ride's price and the tip ratio (see Subsection 5.3.2). Figure 5.3 shows that the tip ratio is lower for more expensive trips. In other words, people are more prone to follow the social norm of tipping when the price is lower. For instance, it is easier

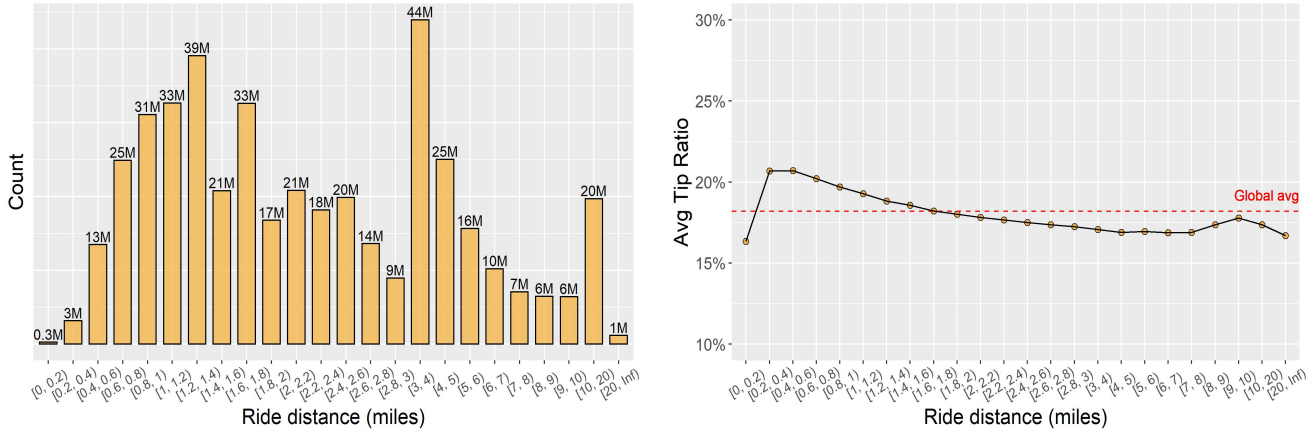

Figure 5.2: Average tip ratio for different taxi ride traveled distance intervals in NYC from January 2009 to June 2016, and their count in millions.

for the customer to pay a 20% tip on a $10 obligatory price (a $2 tip) than on a $100 obligatory price (a $20 tip).

Another reason is that there is a lot of people who tip an amount that is rounded to a dollar (2, 5, 10 dollars) that is mostly lower than choosing to give a tip as one of the default percentages (20%, 25%, 30%) of the ride's price. This behavior can be noticed in Figure 5.4 which shows a scatter plot of a sample of 48,104 taxi rides from the dataset. Particularly, it shows a great number of given tips on horizontal lines of round values such as 2, 5, 10, or 15 dollars, for various obligatory prices. Further analysis of these customers would be of great benefit for this research because the premise that customers give less tip percentage for more expensive rides is an ongoing phenomenon throughout this research. However, this exploration will be left for future work. In any case, the obligatory part of the taxi ride's price is a great predictor for the tip ratio and will be included in future models.

Additionally, some features from the original dataset contain useful information but can not be directly used for modeling. These features are related to the time and the location of the ride and have already been used in the filtering section (5.3.2) to derive new key features needed to ensure a higher quality of input data. These derived features were: trip duration, average speed, direct distance and ineffectiveness. The effect of change of these features to the average tip ratio is shown in Figure 5.5.

To start with, Figure 5.5 shows that tips are better for shorter rides. For example, rides that were shorter than five minutes had 3.12% higher tips than the rides longer than half an hour. In other words, the ride's duration has a similar effect on the tip ratio as the ride's distance. That makes sense because the taxi ride's duration and distance are highly correlated values. There is even the same anomaly for very short rides (here

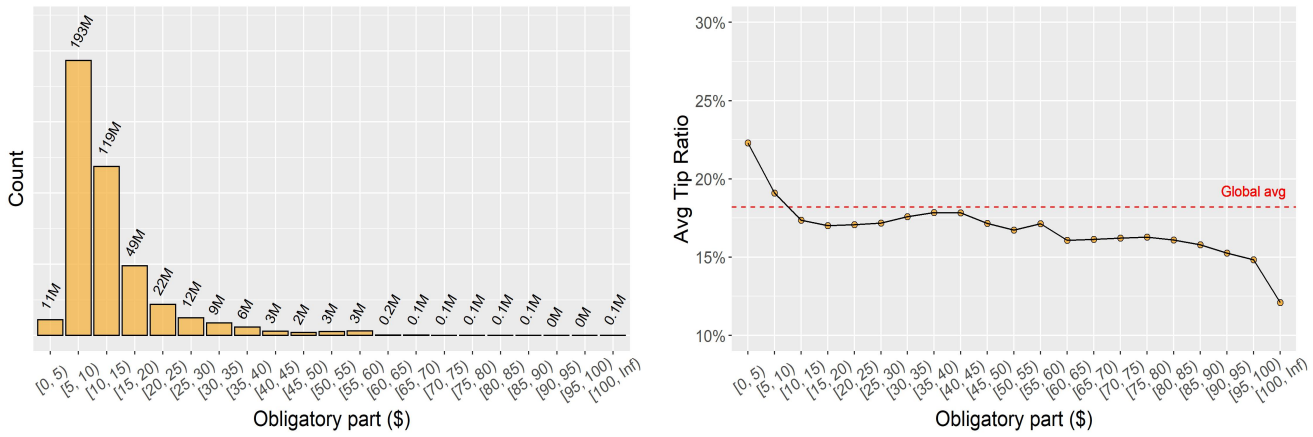

Figure 5.3: Average tip ratio for different intervals of obligatory taxi ride's price in NYC from January 2009 to June 2016, and their count in millions.

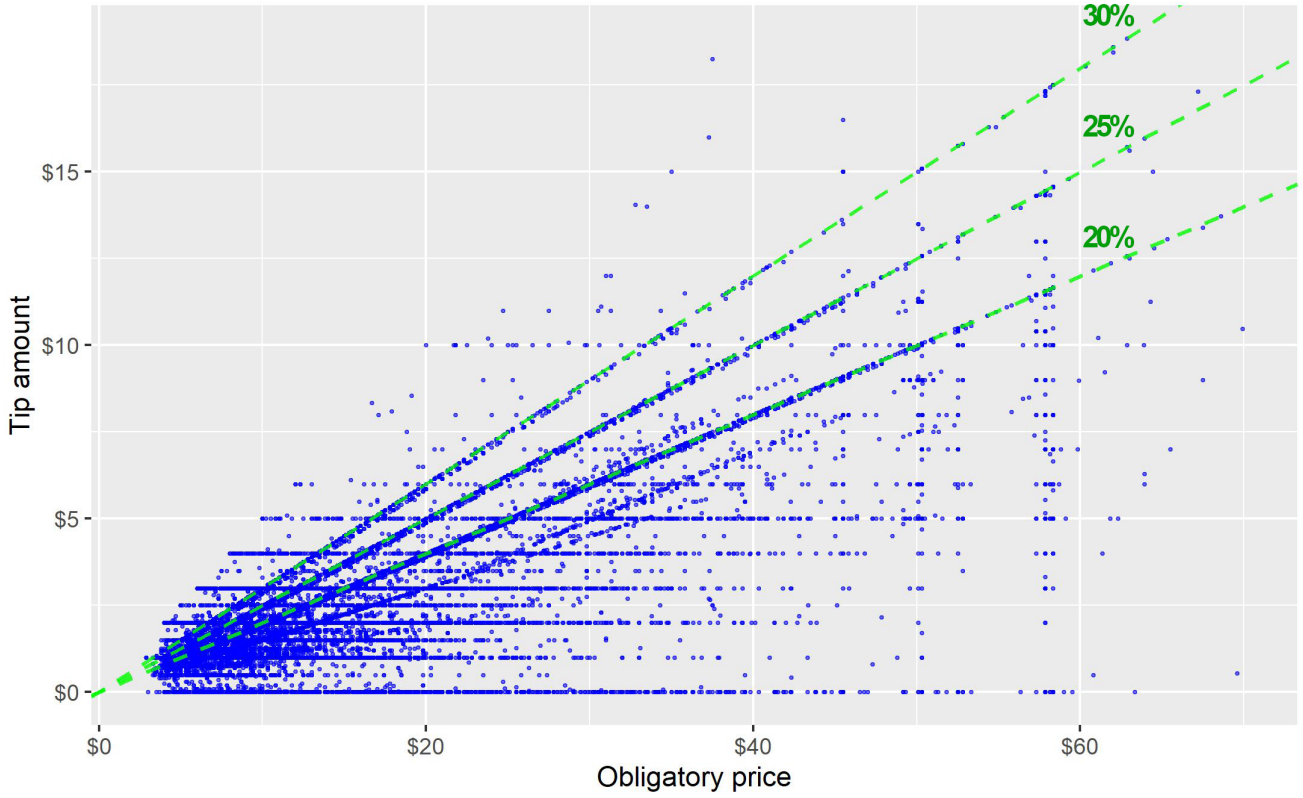

Figure 5.4: A plot of taxi tip amounts in relation to its obligatory price on a sample of 48,104 NYC taxi rides. The points on green lines represent rides that got a tip as a percentage of the obligatory amount, while the points on horizontal lines represent rides that got a rounded tip.

for rides shorter than a minute) where these rides have below-average tips. The ride's duration has been chosen to be included in the later model.

Another derived feature is the average vehicle speed. Figure 5.5 shows that tips are lower for rides with higher average vehicle speed which is an unexpected result. It would have been expected that a customer would be happier to travel with a higher average speed to their destination and would thus reward this service with a higher tip. However, that is not the case. The real reason for these results probably lies in the fact that the rides with lower average vehicle speeds belong to the rides going through the Manhattan area which is known for more well-off customers. Another reason could be that higher average vehicle speeds mean higher obligatory ride's price and could be related to the phenomenon described with the feature *obligatory_price*. This paper will not explore this effect any deeper and leaves it for further research. Whatever the reason, average vehicle speed shows that it could be useful for prediction and has been chosen to be included in later models.

One more derived feature is the direct (aerial) distance. Figure 5.5 shows that taxi rides that travel shorter aerial distances get higher tips. This behavior again traces back to the same phenomenon described with the *obligatory_price* because the rides with the shorter aerial distances have a smaller obligatory price and thus get higher tips. Despite the correlation, the feature could be important and is decided to be included in future models.

The ineffectiveness has been derived as the ratio of road distance and aerial distance traveled by taxi. It was expected to be a good feature to identify taxi rides that have been driving their customers using less-than-optimal routes. However, Figure 5.5 shows that this feature has practically no effect on the tip ratio. Only the first group of rides which has the ineffectiveness value smaller than one has the tip ratio higher than average. This group contains only two million rides and their values were mainly influenced by the error in the devices or the calculations because it should be impossible to have the value less than one by definition of the feature. Nevertheless, the feature was still decided to be included in the model mainly to see whether the model would find any importance in the feature.

Moreover, the ride's timestamp format of the time features (dropoff and pickup time) can not be directly used for modeling but can be used to derive other useful features such as the month, the year, the day of the week and the hour of the day of the taxi ride. The effect of change of these time-related features to the average tip ratio is shown in Figure 5.6.

The first time-related feature is the hour of the day. Figure 5.6 shows that there is

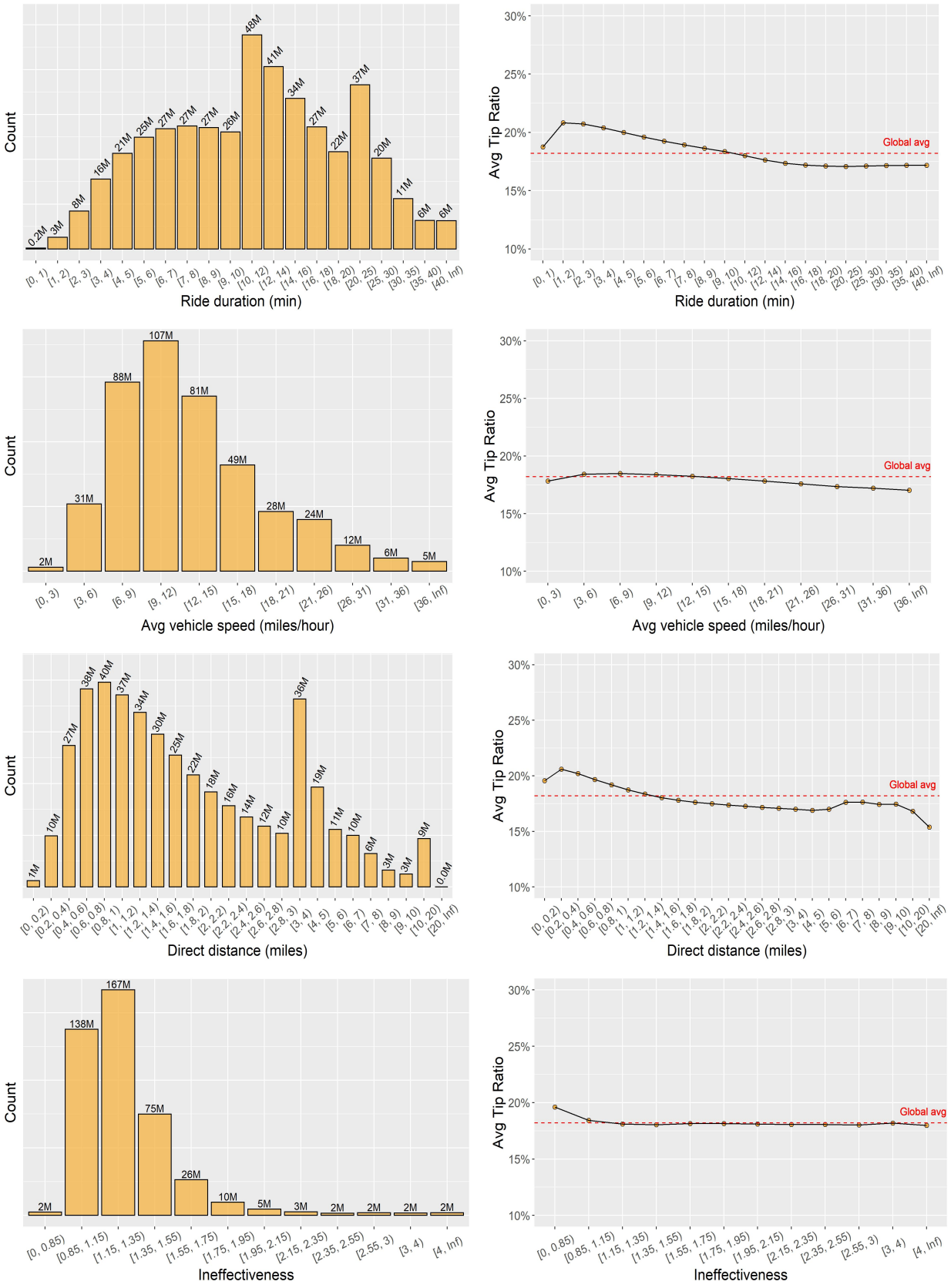

Figure 5.5: Average tip ratio for different value intervals of derived features (ride duration, average vehicle speed, direct distance and ineffectiveness) in NYC taxi rides from January 2009 to June 2016, and their count in millions.

a small deviation of tips throughout the day. On average, the smallest tips are given at five o'clock in the morning (17.45%) while the highest tips are received at one o'clock in the afternoon (18.60%). There is an additional smaller local minimum at eight o'clock in the night (18.03%) and local maximum at eleven o'clock in the night (18.26%). Intuitively, these observations could be the result of people's working and nightlife habits. However, these explorations will be left for future work. The hour of the day was decided to be included in the models.

The second time-related feature is the day of the week. As Figure 5.6 shows, there is a small influence of the day of the week to the average tip ratio given to the drivers. The lowest tips are recorded on Mondays (17.97%), while the highest are on Wednesdays (18.37%). Nevertheless, the feature will be included in future models.

The last two time-related features were the year and month in which the taxi ride took place. Figure 5.6 shows that it is hard to identify a pattern in the taxi ride's average tip ratios throughout the months and years. The years 2009 and 2011 have higher tip ratios while the years 2013 and 2014 have the lowest tip ratios. An anomaly is September 2010 with the lowest average tip ratio (17.62%) in the dataset. On the other hand, the highest average tip ratio was recorded for August 2009 (18.89%). There will be no more exploration of these patterns but the features of the month and year of the taxi ride might be useful and will be included in the models.

Features integrated from location data

As has already been mentioned in the dataset description section (5.3.1), the dataset contains some location-related features such as the dropoff and pickup latitude and longitude, as well as the corresponding pickup and dropoff taxi zone identifiers. These identifiers can be deduced from the dropoff and pickup geographical coordinates together with the official taxi zone shapefile that can be downloaded with the dataset from [32], but the dataset used in this paper had these identifiers already included. The official website [32] also includes a lookup table that contains names of each NYC taxi zone and its corresponding borough (region).

There are all in all 260 taxi zones in NYC divided into five main boroughs (Manhattan, Brooklyn, Bronx, Queens, Staten Island) and the Newark Liberty International Airport (EWR) that lies outside of all the other boroughs. The general map of NYC with the zones, boroughs and airports is shown in Figure 5.7.

Moreover, each of these taxi zones is shown in Figure 5.8, as well as the effect of the ride's pickup or dropoff zone to the ride's average tip ratio. It shows that the taxi

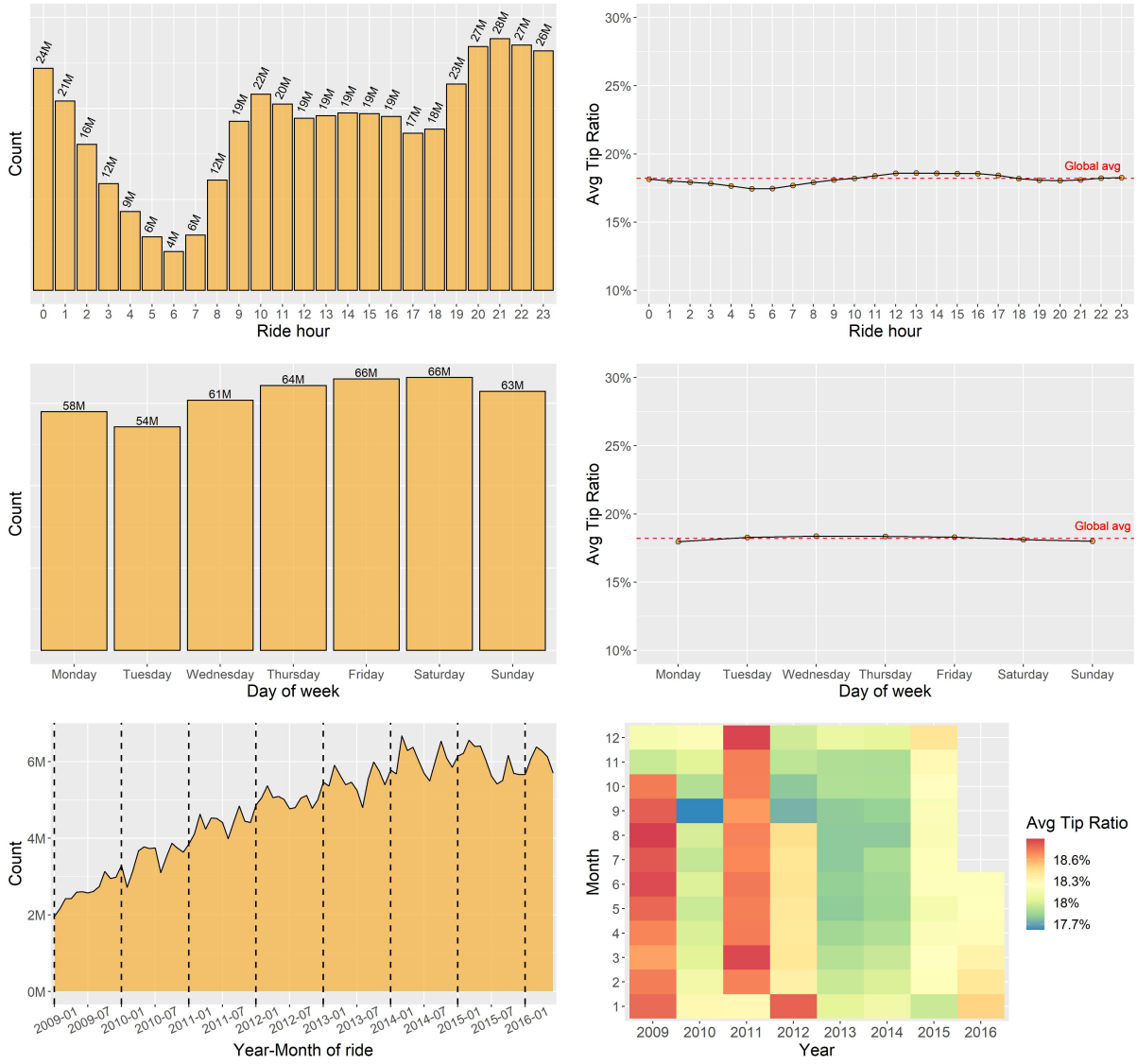

Figure 5.6: Average tip ratio for different days of week and months in years in NYC taxi rides from January 2009 to June 2016, and their count in millions.

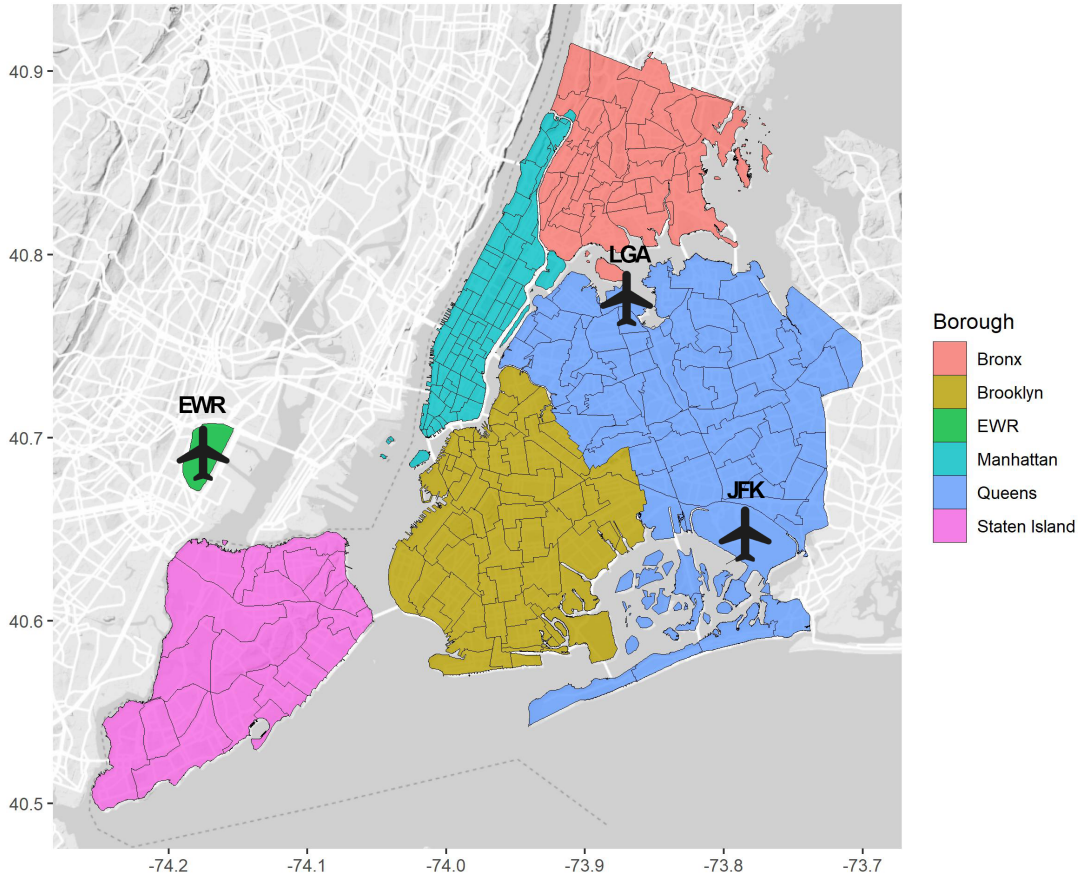

Figure 5.7: NYC taxi zones, airports and their corresponding boroughs.

zones in the boroughs of Bronx and Brooklyn get the lowest tips while the taxi zones in Manhattan seems to get the highest tip ratios for both the pickups and the dropoffs. It also shows that a majority of the pickups and dropoffs are done in the Manhattan area and the three airports.

The localization of higher and lower tips brings out curiosity about the general taxi ride's average tip ratios for dropoffs and pickups in different boroughs and airports. These relations are shown in Figure 5.9.

Firstly, it shows that 91.7% of pickups and 87.6% of dropoffs are located in the Manhattan area. The difference is interesting, though, because it means that taxi rides more often originate in Manhattan than they end there. In other words, taxi drivers are more often searching for clients in the Manhattan area. A reason for that could be that Manhattan is stereotypically the source of richer clientele. Despite the stereotypes, the rides that originate from Brooklyn (18.27%) have a slightly higher tip ratio than the rides originating from Manhattan (18.24%). Secondly, the areas from which

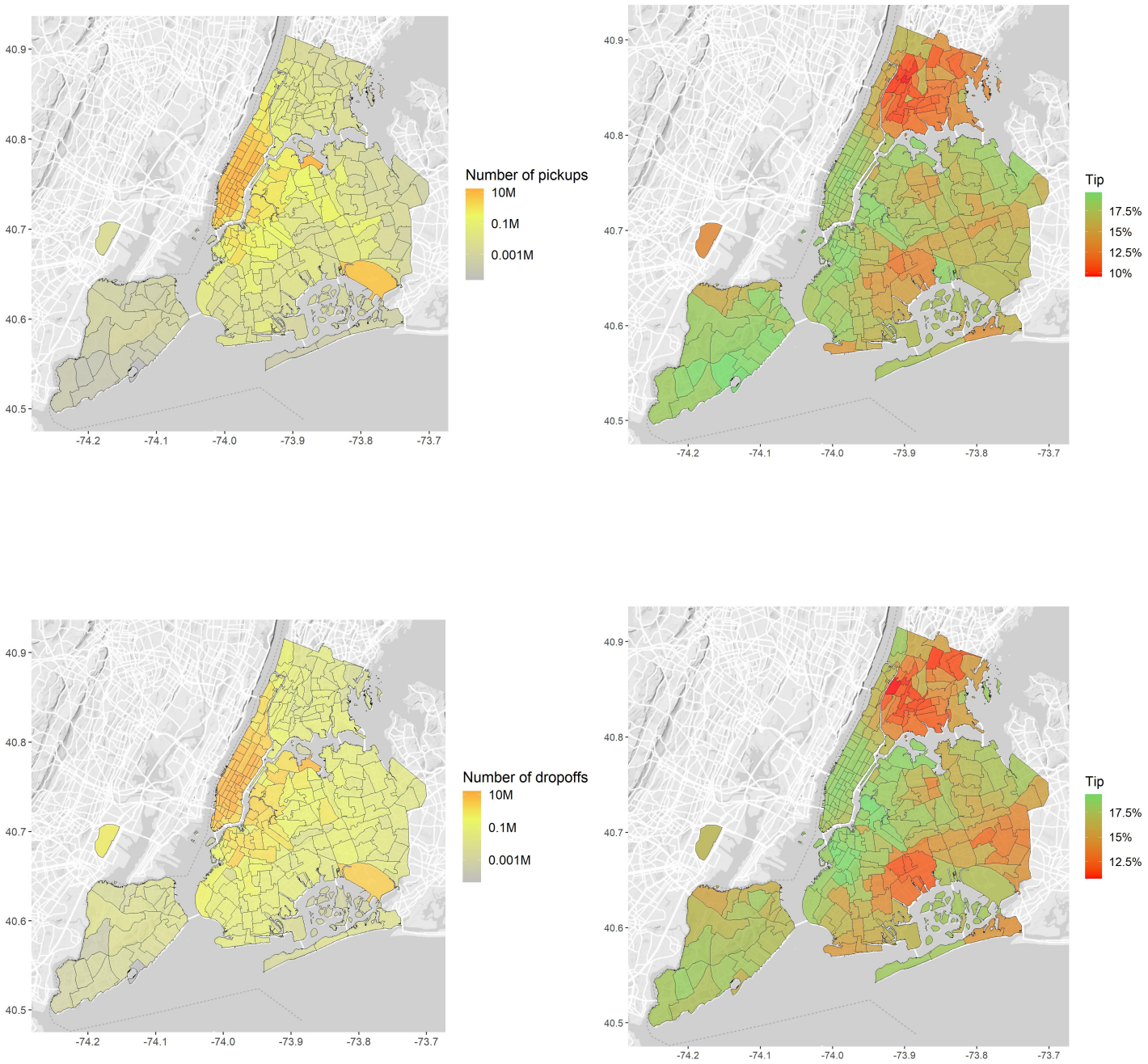

Figure 5.8: Average tip ratio for different taxi rides' pickup (upper two plots) and dropoff (lower two plots) zones in NYC from January 2009 to June 2016, and their count in millions.

the ride results with the lowest tips are Bronx (11.74%) and EWR Airport (13.39%). Interestingly enough, for both these areas, the average increases if they are the dropoff instead of the pickup location. A possible reason might be that the customers are more grateful to the drivers for accepting the ride and driving them far away from the Manhattan area. Whatever the reason, the difference between different dropoff and pickup locations is quite high and these features will be included in the models.

Apart from Manhattan, airports are additional hot spots for taxi drivers. There are three airports in NYC (see the map in Figure 5.7): John F. Kennedy Airport (JFK), LaGuardia Airport (LGA) and Newark Liberty Airport (EWR). EWR Airport is outside of the five boroughs, while the JFK and LGA airports are a part of the Queens borough. Figure 5.9 shows how going to or from airports effects the taxi rides' tip ratio. It shows that only 3.60% of the rides are going from an airport and 2.45% of the rides are going to an airport, of which LGA is the most popular and EWR is the least popular as both pickup and dropoff location. Furthermore, only 3,726 rides originated from EWR Airport in the seven and a half years that this dataset includes. It could be because EWR Airport is far from Manhattan, so the only pickups are probably just on the way back to the city.

Moreover, the data shows that the rides going from any of the three airports in NYC are getting higher tips than those going to an airport. This fact could be the result of tourists' better understanding of the United States' tipping norms on their way back than on their way to NYC. Another interesting fact is that, while the rides that are ending or beginning with the LGA Airport are getting higher tips than average, the rides from or to the other two airports are getting below-average tips. This result traces back to the same story with the customers paying less tip percentage for more expensive rides which was discussed when analyzing the effect of the obligatory part of the ride's price. In other words, rides to or from LGA Airport are less expensive due to two facts: LGA is closer to Manhattan than the other two airports, and the other two airports have a fixed charge that is higher than the standard metered fare applied in rides that do not include the two airports. Nevertheless, these features will be included in the future model.

Features integrated from weather data

As the dataset already contains information on the exact times of taxi ride's pickup and dropoff, this information could be used to find out the weather conditions on the day of the taxi ride. The relevant dataset was downloaded from [33] and was provided by

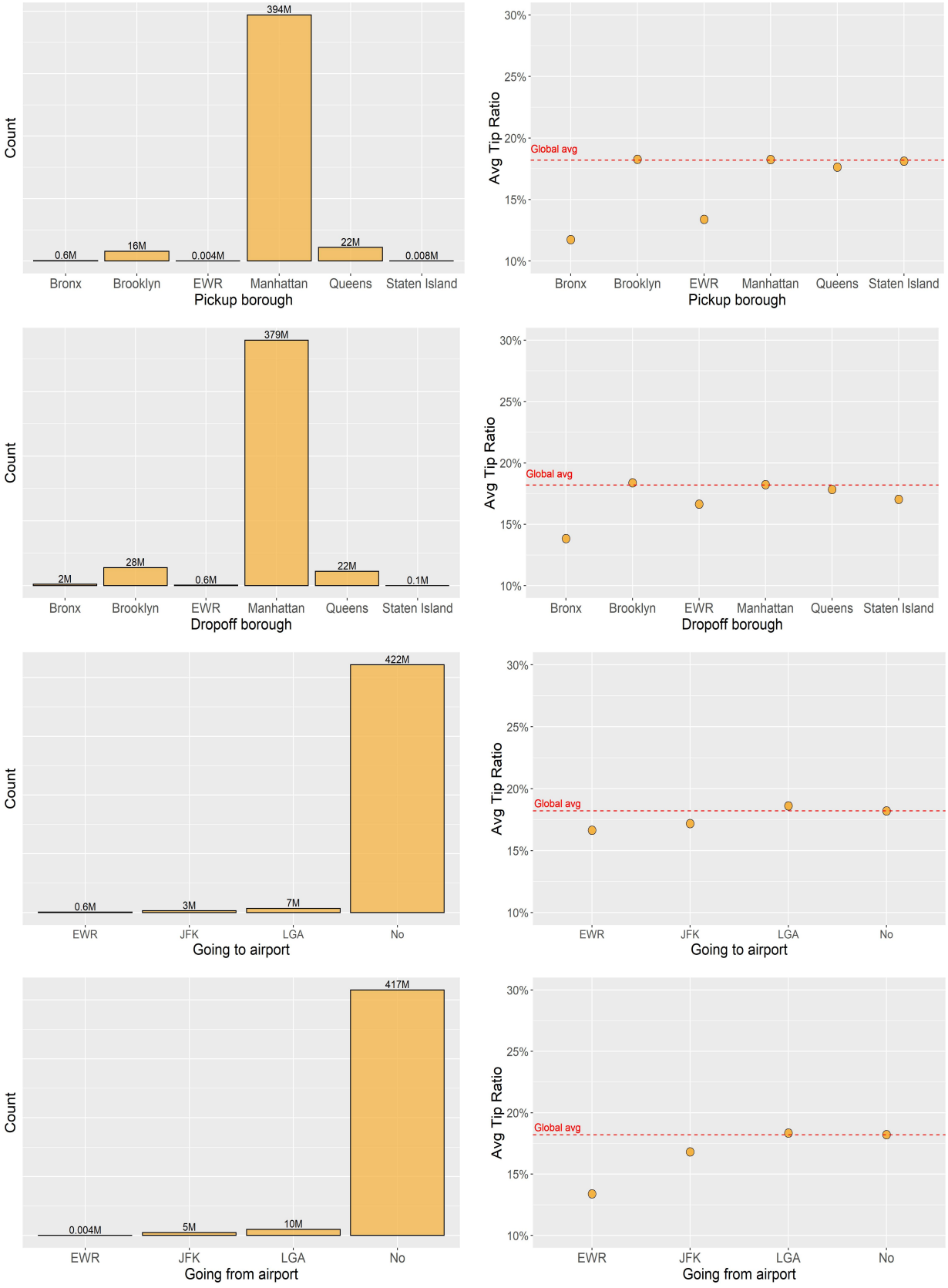

Figure 5.9: Average tip ratio for different taxi rides' pickup/dropoff boroughs or airports in NYC from January 2009 to June 2016, and their count in millions.

the relevant institution for weather information in the United States - National Oceanic and Atmospheric Administration (NOAA).

The downloaded dataset on weather information contains the following daily measurements:

- Precipitation (mm)
- Snowfall (mm)
- Snow depth (mm)
- Average wind speed (m/s)
- Minimum daily temperature (°C)
- Maximum daily temperature (°C).

Out of these six weather features, only the average wind speed had missing values in 4.3% of the days but was still included in the exploration.

Figures 5.10 and 5.11 show how the weather on the day of the taxi ride effects the tip. The results show a practically zero correlation across all six features. The average tip ratio is the same no matter the weather conditions on the taxi ride's day and, therefore, these features will not be included in the models.

Data visualization using *sparklyr* and *ggplot2*

Creating visualizations to notice patterns in big data is not an easy process. The main problem is that the dataset and all of the visualization's computation has to fit in memory for using the benefits of the most popular R's visualization package - *ggplot2*. However, when the dataset is too big, there are three options:

1. Aggregating big data to concise tables
2. Sampling big data to smaller datasets
3. Directly plotting big data using *dbplot*

In this paper, all of the visualizations have been done using the first technique except for the plot about the customers that tip a rounded amount in Figure 5.4 that was done by plotting a sample of the original dataset.

Firstly, aggregating big data comes down to executing aggregation functions on the dataset such as sum, mean, median, max, min, etc. Furthermore, if the observed feature is not discrete, it is firstly binned into categories to make the computation easier. Another problem is that it is not possible to use the *ggplot2* package with references to

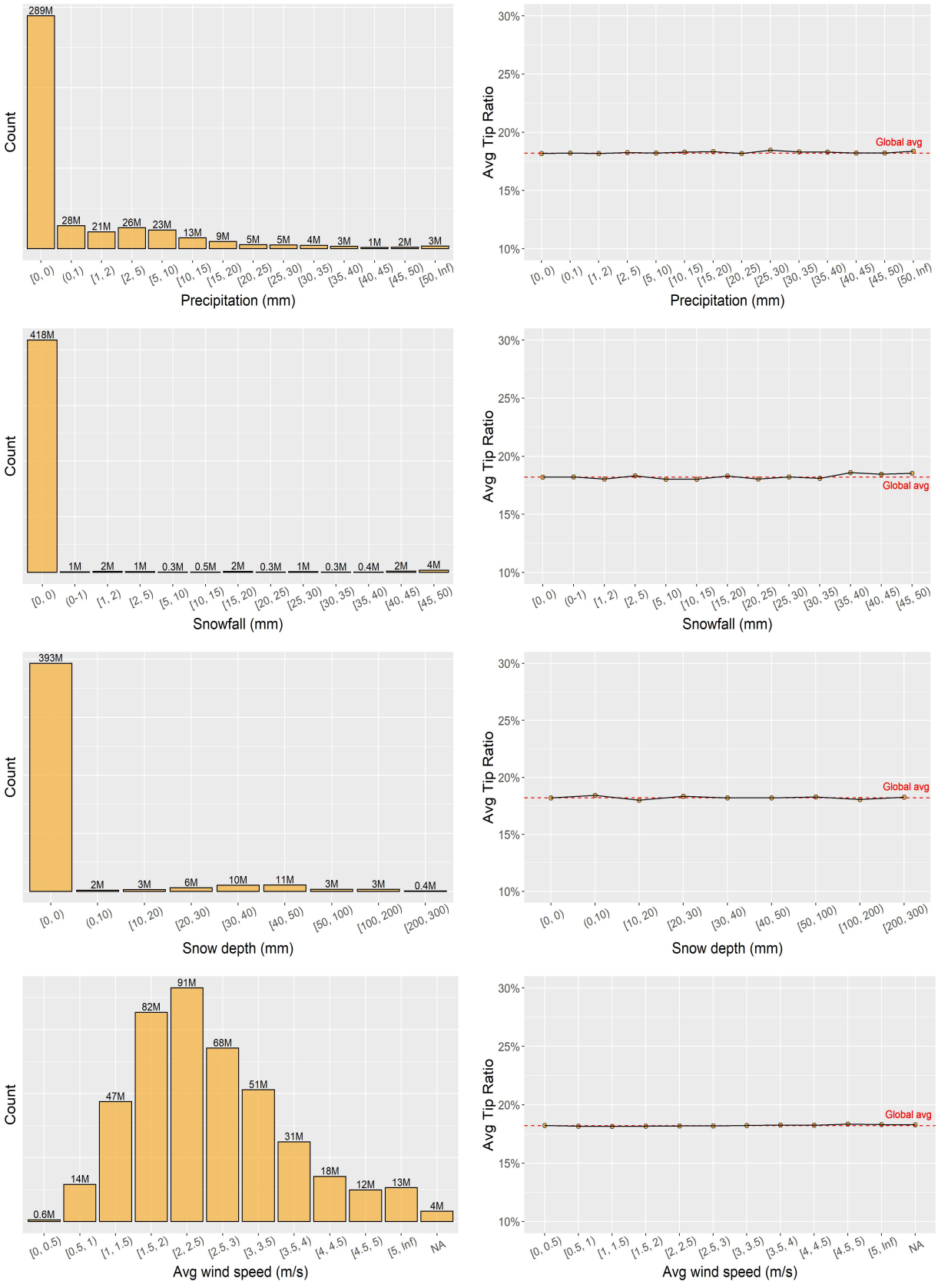

Figure 5.10: Average tip ratio for different value intervals of weather-related features (precipitation, snowfall, snow depth and average wind speed) on the taxi ride's day in NYC from January 2009 to June 2016, and their count in millions.

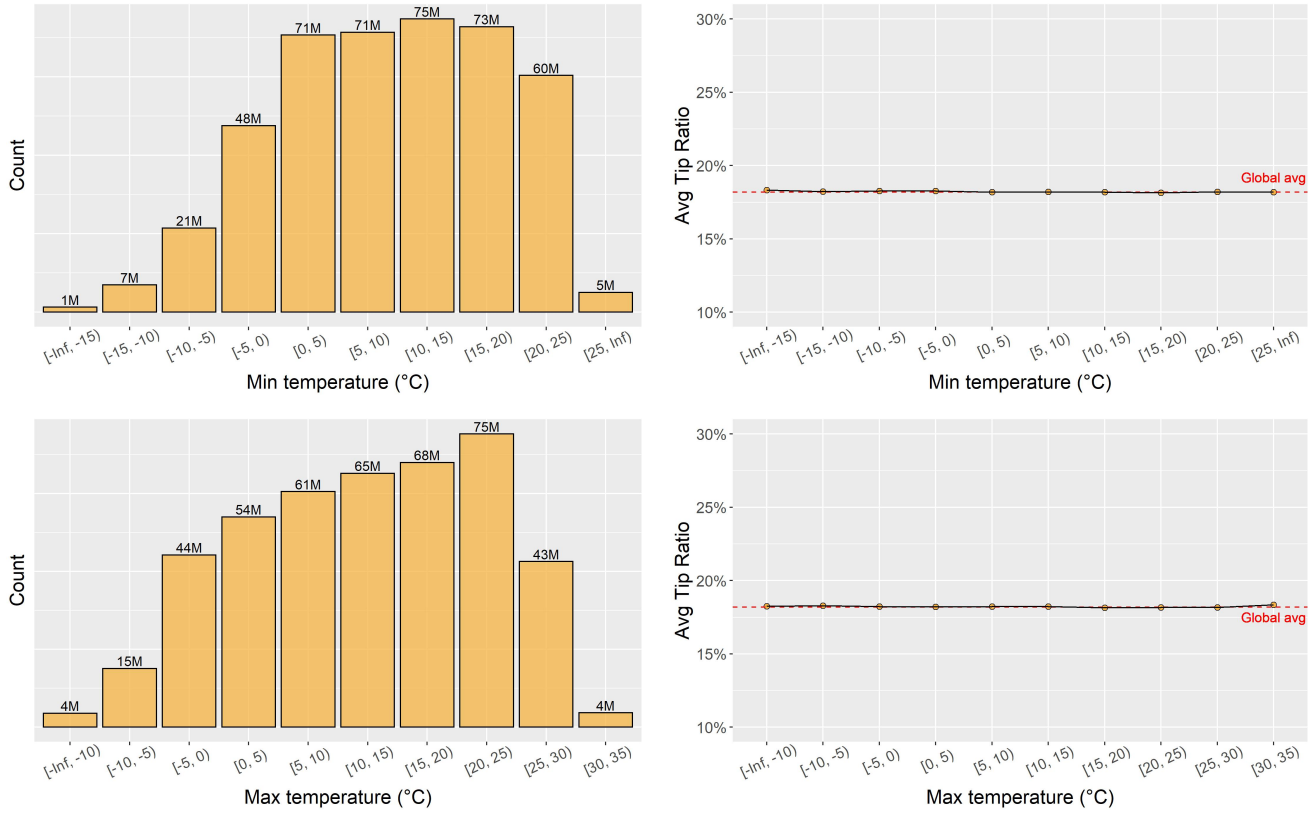

Figure 5.11: Average tip ratio for different value intervals of minimum and maximum temperature on the taxi ride's day in NYC from January 2009 to June 2016, and their count in millions.

Spark DataFrames (commonly known as *tibbles*) directly, but they need to be retrieved to the local R's *data.frame* object. This retrieval is possible using the *sparklyr*'s *compute* function. Example of aggregating data plotted in Figure 5.3 (about how the taxi ride's obligatory price effects the tip ratio received) is shown in code snippet 5.4.

Code snippet 5.4: Example of aggregating big data

```
nyc_taxi_data_obligatory_part <- nyc_taxi_data %>%
    # Binning the feature
    ft_bucketizer(
        "obligatory_part",
        "obligatory_part_bucket",
        splits = c(seq(from = 0, to = 100, by = 5), Inf)
    ) %>%

    # Aggregating values
    dplyr::group_by(obligatory_part_bucket) %>%
    dplyr::summarise(
        avg_tip_ratio = mean(tip_ratio),
        N= n()
    ) %>%

    # Retrieving data to a local tibble
    collect()
```

The retrieved aggregated data can then be visualized using *ggplot2* as if it was created from a smaller dataset. Additionally, the map of NYC and the average tips by zones (shown in Figures 5.7 and 5.8) were visualized using the *ggmap* package.

On the other hand, sampling may give better insights into hidden patterns in data as it was the case with noticing a large number of customers that gave a rounded tip in Figure 5.4. Sampling big data can be done using the *sparklyr*'s function *sdf_sample* as shown on code snippet 5.5.

Code snippet 5.5: Example of sampling big data

```
data_sampled <- data %>%
    sdf_sample(fraction = 0.001, replacement = FALSE) %>%
    collect()
```

5.4. Data mining

5.4.1. Feature selection

The exploratory data analysis has resulted in identifying sixteen taxi ride's key features that will be included as input for the later models. In this section, the key features are grouped by whether they were known before the start of the taxi ride (apriori knowledge) or only after it ended (aposteriori knowledge).

On one hand, apriori knowledge includes taxi ride's year, month, hour of the day, day of the week, pickup taxi zone, pickup borough, trip type and pickup airport. On the other hand, aposteriori knowledge includes all the features listed with the apriori knowledge and, additionally, the taxi ride's obligatory price, duration, direct (aerial) distance traveled, ineffectiveness, average speed, dropoff zone, dropoff borough and dropoff airport. They are grouped in such a way to investigate how well it is possible to predict the tip ratio before the ride itself as this knowledge would be of more use to the taxi drivers choosing where to pick up their passengers. On the other hand, predictions including aposteriori knowledge would be of more use to the taxi companies and the general scientific research of the tipping system.

5.4.2. Data formatting

Data has to be properly formatted before being used as input for machine learning models. The models need numbers as input and, therefore, it is necessary to convert the categorical features to one-hot vectors[1]. One-hot vectors are the best choice when there is no numerical relationship between the categories because the numerical distance between the one-hot vectors is always equal to one. Categorical features that will be used are pickup borough, dropoff borough, trip type, pickup taxi zone, dropoff taxi zone, pickup and dropoff airport. An example of how to convert a categorical feature to one-hot vectors is shown in code snippet 5.6.

Code snippet 5.6: Formatting categorical data for machine learning

```
nyc_taxi_data <- nyc_taxi_data %>%
    ft_string_indexer("trip_type", "trip_type_id") %>%
    ft_one_hot_encoder("trip_type_id", "trip_type_vec")
```

[1]One-hot vector is a vector in which all the dimensions are set to zero, but one dimension which is set to one. In practice, each vector's dimension represents a category of a categorical feature.

5.4.3. Splitting the dataset

It is a common practice in machine learning tasks to split the dataset to a train and test sets where the test set is used for training the model and the test set for evaluating the model's performance. For the abovementioned tip ratio prediction task, 75% of the dataset has been assigned to the train set and 25% to the test set. Code snippet 5.7 shows how this is done using the *sparklyr* package.

Code snippet 5.7: Splitting the dataset

```
partition <- nyc_taxi_data %>%
    sdf_random_split(train = 0.75, test = 0.25, seed = 8585)
nyc_taxi_data_train <- partition$train
nyc_taxi_data_test <- partition$test
```

The resulting train and test datasets contain 432,165,570 and 144,064,158 rows, respectively.

5.4.4. Models

Available regression models in *sparklyr*

The problem of predicting the tip ratio as described in the previous chapter is a regression problem[2] for which it is possible to use some of the core machine learning algorithms that *sparklyr* supports. All the models supported by the package are the same as the ones included in the Spark's machine learning library (MLlib). In other words, *sparklyr* just provides an R interface to MLlib's models. The supported regression models are listed in Table 5.5 together with their recommended scalability.

Model	Number of features	Training examples	*sparklyr* function's name
Linear regression	1-10 million	No limit	ml_linear_regression
Generalized linear regression	4,096	No limit	ml_generalized_linear_regression
Isotonic regression	Unknown	Millions	ml_isotonic_regression
Decision trees	1,000s	No limit	ml_decision_tree
Random forest	10,000s	No limit	ml_random_forest
Gradient-boosted trees	1,000s	No limit	ml_gradient_boosted_trees
Survival regression	1-10 million	No limit	ml_survival_regression

Table 5.5: Regression models supported through MLlib and their scalability [13]

[2]Predicting a continuous value out of correlated independent features

Moreover, Table 5.5 includes the name of the function in *sparklyr* which provides the R users with the corresponding model. An example of training, testing and evaluating a linear regression model which predicts tip ratio on any given combination of features (defined by the formula) is given with code snippet 5.8, but it can be used with any other model from the table.

Code snippet 5.8: Example of training and evaluating a linear regression model in *sparklyr*

```
# Define input and output features
ml_formula <- formula(tip_ratio ~ x + y + z)

# Train the model on the train set
ml_model <- ml_linear_regression(nyc_taxi_data_train, ml_formula)

# Run the model on the test set
predictions <- ml_predict(ml_model, nyc_taxi_data_test)

# Evaluate performance
(ml_regression_evaluator(
    x = predictions,
    label_col = "tip_ratio",
    prediction_col = "prediction",
    metric_name = "rmse")
)
```

Regression models used

From the available models, this paper will be using only the decision trees, the random forest and the linear regression model. The hyperparameters for the selected models are shown in Table 5.6.

Additionally, the models will be compared to a baseline model which always outputs the mean value (18.2%) for the tip ratio.

5.4.5. Evaluation metrics

Once the models have been trained, there is a need to decide which model performs better over the test set. This is done using the following available evaluation metrics in *sparklyr*:

Model	Parameter	Value
Linear regression	elastic_net_param	0
	reg_param	0
	max_iter	100
	loss	"squaredError"
Decision trees	max_bins	32
	max_depth	5
Random Forest	max_bins	32
	max_depth	4
	num_trees	15

Table 5.6: Selected models and correspondent hyperparameters used for training.

1. R-squared (R^2)
2. Root mean squared error (RMSE)
3. Mean squared error (MSE)
4. Mean absolute error (MAE)

Out of these four metrics, R-squared, MAE and RMSE will be used for evaluating the models' performance in this paper.

R-squared

R-squared (R^2) is a statistical measure that represents the proportion of the variance for a dependent feature that's explained by an independent feature or features in a regression model. Whereas correlation explains the strength of the relationship between an independent and dependent feature, R-squared explains to what extent the variance of input features explains the variance of the output feature. So, if the R^2 of a model is 0.50, then approximately half of the observed variation can be explained by the model's inputs. It is calculated as:

$$R^2 = 1 - \frac{MSE(model)}{MSE(baseline)}$$

where the baseline model is a model outputting the mean value for each input sample. R-squared values range from 0 to 1 and are commonly stated as percentages from 0% to 100%. Usually, the higher the R-squared value the better the model fits the

observations. It is hard to determine what is a high or a low value for the metric, it depends on the domain in question. For example, a value above 60% is expected for studies in the "pure" sciences such as chemistry, physics, biology, etc. However, a value as low as 10% might already be enough for studies in the fields of art, humanities and social sciences because their work is related to human behavior which is way less exact than nature's laws.

Root mean squared error

Root mean squared error (RMSE) is a quadratic scoring rule that measures the average magnitude of the error. It's the square root of the average of squared differences between the prediction and the actual observation.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(predicted\_value_i - observed\_value_i)^2}$$

Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means that the RMSE should be more useful when large errors are particularly undesirable.

Mean absolute error

Mean absolute error (MAE) is the mean of absolute differences between the target values and the values predicted by the model. It is more robust to outliers and does not penalize the errors as extremely as MSE or RMSE. MAE is a linear score which means all the individual differences are weighted equally. It is not suitable for applications where there is need to pay more attention to outliers. It is calculated as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|predicted\_value_i - observed\_value_i)|$$

5.4.6. Results

The performance of baseline, linear regression, decision trees and random forest models on the test set is shown in Table 5.7. It shows poor results by using both apriori and aposteriori knowledge about the taxi rides.

The best performing model using apriori knowledge is the linear regression model with a 0.9% R^2 metric which is almost the same as the baseline model, and is not

worthy of further exploration. On the other hand, the best performing model using aposteriori knowledge is the decision trees model with a 4.355% R^2 metric. This result is notable on such large datasets in the case of social sciences[3], which seems to be related to the domain of tipping because tips are partly a social construct.

An additional peculiarity is a fact that the two better models (decision trees and random forest) using aposteriori knowledge performed worse than the baseline in reference to the MAE metric. The reason lies in the fact that all the models were trained using quadratic error functions which is more useful for punishing large errors. Nevertheless, this shows that, although the current models might be useful for a better understanding of the tipping system in the United States, they are not particularly useful for predictions in the taxi service industry.

Model	**Apriori knowledge**			**Aposteriori knowledge**		
	RMSE	**R^2**	**MAE**	**RMSE**	**R^2**	**MAE**
Baseline	0.06924	0.00000	0.04971	0.06924	0.00000	0.04971
Linear regression	**0.06892**	**0.00912**	**0.04959**	0.06805	0.03403	**0.04934**
Decision trees	0.06903	0.00614	0.04962	**0.06771**	**0.04355**	0.04993
Random forest	0.06902	0.00626	0.04962	0.06784	0.03988	0.04989

Table 5.7: Evaluation results of various models using apriori and aposteriori knowledge about the taxi rides.

5.4.7. Residual analysis

A residual for any given taxi ride is the difference between the observed tip ratio and its prediction. This subsection will explore residuals for the selected models using aposteriori knowledge about the taxi rides, but will not do so for models using only the apriori knowledge because of their poor performance.

Figure 5.12 shows a scatter plot of around 19,000 sampled residuals for each of the models. Firstly, the plots show the models' general characteristic patterns. For instance, the decision trees model narrows its prediction to thirty-two discrete values due to the selected hyperparameters (see Table 5.6), whereas the linear regression model outputs continuous values that create a more disperse plot.

[3]Social science is the branch of science devoted to the study of human societies and the relationships among individuals within those societies.

Secondly, it shows that the random forest model is the least daring because its predictions are the most centered around the mean (18.2%). On the other hand, the linear regression model is the bravest because its predictions vary the most.

Lastly, even though the models' residual plots show their implementation differences, they also all show a similar declining pattern which is characterized by predictions being grouped on four or five declining lines. This pattern of residuals indicates that there is more to be explored on the topic of tips in NYC taxis, and leaves room for improvement in the research's future work.

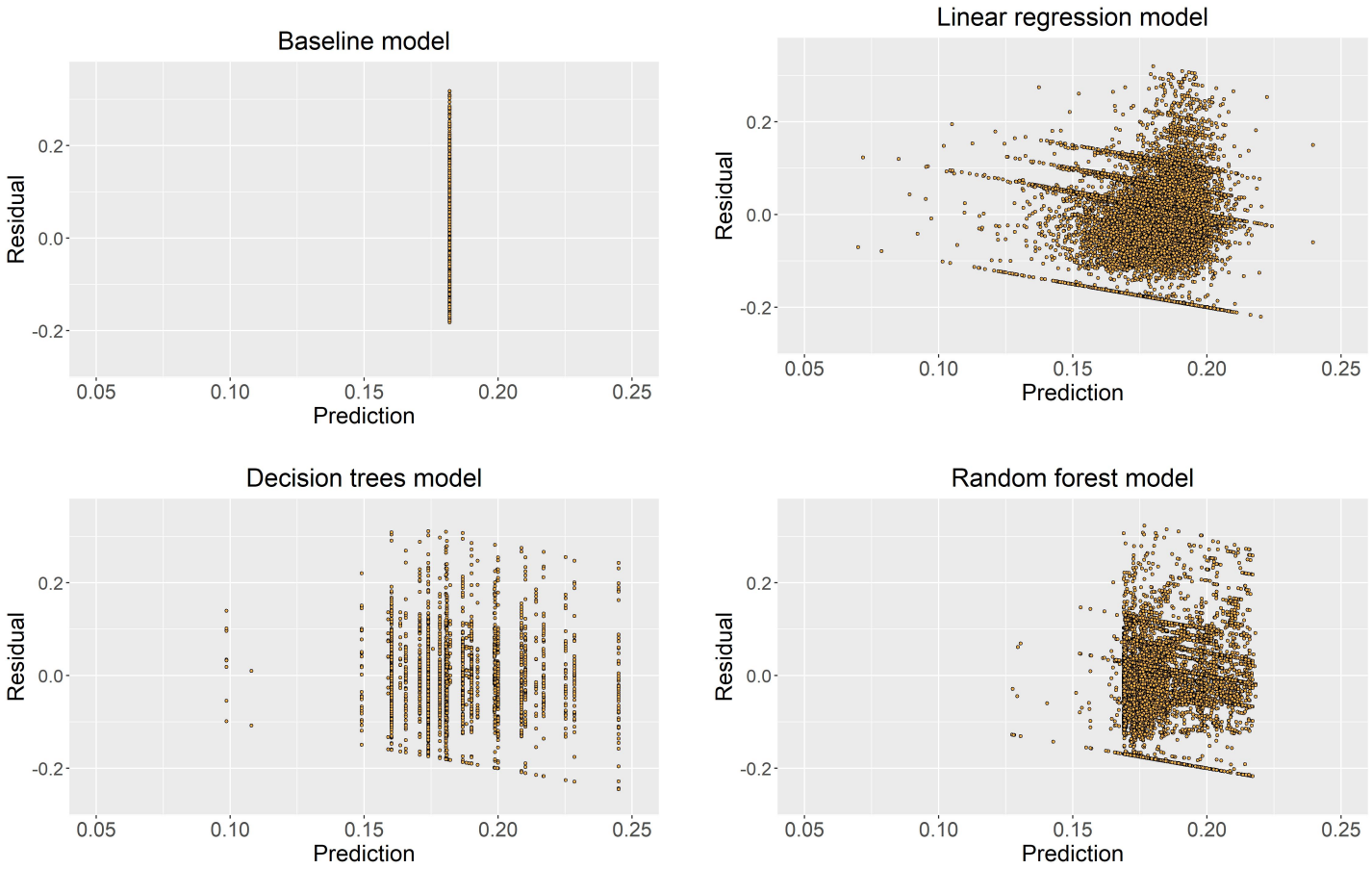

Figure 5.12: A sample of residuals from the used models while using aposteriori knowledge.

5.4.8. Feature importance

The importance of the input features may be extracted in *sparklyr* only from tree-based models such as the decision trees and the random forest models used in this paper. It can be done by applying the *sparklyr*'s *ml_feature_importances* which is a fast function because the feature importances are already contained in the trained model. On the

other hand, there is no simple way to extract feature importance or significance from the linear regression model without performing a feature selection algorithm which is always an exhaustive method when analyzing big data. Furthermore, models that used apriori knowledge showed poor performance and will also not be further explored. Therefore, only the decision trees and the random forest models using the aposteriori knowledge about the taxi rides will be explored in this subsection.

The decision trees model gave non-zero importance to only nine features which are shown in Figure 5.13. On the other hand, the random forest model found importance in thirty-one features of which ten are shown in Figure 5.14.

Both models identified the taxi ride's obligatory price to be the best predictor for the tip received. While the obligatory price dominates the decision trees model's importance (0.780), the random forest has divided this importance not only to the obligatory price (0.435) but also to the direct (aerial) traveled distance (0.211) and the trip duration (0.199) which are, again, correlated to the ride's obligatory price. Furthermore, both models found importance in the ride's year, type and whether or not the dropoff was in the Bronx borough.

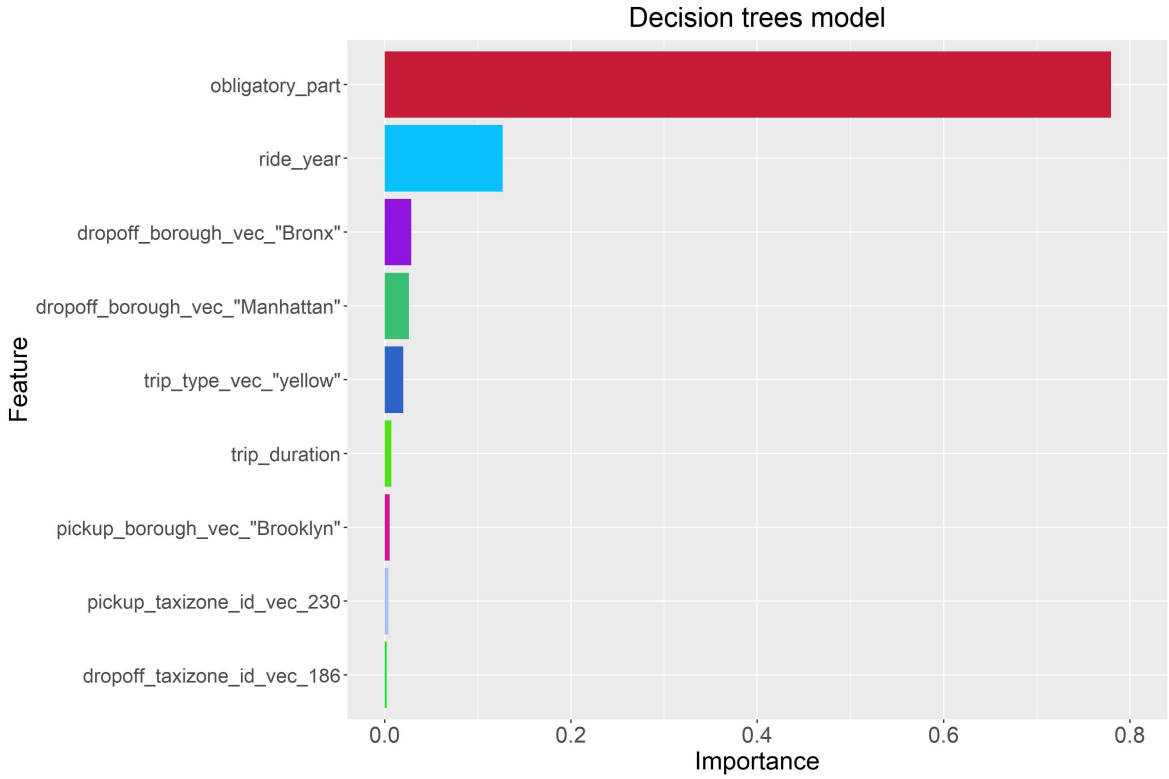

Figure 5.13: Feature importance extracted from the decision trees model.

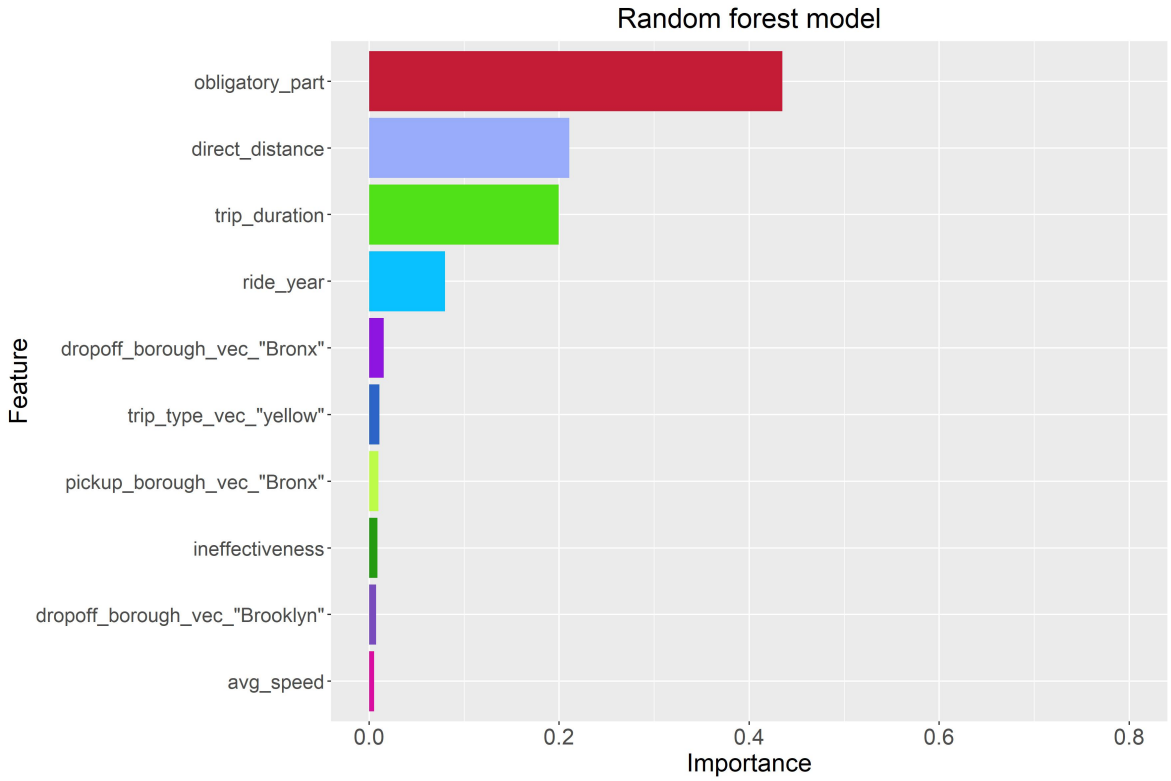

Figure 5.14: Feature importance extracted from the random forest model.

5.5. Conclusion of the analysis and future work

This analysis of tip ratios on over 1.3 billion NYC taxi rides showed that predicting tips is not an easy task. Even though twenty-three features have been considered including weather and location features, the highest performing model achieved an R^2 metric of 4.355%. This result may be notable on such a large scale in social sciences, and tips are, in its great part, a social construct. In other words, this research is supporting the statement that tips are mainly motivated by social pressure and that tips do not depend on the quality of received service or the customer's happiness.

The main supporting argument is how little correlation was found between the taxi ride's tip percentage and its weather-related (precipitation, snowfall, etc.) or its performance-related (average speed, ineffectiveness, etc.) features. Another supporting argument is that the most important feature for the task was the taxi ride's obligatory price, which means that customers are less prone to abide by social norms if its price is higher. Lastly, it is shown that the tipping norm has been increasing over the years and that it is different for different NYC boroughs.

As far as advice for taxi drivers goes, they can increase their tips by picking up passengers in the more well-off boroughs such as Manhattan and Brooklyn, or waiting

at the LGA Airport. Other than that, all the other indicators that could be known before the taxi ride have not shown to make a considerable effect on the received tip percentage.

The future work for this research would include data from later years (after 2016), different models and their hyperparameter tuning, and a more detailed analysis of the highest-giving and the lowest-giving tippers, as well as the customers that give a rounded tip amount. In addition, it may be interesting to try to perform clustering to identify all the different taxi drivers by their driving patterns and to use this prediction as input to predicting tips. Besides, the residual analysis showed a pattern with several declining lines across all the used models which means that there is more to be explored on the topic.

6. Conclusion

This paper argued that big data processing is an increasingly challenging and profitable field in computer science, and it addressed common practical problems in exploratory and data mining analysis of big data in a distributed environment. Moreover, the paper proposed distributed computing as a solution for big data processing and the Apache Spark platform as its powerful representative.

Furthermore, it presented an example of a workflow of big data processing using the Apache Spark platform and the R programming language by analyzing tip percentages in over 1.3 billion taxi rides in New York City from January 2009 to June 2016. The analysis included twenty-three different taxi ride's features and three different machine learning models in predicting the taxi ride's tip percentage.

However, the results showed that none of the models would be useful for predicting tips in the taxi industry as their predictions are on average almost equal or worse than the baseline model. Additionally, the results showed that the best model performed with the R^2 metric equal to 4.355% which indicates that the variance of the tip ratio is poorly explained by the variance of any of the other input features. Consequently, it is reasonable to conclude that predicting tips is a difficult task and that tips are socially motivated and have little correlation to the quality of service or the customer's state of mind. Additionally, the research showed that the social norm of tipping is different for different areas of New York City, and that it is rising over the years.

BIBLIOGRAPHY

[1] Internet usage statistics, 2019. URL https://www.internetworldstats.com/stats.htm. 25.4.2020.

[2] The top 20 valuable facebook statistics, 2019. URL https://zephoria.com/top-15-valuable-facebook-statistics/. 25.4.2020.

[3] Number of smartphone users worldwide, 2016. URL https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. 25.4.2020.

[4] Iot devices connected, 2016. URL https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. 25.4.2020.

[5] How much data is generated every minute?, 2018. URL https://www.socialmediatoday.com/news/how-much-data-is-generated-every-minute-infographic-1/525692/. 25.4.2020.

[6] Global datasphere, 2019. URL https://www.datanami.com/2018/11/27/global-datasphere-to-hit-175-zettabytes-by-2025-idc-says/. 25.4.2020.

[7] John Spacey. What is big data?, 2019. URL https://www.oracle.com/big-data/guide/what-is-big-data.html. 25.4.2020.

[8] Change perspective on big data growth, 2019. URL https://www.forbes.com/sites/louiscolumbus/2018/05/23/10-charts-that-will-change-your-perspective-of-big-datas-growth/#6a9f17729268. 25.4.2020.

[9] Apache hadoop ecosystem, 2017. URL http://blog.newtechways.com/2017/10/apache-hadoop-ecosystem.html. 10.5.2020.

[10] What is hdfs?, 2018. URL https://www.quora.com/What-is-hdfs. 15.4.2020.

[11] Hdfs and yarn tutorial, 2016. URL https://www.simplilearn.com/hdfs-and-yarn-tutorial. 11.5.2020.

[12] Mapreduce, 2017. URL https://www.oreilly.com/library/view/distributed-computing-in/9781787126992/5fef6ce5-20d7-4d7c-93eb-7e669d48c2b4.xhtml. 11.5.2020.

[13] Matei Zaharia Bill Chambers. *Spark: The Definitive Guide*. O'Reilly Media, 2018.

[14] Edgar Ruiz Javier Luraschi, Kevin Kuo. *Mastering Spark with R: The Complete Guide to Large-Scale Analysis and Modeling*. O'Reilly Media, 2019.

[15] Education, languages and salary, 2019. URL https://www.kaggle.com/kailex/education-languages-and-salary. 25.5.2020.

[16] Crisp-dm popularity poll, 2014. URL https://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-data-mining-data-science-projects.html. 2.5.2020.

[17] Crisp-dm and why you should know about it, 2017. URL https://itsalocke.com/blog/crisp-dm-and-why-you-should-know-about-it/. 2.6.2020.

[18] Brief history of tipping, 2019. URL https://www.tripsavvy.com/a-brief-history-of-tipping-1329249. 5.5.2020.

[19] Twenty-three years and still waiting for change, 2014. URL https://www.epi.org/publication/waiting-for-change-tipped-minimum-wage/. 5.5.2020.

[20] Why do we tip?, 2016. URL https://www.pbs.org/newshour/economy/why-do-we-tip. 5.5.2020.

[21] When you make $2.13 an hour, tips really, really matter, 2016. URL https://www.cnbc.com/2018/04/14/minimum-wages-for-these-workers-havent-gone-up-since-1996-thats-a-problem.html. 5.5.2020.

[22] Jeffrey Graves Michael Lynn. Tipping: An incentive/reward for service? 12 2017.

[23] Who tips more, men or women, 2017. URL https://learningenglish.voanews.com/a/who-pays-better-tips-men-or-women/3941732.html. 5.5.2020.

[24] Gabriela Quintana. http://www.opportunityinstitute.org/blog/post/im-going-to-tip-minority-servers-more-and-white-servers-less/. 10 2018.

[25] Chalk another one up to the generation gap: Millennials tip less and stress about it more, 2019. URL https://www.washingtonpost.com/news/voraciously/wp/2019/12/20/chock-another-one-up-to-the-generation-gap-millennials-tip-less-and-stress-out-about-it-more/. 5.5.2020.

[26] The psychology that motivates tipping, 2017. URL https://www.bbc.com/worklife/article/20171122-the-psychology-that-motivates-tipping. 5.5.2020.

[27] 2016 tlc factbook, 2016. URL https://www.cnbc.com/2018/04/14/minimum-wages-for-these-workers-havent-gone-up-since-1996-thats-a-problem.html. 5.5.2020.

[28] How much money do nyc cab drivers make on average (before taxes but including tips)?, 2018. URL https://www.quora.com/How-much-money-do-NYC-cab-drivers-make-on-average-before-taxes-but-including-tips. 5.5.2020.

[29] Taxi & limousine services industry in the us - market research report, 2019. URL https://www.ibisworld.com/united-states/market-research-reports/taxi-limousine-services-industry/. 30.5.2020.

[30] New York Taxi and Limousine Commission. New york taxi data 2009-2016 in parquet fomat. 2017. URL http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

[31] Geographical distance, 2016. URL https://en.wikipedia.org/wiki/Geographical_distance. 7.5.2020.

[32] Tlc trip record data, 2020. URL https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page. 9.6.2020.

[33] Noaa - weather in nyc data, 2020. URL https://www.ncdc.noaa.gov/cdo-web/datasets/GHCND/stations/GHCND:USW00094728/detail. 15.5.2020.

Eksploratorna i dubinska analiza u okruženju Velikih podataka

Sažetak

Kako raste ukupna količina podataka u svijetu svake godine, tako i analiza podataka postaje sve zahtjevniji zadatak. Ovaj rad prolazi kroz uobičajeni tijek eksploratorne analize i rudarenja velikih podataka u raspodijeljenom okruženju za obradu podataka. Točnije, predstavljena je analiza napojnica u taksi vožnjama u New York City-u od siječnja 2009. do lipnja 2016. godine korištenjem programskog jezika R i platforme Apache Spark. Iako je istraženo dvadeset sedam značajki i trenirana tri modela, istraživanje pokazuje da je predviđanje napojnica u taksi vožnjama težak zadatak i da su napojnice najviše motivirane društvenim pritiskom.

Ključne riječi: Apache Spark, R, veliki podaci, sparklyr, eksploratorna analiza podataka, rudarenje podataka, sustav napojnica, taksi vožnje

Exploratory and Data Mining Analysis in Big Data Environment

Abstract

As global datasphere is growing faster every year, data analysis is becoming an increasingly challenging task. This paper goes through a big data exploratory analysis and data mining workflow using a distributed data processing environment. Specifically, it presents an analysis of tips in New York City's taxi rides from January 2009 to June 2016 using the R programming language and the Apache Spark platform. Although twenty-three features have been explored and three different models have been trained, the research shows that predicting tips in taxi rides is a difficult task and that tipping is mostly motivated by social pressure.

Keywords: Apache Spark, R, big data, sparklyr, exploratory data analysis, data mining, tipping system, taxi rides