POLITECNICO DI MILANO
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

Using Functional Dependencies to discover Data Bias

Supervisor: Prof.ssa Letizia Tanca
Co-supervisor: Dr. Fabio Azzalini

M.Sc. Thesis by:
Chiara Criscuolo, 928714

Academic Year 2019-2020

To all my family and friends, nothing could be done without you.

Abstract

Computers and algorithms have become essential tools that pervade all aspects of our daily lives and, since this technology is based on data, for it to be reliable we have to make sure the data on which it is based on is fair and without bias. As a consequence, **Fairness** has become an important topic of interest within the field of *Data Science Ethics*, and in general in Data Science.

Today's applications should therefore be associated with tools to discover bias in data, in order to avoid (possibly unintentional) unethical behavior and consequences; therefore, technologies that accurately discover discrimination and bias to obtain fair databases are badly needed.

There are already emerging technologies that detect biases and discover discrimination in datasets. In this thesis, we propose a novel solution, called ***FAIR-DB*** (**F**unction**A**l dependenc**I**es to discove**R** **D**ata **B**ias), that exploits the notion of **Functional Dependency**, a particular type of constraint on the data. The proposed solution is implemented as a framework that focuses on the mining of such dependencies, also proposing some new metrics for evaluating the bias found in the input dataset.

By means of *data mining techniques*, our tool can identify the groups that are discriminated and the groups that verify various fairness measures in the dataset; moreover, based on special aspects of these metrics and the intrinsic nature of dependencies, the framework also checks group and subgroup fairness, obtaining more insight about the already existing bias in dataset than other tools.

Finally, our system also suggests possible future steps, by indicating the most appropriate (already existing) algorithms to correct the dataset on the basis of the computed results.

Sommario

Computer e algoritmi sono diventati pervasivi in tutti gli aspetti della nostra vita quotidiana, e dato che la tecnologia in generale è basata sui dati, ed essa è affidabile solamente se i dati sui quali essa è costruita sono etici e senza alcun tipo di bias.
Ne consegue quindi, che il concetto di **Fairness**, in modo approssimativo tradotto come *equità*, è diventato un argomento di interesse rilevante nell'area del *Data Science Ethics* e in generale nella *Data Science*.

Ci sono già tecnologie emergenti che riconoscono bias e discriminazioni all'interno dei dataset.
In questa tesi proponiamo una soluzione nuova, chiamata ***FAIR-DB*** (acronimo di **F**unction**A**l dependenc**I**es to discove**R** **D**ata **B**ias), basata sulle **Dipendenze Funzionali**, le quali sono un particulare tipo di vincolo esistente nei dati. La soluzione proposta è un framework che si focalizza sull'estrazione di tali dipendenze, proponendo nuove metriche che possono valutare i bias trovati in suddetti dataset.

Grazie alle tecniche di *Data Mining*, il nostro sistema è in grado di identificare nei dati i gruppi maggiormente discriminati e anche quelli invece che verificano i controlli di equità; inoltre, grazie alla particolarità delle metriche e alla innata natura delle dipendenze, il nostro sistema riesce ad eseguire controlli di *group and subgroup fairness*, ottendendo maggiori informazioni su i bias già presenti nei data, rispetto ad altri strumenti già esistenti.

In conclusione, il nostro sistema suggerisce anche possibili passi da eseguire al termine della ricerca; indichiamo alcuni algoritmi già esistenti che, a partire dai risultati finali del nostro sistema, risolvono i bias nei dati.

Acknowledgements

I begin saying that this thesis is for me a first concrete realization in the Computer Science Engineering field. A work that really interested me, that has certainly barriers and difficulties, but that has given to me many joys. For this reason, the acknowledgements are even more necessary and they originate directly from my soul.

Firstly, I want to express my gratitude to my supervisor, prof. Letizia Tanca. Collaborating with her is really a pleasure. She has followed me constantly and her contribute is fundamental to improve the realisation of the thesis. Her strong passion, humanity and professionalism is so clear and evident from the first meeting, that it motivated me. Together with my supervisor, I want to thank my assistant supervisor, Fabio Azzalini. They both supported me during this work much more than what I would have ever expected and their advice are very precious.

To my family and friends, they are not only the basement of my life, but they give me the motivation, the support and the encouragement to finish the university and in particular, this thesis.
I want to thank in particular my two grandmothers, Olga and Immacolata, they are not only part of my name, but they are also the inspiring women that are more important for me.
I thank my siblings that assist me every day and my parents that give me not only the financial resources to study in Politecnico di Milano, but most important, the faith and the love.
To all my friends, they are a lot, but each of them has a special part in this work. Each of them is irreplaceable and vital. I thank all of you.

Contents

Chapter 1

Introduction

1.1 Motivation and Context

The research area in which this thesis focuses on is *Data Science Ethics*, a new branch of Ethics that studies and evaluates moral problems related to data, algorithms and corresponding practices, in order to formulate and support morally good solutions.
Another area related to this research is the *Functional Dependencies* field and consequently the Databases area. This thesis is also related to the sub-areas of Data Mining and Machine learning studies on Data Science Ethics.

The scenario that motivates this research is characterized by the presence of bias and unfairness in data that affect the technology nowadays.
In recent years, *fairness* has become an important topic of interest in Data Science Ethics. Algorithms and computers have been pervasive in all aspects of our daily lives as they have become essential tools based on data. It is therefore of societal and ethical importance to ask whether data and datasets, that are the starting points of every activity, are fair or not.

Mainly, the necessity of Data Science Ethics arises to avoid bias and discrimination in Machine Learning tasks.
In fact, the application domain of Machine Learning is pervasive and can influence different spheres, and real world problems, thus these algorithms received more attention also for the ethical consequences of applications.
For example, recent breakthroughs in justice, as discrimination against black people, reveal just how much the discovery of data bias is urgent. Technology is based on data, and technology is reliable only if the data on which it is based on is fair and without bias. This is critically important.

Applications nowadays need the discovery of bias in data to avoid unintentional behaviours and unethical consequences, so the presence of technologies that detect biases and accurately discover discrimination to obtain fair databases is more and more necessary.

1.2 Using Functional Dependencies to discover Data Bias

To obtain fair databases there are emerging technologies that detect biases and discover discrimination.

In the Machine Learning field, there are *statistical tests* to check fairness in data, computed combining the real data and predictions based on data models.
Nowadays, there are many implementations of most of the techniques based on the statistical tests.

Our approach it is not based on statistical tests, but it is based on a particular type of constraint on data: **Functional Dependencies**. In a Functional Dependency the *values of some attributes of a tuple uniquely (i.e. functionally) determine the values of other attributes of that tuple.*
Specifically, the database designer needs to specify constraints to correctly model the real world, but it could happen that some Functional Dependencies, that are not imposed in the design phase, are actually satisfied by the data itself. We will see later that, we can relax this notion obtaining various kinds of dependencies.

Our framework, called ***FAIR-DB*** (**F**unction**A**l dependenc**I**es to discove**R** **D**ata **B**ias), extracting a specific type of dependencies from data, can analyze bias already present in the dataset and consequently check fairness.

Our research is based on a systematic literature review over the main Machine Learning studies in Data Science Ethics area. From these surveys, we formulate hypothesis, and through the usage of an already existing algorithm that finds Functional Dependencies [34], we are able to check them.
This iterative development focuses on the mining of dependencies and on the metric study: some measures already existed, others are derived from mathematical reasoning. The obtained framework is also tested simulating experiments with two real world datasets.

At the end, we compare our framework with another very well-known one [37], evaluating the results and drawing conclusion from this comparison.

1.3 Structure of the thesis

Here, we provide a brief description for each chapter that follows this introduction.

- Chapter 2: **Data Science Ethics**. In this section we define the area in which our work is located: the Data Science Ethics. Starting from the notion of Ethics and Computer Ethics, we explain what Fairness, Data Bias and Discrimination mean. Finally, we propose a theory that explains the origin of unfairness in data, the Invisibility Factor.

- Chapter 3: **Preliminaries**. In this section we define all the theoretical tools and concepts that we considered for our research. In the first part, we start from the notion of Relational Databases and we explain in details what is a Functional Dependency. In the second part, we present the Data Mining area and its connection with Functional Dependencies, concluding with a final overview of the most used metrics.

- Chapter 4: **Literature Review**. In this section we start presenting a comprehensive summary of all the fairness definitions analyzing previous researches done in the area of Data Science Ethics, especially in the Machine Learning context. Finally, we give an overview of the most important works and applications that aim to solve the problem of discovering data bias.

- Chapter 5: **Methodology**. In this section we will provide a description of the methodology on which our framework to discover already existing bias in data is based. For each phase, we proposed a running example, from the U.S. Census Adult dataset, on which we present how the framework works. Then, we propose a workflow over other two datasets: a minority study of the U.S. Census Adult dataset and a workflow over the Titanic dataset.

- Chapter 6: **Experimental Results**. In this section we present a comparison between our framework and the one presented in paper "Nutritional Labels for Data and Models" [37]. We start presenting this tool, then with two concrete examples, we enumerate the main differences and similarities between this work and our framework.

- Chapter 7: **Conclusions and Future Works**. In this section we draw the conclusions of our work and we suggest possible future research directions.

Chapter 2

Data Science Ethics

In this section we will define the area in which our work is located: the Data Science Ethics. Starting from the notion of Ethics and Computer Ethics, we will define the objectives of this discipline.
We will explain what *Fairness* means, starting from the notion of Data Bias, in order to understand the different types of fairness. We will discuss when a dataset can be said **unfair**, **biased** or **discriminatory**. In this section we present the definitions of *Data Bias*, *Fairness* and *Discrimination* not only in Data Science, but also in psychology and philosophy.
In our opinion, mapping the terrain of the philosophical debate with Data Science issues and locating those definitions provide helpful clarifications for this type of research on fairness, and it can raise relevant questions which have yet to be considered.
Finally, we propose a theory that explains the origin of unfairness in data, the **Invisibility Factor**. The following definitions create the *substratum* of Data Science Ethics, bringing the reader in this huge area, in order to understand why the definition of "database without bias" is not so rigorous as one could expect.

2.1 Ethics

There is not a common definition of Ethics. In general, it is a branch of knowledge dealing with *moral principles* and it is the set of rules that help in distinguish between good and wrong.
Ethics is the **systematic reflection on what is moral**, it is not a manual with answers, but it is a process for searching the right kind of morality.
The first definition of Ethics in the world of computer technology is **Computer Ethics**. Defined by James Moor in 1985, Computer Ethics is the

analysis of the nature and social impact of computer technology and the corresponding formulation and justification of policies for the ethical use of such technology [29, p.266].
From his notion, understanding the social and ethical implications of our choices about computers and information technologies, studying the present technologies and steering the development of future ones in a direction that is good for humanity appear essential and necessary.

Data Science Ethics is a *new branch of Ethics that studies and evaluates moral problems related to data, algorithms and corresponding practices, in order to formulate and support morally good solutions* [15, p.1]. Data Ethics is built on the foundation provided by Computer Ethics, but, at the same time, it refines the approach endorsed so far in this research field, by shifting the level of abstraction of ethical enquiries, to being *data-centric*.
Data Science Ethics highlights the need for ethical analyses to concentrate on the content and nature of computational operations, the interactions among hardware, software, and data, rather than on the variety of digital technologies that enables them. Furthermore, it emphasises the complexity of the ethical challenges posed by Data Science.
In particular, for our research there are concepts that are necessary to explain before entering in this huge area, so we will define the notion of Data Bias, Fairness and Discrimination.

2.2 Data Bias

Bias is a broad concept that has been studied across many disciplines, such as social science, cognitive psychology or law. In statistics, a **Data Bias** is a *systematic distortion in the sampled data that compromises its representativeness* [33, p.6]. The representativeness is related to the scope of the activity. In the majority of the cases, the samples should be representative of a larger population of interest, and should also well represent the content being produced by different groups.
To clarify better the concept of data bias, we report two basic definitions in the context of decision-making:

- A **Historical bias** is an already existing bias caused by socio-technical issues in the world [38].

- An **Algorithmic bias** is when the bias is not present in the input data and is added purely by the algorithm [17].

2.3 Fairness

There is no universal definition of *Fairness*, especially because it is a wide concept that intersects different scenarios. For instance, philosophers have been debating about various definitions of fairness for centuries and psychologists have been trying to measure how people perceive fairness for decades. Now, along come the computer scientists, attempting to make computing and algorithmic decision making systems fair.
For philosophers, Fairness is usually related to the concept of **justice**. It involves what is right and equal and it can be interpreted as being equal in provision, in opportunity or in result [35].
In psychology, there are manly three ideas related to fairness [20]:

1. **Sameness**: the fairness where everything is equal. It means ensuring that everyone gets the same things with respect to others, so no one has more than another. For example, in a generic shop everyone pays a fixed price for the same item.

2. **Deservedness**: in this notion of fairness you get what you deserve. One case could be employees that in a company, do not divide resources equally between individuals, but adopt divisions based on merit.

3. **Need**: this idea of fairness is that those who have more to give should give a greater percentage of what they have to help others who are unable to contribute much, if anything at all. Fairness here takes into account the facts that humans have obligations to one another, and the more one has, the more is demanded of that person to contribute to the common good. For example, in Italy, taxes and medical health care costs are applied with this criterion.

Philosophy and psychology have tried to define the concept of fairness long before computer science started exploring it.
It's interesting that, from these ideas of fairness that come from other fields, computer scientists shaped their definitions that fall under three different types [28]:

1. **Individual Fairness**: Give similar predictions to similar individuals. It follows the idea of *Sameness*.

2. **Group Fairness**: Treat different groups equally. It is related to the idea of *Deservedness* and *Sameness*. In fact, given a company and dividing employees into two groups "Male" and "Female", group fairness implies the same average salary for both.

3. **Subgroup Fairness**: Subgroup fairness intends to obtain the best properties of the group and individual notions of fairness. It takes the idea of *Deservedness*, but analyzing it more in deep. Taking the previous example, subgroup fairness means that men and women take the same salary, if they do the same task in the company.

Given all these three main groups, we can build more than twenty notions of fairness according to specific needs. Although, the *Chouldechova's Impossibility Theorem* [6] states that we can choose any three notions and reach the impossibility result. In fact, it is not possible to guarantee in a dataset that more than two definitions of fairness are satisfied simultaneously.

To summarize, in the context of decision-making, **Fairness** is the *absence of any prejudice or favoritism toward an individual or a group based on their inherent or acquired characteristics* [36, p.100].
In order to understand how we can have so many definitions of fairness and select the more appropriate, it is also crucial to understand the different kinds of *discrimination* that can occur.

2.4 Discrimination

For the *Stanford Encyclopedia of Philosophy* [2], **Discrimination** are *actions, practices, or policies that are "in some appropriate sense" based on the (perceived) social group to which those discriminated against belong and that the relevant groups must be socially salient in that they structure interaction in important social contexts.*
Discrimination against people, then, is necessarily oriented toward them based on their membership in a certain type of social group.

To deal with discrimination in Data Science Ethics, there emerged the notion of *protected attribute* that is a characteristic for which non-discrimination should be established, like age, race, sex, etc.
From this notion, in Data Science Ethics there are many types of discrimination, and here we report the most common ones:

- **Direct discrimination** happens when protected attributes of individuals explicitly result in non-favorable outcomes toward them [42].

- In **Indirect discrimination**, individuals appear to be treated based on seemingly neutral and non-protected attributes; however, protected

groups or individuals still get to be treated unjustly as a result of implicit effects from their protected attributes [42].

- **Systemic discrimination** refers to policies, customs, or behaviors that are a part of the culture or structure of an organization that may perpetuate discrimination against certain subgroups of the population [9].

We present some examples of these types of discrimination in the next paragraph, so that the reader can understand the differences between one another and why they are so important.

The fact that no universal definition of fairness exists and there is still no clear agreement on which definition is better, shows the difficulty of giving a definition of *unfair dataset*.
To conclude, fairness is an incredibly desirable quality in society, and how many datasets are fair in real life? If unfairness is present in data and applications, it is necessary to go in deep and analyze the origin of this behaviour to possibly correct and solve this issue. We provide some examples of unfairness in data and present a theory that explains the causes of this problem.

2.5 Invisibility Factor

A possible cause of unfairness in data is described by the theory of *James Moor* on the **Invisibility Factor** [29]. This notion is presented in the discussion over the meaning and the responsibility of *Computer Ethics* presented in the previous section.
To support the importance of Computer Ethics, the author proposed the **Invisibility Factor** as central concept. The invisibility factor expresses the idea that computer operations are invisible, so often generates *policy vacuum*, because we could be dimly aware of the internal processing in computers.
The author presents three different types of invisibility factor which can have ethical significance:

- **Invisibility of abuse:** it can be caused by an algorithm that, through intentional unfair decisions and not transparent processes, can conduct in a obscured way, to unmoral actions.

- **Invisibility of programming values:** invisible programming values are those values which are embedded in a computer program. Not

paying attention to them can cause intentional or unintentional biases performing unethical conducts.

- **Invisibility of complex calculation:** due to calculations that are too complex for human inspection and understanding, it is difficult to justify the model and the computations done, so that the behaviour of the application could be not moral.

To better discuss all these three types of invisibility, we present the three definitions in details. For each one of them, we propose an example presented by Moor in the original paper of the specific type of invisibility together with a contemporary case in Data Science, to show how this theory is actual and strong.

2.5.1 Invisibility of abuse

As defined before the first type of invisibility factor, the invisibility of *abuse*, implies the **intention** of unethical decisions and possibly the generation of biases. So, due to this specific kind of invisibility, people can suffer illegal actions done with a computer.
Moor presents a classical example for this type on invisibility, that is a programmer that intentionally **steals excess interest from a bank** with a computer. This is an ordinary stealing, but involves Computer Ethics. Indeed, this behaviour shows that new type of abuses are possible thanks to technology and that they need to be detected and prevented.

In real world there are many cases of invisibility of abuse behaviours, and we present one coming from *ProPublica* to support the strengths of this theory in Data Science Ethics [21].
This journal reports that, the social network **Facebook** allows advertisers to put *ads that exclude people* based on race, gender and other sensitive factors that are prohibited by federal law in housing and employment. Excluding specific groups from some ads is discriminatory and illegal. The privacy and public policy manager at Facebook said: "It's important for advertisers to have the ability to both include and exclude groups as they test how their marketing performs.". Facebook assigns to members an *Ethnic Affinity* based on pages and posts they have liked or engaged with on Facebook. Facebook's *micro-targeting* is particularly helpful for advertisers looking to reach niche audiences [21].
In conclusion, this kind of abuse was caused by algorithmic biases that invalidate *individual and group fairness*. In fact, this evident example

of **direct discrimination** against various groups of people reveals the intentional choice of unethical decisions made with computers, based on a business model that get money from this data, allowing to perform illegal actions.

2.5.2 Invisibility of programming values

The second type of invisibility involves *programming values* which are embedded in a computer program that can cause **intentional or unintentional** biases.
Moor says that *sometimes invisible programming values are so invisible that even the programmers are unaware of them. Programs may have bugs or may be based on implicit assumption which do not become obvious until there is a crisis* [29, p.273].
The author cited a famous example: a **computerized airline reservation** suggests always first one company with respect to others, even if it does not show the best flight available. There may be a difference between how a programmer of a reservation service intends its program and how it is actually used. How the airlines are listed will build certain values into the program and may bring to this kind of invisibility.
In Data Science, this kind of invisibility is presented with an example of discrimination against black people in American justice [22]. *ProPublica* explains that a **software** used across the United States to predict future criminals is **biased against black people**. In fact, the scores that express the risk of recidivism of white people are lower than the ones of black people. It is not clear today why the software gives these results and if the intent of its programmer was to discriminate black people or not. Moreover, we cannot state if it is an example of **direct or indirect discrimination**. ProPublica analysis shows that the protected attribute *Race* seems quite predictive of a higher score [19], so, probably, the programmer did not pay the right attention to the creation of these programming values during the development of this software introducing algorithmic biases. To conclude, in this case, *group* and in particular *subgroup fairness* was not checked.

2.5.3 Invisibility of complex calculation

The invisibility of *complex calculations* can bring to unfairness due to the difficulties in **understanding** the model and the context, because they are too hard to analyze or even beyond the human comprehension. The author says that *computers today are capable of enormous calculations, so often*

they are too complex for human inspection and understanding [29, p.273]. Moor presents an example that is a program that solved the problem of the **four color conjecture**. It showed that four colors are sufficient to color a map so that no adjacent areas have the same color. Programmed computers proved the four color conjecture and the mathematical proof of it is known and examined, but is too enormous for human to completely understand it.
Going back to Data Science, there is very interesting a study by *Carnegie Mellon University* on ads [10]. It tests how online user behaviors and interaction with **Google's Ad Settings** would impact ads displayed, in particular, setting the user profile gender to female results in getting fewer instances of an ad related to high paying jobs than setting it to male [10]. However, the authors of the study admit that the gender discrimination shown is difficult to pin to one factor, due to the complexity of not only Google's profiling systems, but also of the way advertisers buy and target their adverts using Google.
So, in this last case is very hard to understand the model and the complex calculations and all the programming values that bring to gender discrimination. The type **discrimination is indirect** and probably **systemic** due to the complexity of the environment, so there is a possible historical and algorithmic bias. Again, *group fairness* was not respected also in this case.

Given all these examples, in the next section we analyze the **justifications** of the theory of the invisibility factor and the possible **questions** that can arise from it.

2.5.4 Strengths and solutions of the invisibility factor

The invisibility factor explained by James Moor seems a very strong theory because presents all possible causes of unfairness in data. So, why it is so reliable? We present the last part of Moor's article with the strengths of his theory and a possible solution to the invisibility factor.

In Moor's paper, the invisibility factor has two strong consequences [29]:

- On one hand, the invisibility factor makes us **happy**, because we do not inspect every transaction or computation making processes more efficient,

- On other hand, it makes us **vulnerable**, because the policy vacuums created by these types of invisibility have an not neutral impact on our lives.

We think that these two points are true as the cases aforementioned confirmed.
Facebook example [21] shows on one hand, the efficiency as the central part of a business model and on other one, the reality of the vulnerability. Computers have made our lives efficient and easier, but the price to pay for it is the presence of discrimination and unfairness in **many fields**, from justice to hiring and in many other areas. Indeed, it is necessary to decide when we can trust the invisible calculations and programming values and when not.

As a possible framework to solve the invisibility problem, Moor introduces the concept of **policy**. Policies, that could be strategies and actions, are necessary to identify and prevent abuses caused by invisibility, in particular in a sensitive area as Data Science. An example of policy could be to check the possible types of discrimination and fairness in a dataset to avoid unethical conducts. So, a central task is to *formulate and justify* such policies according to the *definitions of fairness, bias and discrimination* that we have mentioned in the first section. Without a clear and precise terminology, we are not able to prevent and detect abuses with the right policy in every possible case.

There are many emerging **technologies that detect biases** and accurately promote fairness. To create policies with these tools, it is necessary to be conscious of the notions of fairness and apply the proper one to the specific case.
In our research we propose a specific type of policy, and associated framework, in which we detect bias and discrimination in dataset through a particular type of constraints: Functional Dependencies.
Before introducing our framework, in the next section we will present the preliminaries that are the necessary basics to understand the database area and the relative constraints.

Chapter 3

Preliminaries

In this section we will define all the theoretical tools and concepts that we considered for our research.
In the first part, we will start from the notion of Relational Databases and we will explain in details what is a Functional Dependency. In this section we present a comprehensive summary of previous researches on a Functional Dependencies and all relative techniques used to mine the different types of dependencies.
In the second part, we will present Data Mining with particular attention on Association Rules and its connection with Functional Dependencies. We conclude this section with an overview of the metrics used to evaluate diverse types of dependencies.

3.1 Relational Databases

When we talk about computers, one of their main uses is to manage information, which in some cases involves simply holding data or exploit them to retrieve interesting information.
A large amount of data stored in a computer is called a database. **Database**, also called electronic database, *is any collection of data, or information, that is specially organized for rapid search and retrieval by a computer* [32]. Databases are structured to facilitate the storage, retrieval, modification, and deletion of data in conjunction with various data-processing operations. Database systems can be viewed as mediators between human beings who want to use data and physical devices that hold it. A database model provides the means for specifying particular data structures, for constraining the data sets associated with these structures, and for manipulating the data [1].
A **Relational Database** is *a digital database based on the relational model*

of data, as proposed by E. F. Codd in 1970 [7]. The term *relational model* has come to refer to databases models that are based on the concept of relation table as the fundamental data structure that incorporates some or all of the query capabilities, update capabilities, and *integrity constraints* (i.e., properties that are supposed to be satisfied by all instances of a database schema).

Because of its simplicity, the relational model has provided an excellent framework for the first generation of theoretical research into the properties of databases. In a relational database, the *data is represented in tables* in which each row gives data about a specific object or set of objects, and rows with uniform structure and intended meaning are grouped into tables.

Play Tennis				
Outlook	Temperature	Humidity	Wind	Play
sunny	hot	high	false	don't
sunny	hot	high	true	don't
overcast	hot	high	false	play
rain	mild	high	false	play
rain	cool	normal	false	play
rain	cool	normal	true	don't
overcast	cool	normal	true	play
sunny	mild	high	false	don't
sunny	cool	normal	false	play
rain	mild	normal	false	play
sunny	mild	normal	true	play
overcast	mild	high	true	play
overcast	hot	normal	false	play
rain	mild	high	true	don't

Table 3.1: Relational table or relation: it presents the decision to whether to play tennis or not according to the weather

Each table is called a **Relation** and it has a name (e.g. in table 3.1 the title is *Play Tennis*). The columns also have names, called **Attributes** (e.g, Outlook, Temperature, Humidity, Wild, Play). Each line in a table is a **Tuple** (or record). The entries of tuples are taken from sets of constants, called **Domains**.

Finally, we distinguish between the **database schema**, which specifies the structure of the database; and the **database instance**, which specifies its actual content.
If for example, we observe the table 3.1, the dataset contains the possible weather conditions and for each one of them, the column *Play* says if it is possible to play tennis or not.
Furthermore, in table 3.2 we present all the symbols that will be used to explain Functional Dependencies.

Symbol	Description
R	relation schema
X, Y	sets of attributes
A,B	attributes
a,b	attribute values
π_X	projection on X

Table 3.2: Symbol Table

To be more clear we make the connections between table 3.1 and Table 3.2:

- R is the relation schema, so the table *Play Tennis* without the values.
- X represents the set attributes $\{Outlook, Temperature, Humidity, Wind\}$; Y represents the set of attributes, in this example Y is composed only by the class attribute $\{Play\}$.
- A could represent any component of X, and B any component of Y.
- a,b represent any value given respectively by A,B.
- π_X means select some tuples of set of attribute X, for example: $\pi_{Outlook}(PlayTennis) = \{sunny, overcast, rainy\}$.

3.2 Functional Dependencies

Starting from *integrity constraints*, that are the properties that are supposed to be satisfied by all instances of a database schema, the study of more *restricted classes of constraints*, called dependencies, has become more and more interesting in last years.
The fundamental motivation for the study of constraints is to incorporate

more semantics into the relational model.
A **Dependency** is *a constraint that applies to or defines the relationship between attributes*. It occurs in a database when information stored in the same database table uniquely determines other information stored in the same table.

In practise, the database designer needs to specify integrity constraints to correctly model the real world, but there could be constraints, as dependencies, that are not imposed in the design phase, but that are already satisfied by the data itself.
A broad theory of dependencies has been developed for the relational model. In a **Functional Dependency** the *values of some attributes of a tuple uniquely or functionally determine the values of other attributes of that tuple.*

A Functional Dependency

$$R : X \to Y$$

can be simply denoted by $X \to Y$, and it stands for *'X uniquely determines Y'*, where X and Y are attribute sets. X is called antecedent or left-hand-side (LHS) and Y consequent or right-hand-side (RHS). A relation I satisfies a functional dependency $X \to Y$ (i.e. $I \models X \to Y$) if for each pair s,t of tuples in I,

$$\pi_X(s) = \pi_X(t) \ \textit{implies}\ \pi_Y(s) = \pi_Y(t)$$

where π is a *"vertical" operator* that can be used to delete and/or permutate columns of a relation. In particular:

$$\pi_{x_1,\ldots,x_n}(I) = \{\langle t(x_1), ..t(x_n)\rangle \mid t \in I\}.$$

Given table 3.1, we provide an example of Functional Dependency:

$$(Wind, -), (Temperature, -), (Outlook, -) \to (Play, -)$$

that could be written as: *"given the attributes Wind, Temperature and Outlook and their corresponding values, we can uniquely determine the attribute value of Play".*

The constraints that Functional Dependencies (for brevity FDs) impose are often too strict for real world datasets since they must hold for all the values of the attribute sets X and Y.
For this reason, researchers have begun to study generalizations of FDs, called Relaxed Functional Dependencies, which relax one or more constraints of canonical FDs.

3.2.1 Relaxed Functional Dependencies

Relaxed Functional Dependencies, or RFDs, are *FDs where some constraints are deleted.*

To have an overview of the possible types of Functional Dependencies, it is fundamental the work *"Relaxed Functional Dependencies - A Survey of Approaches"* by Caruccio, Deufemia and Polese [5] in which the authors presented and enumerate the possible types of Relaxed Functional Dependencies (or RFDs).

The authors analyzed 35 different RFDs studying the application domain and their characteristics.
There are two main categories on which Functional Dependencies can be relaxed: on **extend** or on **attribute comparison** [5].

Relaxing on the **extent parameter** means that the FD holds on *"almost" all tuples or on a subset of them.* In fact, FDs might not hold for some tuples due to errors, missing values, or different data formats, but also to the possibility for some application domains to admit outliers. An example of this type of dependencies is *Approximate Functional Dependencies.*

Relaxing FDs on the **attribute comparison** means using *approximate matching paradigms to compare the attribute values* on the Left Hand Side of the rule (LHS) and the *implied attribute values* on the Right Hand Side of the rule (RHS). An example of this type of dependencies is *Conditional Functional Dependencies.*

We cite (among others) four types of Relaxed Functional Dependencies that are interesting for our research:

- *Approximate Functional Dependencies* (AFDs) which are FDs which almost hold on a given relation,
- *Conditional Functional Dependencies* (CFDs) that are dependencies which holds on instances of the relations,
- *Approximate Conditional Functional Dependencies* (ACFDs) that are dependencies that combine together AFDs and CFDs,
- *Association Rules* (ARs) which are dependencies almost holding on particular values of attributes.

For our research, we are interested in these types of dependencies, so in next sections, we will present them with formal definitions, analyzing more in deep and proposing related examples. At the end of each section, for each type of dependency, we will also suggest the main algorithms that find each specific type of dependency.

3.2.2 Approximated Functional Dependencies

Approximate Functional Dependencies (for brevity AFDs) are *FDs holding on almost all tuples* [5, p.5].

In order to quantify how an AFD "almost" holds, several measures have been proposed and studied [16], such as the *g3 error* measure or an *information dependency* measure.
In our research we focused on **g3 error**, that is defined as *"the (normalized) minimum number of tuples that need to be removed from a relation instance R in order for an FD to hold"* [5, p.6].
Note that, normalized g3 error ranges from 0 to 1 and it is equal to zero exactly when the rule is a FD.

An example related to table 3.1:

$$(Outlook, -), (Wind, -) \rightarrow (Play, -)$$

In fact, for almost all values of the attributes *Wind* and *Outlook* it determines the value of attribute *Play*, but this is not true, for example, for:

$$(Wind, false), (Outlook, sunny) \rightarrow (Play, -)$$

because the value of attribute *Play* in two cases is *don't* and in one case is *play*.

With this type of dependencies we are able to catch more interesting patterns in data, especially the ones that are not *"valid"* for all tuples.
Many algorithms have been designed to find **Functional Dependencies**, and in particular, **Approximate Functional Dependencies**.

- More than 20 years ago, Huhtala et al. [18] proposed *TANE*, an algorithm that finds Functional Dependencies based on a level-wise research of rules through a continuous pruning from an initial set of attributes.
- Wyss et al. proposed a different type of algorithm called *FastFD* [41]. This method was based on a depth-first, heuristic-driven research instead of a level-wise research.

These algorithms can be relaxed on the *extent*, so that they can find both FDs and AFDs. Throw the setting of the *g3 error*, that in practise is the confidence level, the programs mine the data to find the relaxed rules.

3.2.3 Conditional Functional Dependencies

Conditional Functional Dependencies (or CFDs) are *a specific type of FDs that use conditions to specify the subset of tuples on which a dependency holds.*
In particular, a CFD is a pair $(X \rightarrow A, t_p)$, where:

- X is a set of attributes in A
- $X \rightarrow A$ is a standard functional dependency
- t_p is a *pattern tuple* with attributes in X and A, where for each B in $X \cup A$, $t_p[B]$ is a constant *'b'* in *dom(B)*, or an unnamed variable '-'.

If a CFD $(X \rightarrow A, t_p)$, has $t_p[A] =$ '-' is called *variable*, otherwise is *constant.*
With this type of dependencies we can catch particular and concrete patterns in the dataset, in fact we are able to analyze precise values of the tuples and be more specific.

An example related to table 3.1 is:

$$(Wind, false), (Outlook, rain) \rightarrow (Play, don't)$$

that can be interpreted as: *"when attribute Wind has value false and attribute Outlook has value rain the attribute value of Play is don't".*

The first algorithms developed for the discovery of Functional Dependencies were the basis for many future works involving the research of Conditional Functional Dependencies.
There are different implementations of CFDs discovery algorithms and many of them are based on the research of Fan et al. [13].
Mainly three algorithms of CFD discovery are adopted:

- *CFDMiner*, which focuses especially on the research of constant CFDs through a mining of closed itemsets. Given a relation R, an itemset is defined as a pair (X, t_p), where X is a set of attributes from R and t_p is a constant pattern over X containing only constant values.
  An itemset (X, t_p) is closed if there exists no more general itemset

(Y, s_p) such that $Y \subseteq X$ and with the same support as (X, t_p) for a certain instance r of R. The support of an itemset (X, t_p) for a certain instance r is denoted as supp (X, t_p, r), and it represents the number of tuples that match with t_p on the set X.

- *CTane*, a level-wise extension of *TANE* for the research of general CFDs. It takes advantage of k-frequent itemsets, which are itemsets whose support is greater than k. In this case, the pattern t_p can include also the unnamed variable '-'.

- *FastCFD*, an extension of *FastFD* which keeps its depth-first research, but also mines closed itemsets. As *CTane*, it leverages k-frequent itemsets.

3.2.4 Approximate Conditional Functional Dependencies

Approximate Conditional Functional Dependencies called also ACFDs, *are Functional Dependencies that relax on both criteria: the extend and the attribute comparison.*
It is Conditional Functional Dependency, as defined before, but that relax also on the extend, so the *g3 error* could be less than 1.

An example taken from table 3.1 could be:

$$(Wind, false), (Humidity, normal) \rightarrow (Play, play)$$

that can be interpreted as: *"when attribute Wind has value false and attribute Humidity has value normal the attribute value of Play is play if we delete a maximum number of tuples N".*
Thus, this rule specifies the attribute values on which it holds and it is true for all tuples less that N tuples in the dataset.

ACFDs represent the most general type of FDs and they will be the most important tool for our research because they catch the most interesting aspects in a dataset. Unifying these two relaxation criteria, we can detect specific and not exact rules that can show anomalies or unexpected patterns in the database.

More difficult is to find algorithms that implement both relaxation criteria on *extent* and contemporary on *attribute comparison.* In fact, finding Approximate Conditional Functional Dependencies is more complex than searching FDs or CFDs because ACFDs relax on both criteria simultaneously allowing

a less number of constraints contemporary, increasing the problem complexity.

An algorithm created to discover ACFDs is *CFDDiscovery* [34].
It was created for database cleaning and repair, and it can relax on both criteria. It is based on the previous AFDs and CFDs discovery algorithms, in fact it supports 10 distinct approaches, belonging to three different general methodologies used in other works:

- **Integrated**, uses a depth-first implementation of the *CTane* algorithm
- **Itemset-First**, uses a breadth-first version of *Eclat* for the itemset mining step and a depth-first *TANE* implementation for the FD discovery step
- **FD-First** uses both a depth-first *TANE* step and depth-first itemset mining.

In next section, we will present Association Rules that are another type of dependencies, but that belong to the Data Mining area.

3.3 Data Mining and Association Rules

Data Mining is the non-trivial process of discovering *valid, novel, potentially useful* and *understandable patterns* in large data sets involving several different methods as classification, regression, association rule mining, clustering etc [39].
It emerged in the late 1980s and it is based on Statistics, Machine Learning and Database Technology. Data Mining has evolved over time thanks to Big Data and to the pressing needs for the automated analysis of massive data.

Data Mining tasks need an input data to start the process, usually in tabular form. If a data mining task uses a relational table as for example, table 3.1, the tuples are also called instances, the attributes are also called variables and the last attribute is usually named *class* or *target attribute* (e.g. *Play*). It is a column that has a special content inside the data, because it has a symbolic meaning.

The Data Mining tasks that are strictly correlated to our work are Classification and Association Rule Mining [39].
Classification is the *problem of identifying to which of a set of categories a new observation belongs, on the basis of a set of data containing observations*

or instances, whose category membership is known.
There are many interesting studies in classification that are related to our research and we will analyze them in later chapters.

In **Association Rule Mining**, the goal, *given a set of transactions, is to find rules that predict the occurrence of an item based on the occurrences of other items in the transaction.* Given a data structure the process finds frequent patterns, associations, correlations or causal structure among the sets of objects in the dataset.
The rule found are in the form: $X \rightarrow Y$ with X, Y defined in the same way as for Functional Dependencies.

Thus, Approximate Conditional Functional Dependencies are present also in the context of Data Mining and they are called **Association Rules**. ACFDs can be seen as a kind of Association Rules. In particular, Association Rules and Approximate Conditional Functional Dependencies are two different concepts that came from different areas (the first came from Data Mining and the second was born in the Database field) but, practically, finding ARs means finding ACFDs and vice-versa. An in-depth discussion that connects the notions FDs, CFDs, and ARs can be found in [27]. The authors show that all those dependencies are indeed structurally the same and can be unified into a *single hierarchy of dependencies.*
For this reason we will use the term *rule* as synonym of *dependency*, aware about the conceptual difference and conscious of the concrete sameness.

A benefit of this hierarchy is that existing algorithms which discover ARs could be adapted to discover any kind of dependencies and, moreover, generate a reduced set of dependencies. To discover **Association Rules** there are mainly three different algorithms:

- *A-priori* uses breadth-first search and proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database.

- *Eclat* is a depth-first search algorithm based on set intersection. It works on the same principle of A-priori, but it optimizes the computation of the support by using a vertical database that makes more efficient the frequency computation.

- *FP-growth*, where FP stands for frequent pattern. The algorithm starts

compressing a large database into a compact, FP-tree structure avoiding costly database scan. It uses a divide-and-conquer methodology avoiding candidates generation.

Using these algorithms we can also discover *ACFDs*.

Knowledge discovery and data mining, database management, reverse engineering and query optimization are among the main applications benefiting from efficient dependencies discovery algorithms.
It is interesting that, from our searches, none of these presented algorithms, and in particular, the Functional Dependencies, are used to discovery bias or discrimination in data. The main function for which FDs and RFDs are discovered is for Data Cleaning and Data Repair.

In the next paragraph, we will specify the *metrics* used for Association Rules that are also measures used to analyze Functional Dependencies.

3.4 Evaluation Metrics

In this section, we provide the definitions of the main evaluation metrics used to evaluate Functional Dependencies and Association Rules.
The most common ones are Support, Confidence and Lift.

- **Support**

$$Support(X \to Y) = sup(X, Y) = \frac{\#(X, Y)}{all - tuples}$$

where $\#(X, Y)$ is the count of how many times X and Y appears together in the dataset and *all-tuples* is the total number of tuples in the table.
The support is the percentage of the samples in the dataset that verifies the dependency $X \to Y$ and it can assume values in the range [0, 1].
We can notice that support is a *symmetric relationship*, in fact the support of $X \to Y$ is equal to $Y \to X$.
This metric allows the user to take advantage of her domain knowledge about the dataset dimensions and the cardinality of its attributes in order to define how many samples should be seen in order to consider a dependency as a valid one. This is highly application specific, and it usually used minimum support threshold of $\frac{2}{\#samples}$ and it can be adapted according to the application needs.

- **Confidence**

$$Confidence(X \rightarrow Y) = conf(X, Y) = \frac{sup(X, Y)}{sup(X)}$$

It shows how frequently the dependency $(X \rightarrow Y)$ is verified knowing that the rule body X is verified. It can assume values in the range [0, 1].
If we want only exact and valid rules the confidence must be set to 1. If we are interested in rules that are not valid on all dataset, so we want ot relax on the extend, we can decrease the confidence keeping attention to the fact that the higher this value is, higher is the probability that the dependency is verified.
The confidence can be seen as the same concept of the *g3 error* previously defined.

- **Lift**

$$Lift(X \rightarrow Y) = \frac{sup(X, Y)}{sup(X) * sup(Y)}$$

Lift is the ratio of the observed support of $X \rightarrow Y$ to that expected if X and Y were independent. This metric allowed us to take into account also the degree of independence of the components of the rules and it can assume values in the range $[0, +\infty]$.
Lift values similar to 1 suggest that X and Y are independent, while lift values greater than 1 suggest that X and Y are dependent. Our desired case is the second one since we want to find highly correlated instances.
Furthermore, lift is a *symmetric relationship*, so it considers only statistical co-occurrence of the components and does not provide any information about the actual implication between them.

Given the basis of Functional Dependencies, in next section we will focus on the main studies done in Data Science Ethics that are related to our framework.

Chapter 4

Literature Review

In this section we start presenting a comprehensive summary of all the fairness definitions in classification context. We will continue analyzing previous researches done in the area of Data Science Ethics, especially in the Machine Learning context. Finally, we give an overview of the most important works and applications that solve the problem of "unfair dataset".

4.1 Machine Learning and Ethics

As already said in Chapter 2, applications nowadays need the discovery of bias in data to avoid unintentional behaviours and unethical consequences, so the presence technologies that detect biases and accurately discover discrimination to obtain fair databases is more and more necessary. For this reason, we present the main researches done in Data Science Ethics, in particular in Machine Learning area.

Machine Learning is *the study of computer algorithms that improve automatically through experience.* In this huge area there are a variety of tasks that can be performed as model description or prediction.
In this context, the necessity of Data Science Ethics arises to avoid bias and discrimination in Machine Learning tasks.
The application domain of Machine Learning is pervasive and can influence different spheres, so from real world problems algorithms received more attention also for the ethical consequences of applications.
Recent breakthroughs justice, as discrimination against black people in America, or in computer vision, specifically object recognition, reveal just how much pattern-discovery can achieve.

Machine learning aims to solve the *"fairness problem"* by abandoning the attempt to teach a computer through explicit instruction in favor of a **process of learning by example**.
However, there are serious risks in learning from examples; learning is not a process of simply committing examples to memory.
This means that reliably generalizing from historical examples to future cases requires that we provide the computer with good examples: a sufficiently large number of examples to uncover subtle patterns; a sufficiently diverse set of examples to showcase the many different types of appearances that objects might take; a sufficiently well-annotated set of examples to furnish machine learning with reliable ground truth; and so on.

A learning system needs data that are not biases, even if bias come from the world. Thus, data have to satisfy three requirements:

- **Diversity:** there be enough representation of even small protected groups so that they are not ignored.
- **Coverage:** is simply the count of elements in a given group. It is another mathematical definition of diversity and it is used when there are multiple criteria on which fairness should be guaranteed.
- **Stability:** it ensures that the results will not change if the model parameters are changed slightly. Without stability the results can be very different with small changes to model parameters.

Thus, evidence-based decision-making is only as reliable as the evidence on which it is based, and high quality examples are critically important to Machine Learning [3]. The fact that machine learning is *"evidence-based"* by no means ensures that it will lead to accurate, reliable, or fair decisions.

4.2 Fairness notions in Classification

In machine learning, defining Fairness is strictly related to a Classification task. **Classification**, as already said in Chapter 2, is *the problem of identifying to which of a set of categories a new observation belongs, on the basis of a set of data containing observations or instances, whose category membership is known.*

In the context of binary classification task (that means that there are only two possible classes: positive/negative) there are various *Fairness* definitions.

The possible definitions are presented in *"Fairness Definitions Explained"* by Verma and Rubin. [40].
The notions are based on **statistical measures** in which the *algorithm imports the data, then it computes a classifier, it associates the class prediction of an unseen value and it compares the result with the actual value.*
The first step consists in computing the *Confusion Matrix* that is a table composed by four indicators:

- **True Positive:** for brevity *TP*, the amount of data in which the predicted value and the actual value are positive.
- **False Positive:** for brevity *FP*, the amount of data in which the predicted class is positive, but the actual one is negative.
- **False Negative:** for brevity *FN*, the amount of data in which the predicted class is negative, but the actual one is positive.
- **True Negative:** for brevity *TN*, the amount of data in which the predicted value and the actual value are negative.

From this table, we can combine these numbers to obtain some interesting measure, the most used are:

- *TPR* or **True Positive Rate**, that indicates the fraction of positive cases correctly predicted to be in the positive class out of all actual positive cases; $TPR = \frac{TP}{TP+FN}$.
- *FPR* or **False Positive Rate**, the fraction of negative cases incorrectly predicted to be in the positive class out of all actual negative; $FPR = \frac{FP}{FP+TN}$.
- *TNR* or **True Negative Rate**, the fraction of negative cases correctly predicted to be in the negative class out of all actual negative; $TNR = \frac{TN}{FP+TN}$.
- *FNR* or **False Negative Rate**, the fraction of positive cases incorrectly predicted to be in the negative class out of all actual positive cases $FNR = \frac{FN}{FN+TP}$.

Next, we list the **Statistical definitions** of fairness that are based on these metrics. Fairness definitions can be based on three different criteria [40]:

- **Definitions based on Predicted Outcome:** they represent the simplest and most intuitive notion of fairness as the *Group Fairness* and *Conditional Statistical Parity.*
  A classifier satisfies Group Fairness if subjects in both protected and unprotected groups have equal probability of being assigned to the positive predicted class.
  Conditional Statistical Parity extends the previous one by permitting a set of legitimate attributes to affect the outcome.

- **Definitions based on Predicted and Actual Outcomes:** they not only consider the predicted outcome, but also compare it to the actual outcome recorded in the dataset. Some examples could be: *Predictive Parity* and *Equalized Odds.* A classifier satisfies Predictive Parity if both protected and unprotected groups have equal probability of a subject with positive predictive value to truly belong to the positive class.
  A classifier satisfies Equalized Odds if protected and unprotected groups have equal *TPR* and equal *FPR.*

- **Definitions based on Predicted Probabilities and Actual Outcomes:** they are used when the classifier is probabilistic, that means that the prediction is a probabilistic score. An example is *Test-Fairness.* A classifier satisfies Test-Fairness if for any predicted probability score S, subjects in both protected and unprotected groups have equal probability to truly belong to the positive class. This definition is similar to predictive parity except that it considers the fraction of correct positive predictions for any value of S.

There are also **Similarity-Based measures** [40], that take into account not only the sensitive attribute as in previous definitions, but all the attributes of the classified subject. Some of this notions are: *Causal discrimination, Fairness through unawareness* and *Fairness through awareness.*

The Data Science Ethics studies done in Machine Learning are based on these fairness definitions, so in the next paragraph we present the main works and framework to solve the fairness problem.

4.3 Machine Learning studies for Data Science

In general, in Machine Learning area, three are the possible branches of techniques that try to avoid unfairness:

- **Preprocessing techniques:** procedures that before the application of the algorithm make data fair.

- **Inprocessing techniques:** that are methods used during the algorithm to detect and avoid bias.

- **Postprocessing techniques:** in this branch there are all the sequences of actions needed to correct the obtained results in order to be fair.

The first category of methods is the more similar to the scope of Functional Dependencies research that aims to display biases in data. Thus, we concentrate in this huge branch, in which the main techniques, are mainly developed for classification task.
The main preprocessing methods are:

- **Suppression of Sensitive Attribute:** remove the attributes that highly correlate with sensitive attribute to avoid discriminatory results.

- **Massaging the dataset:** change the label of some tuples in the data in order to minimize the discrimination. For example, change the target label of some data in the minority group to avoid biases, or change a target label from the majority group to create fairness.

- **Reweighting:** instead of changing the labels, each tuple in the data is assigned with a weight.

- **Resampling:** calculate the sample size for each group-value combination.

All these data transformations done in preprocessing can control discrimination and limit the distortion in data.
A paper that presents the implementation of most of these techniques [4] by Bellamy et Al. in which they introduce a new open source toolkit to reach algorithmic fairness. In their work, there are techniques that belong to all branches; in the preprocessing methods, there are four different algorithms implemented in their toolkit:

- **Reweighing** generates weights in each (group, label) combination to ensure fairness before classification [23].

- **Optimized Preprocessing** learns a probabilistic transformation that edits the features and the labels to ensure fairness [11].

- **Learning Fair representations** finds a latent representation that encodes the data well but obfuscates information about protected attributes [24].
- **Disparate Impact Remover** edits feature values to increase group fairness while preserving rank-ordering within groups [14] (also this one is used manly in classification).

This platform enables experiment over the dataset and allows to contribute with other algorithms and dataset.
There are a lot of algorithms and metrics among which the user can choose. All the methods in the paper do not require user interaction, so the software repairs the "unfair dataset" and the user retrieves the dataset without bias according to the chosen fairness definition.

Another interesting work on fairness in Machine Learning is by Stoyanovich and Howe [37]. The authors proposed to develop interpretability and transparency tools based on the concept of a **Nutritional Labels**, drawing an analogy to the food industry, where simple, standard labels convey information about the ingredients and production processes. Nutritional labels are derived automatically or semi-automatically as part of the complex process that gave rise to the data or model they describe, embodying the paradigm of interpretability-by-design.
Fairness measures are based on a generative process for rankings that meet a particular fairness criterion and are drawn from a dataset with a given proportion of members of a binary protected group. The final system is called **Ranking Facts** and it automatically derives nutritional labels for rankings.

As said before, the majority of the notions are related to classification task and they can be computed only with a classifier previously built on data. The main difference between these approaches and our research is that our policy does not need a classifier, because it is based on finding constraints that are already present in the dataset. Furthermore, building a classifier to solve the fairness problem imposes to the policy to be application oriented and need a choice over the possible fairness definitions.

In next section we introduce our framework based on the discovery of various kinds of Functional Dependencies. Without being application oriented, our methodology will lead to detect bias already existing in data with a list of dependencies that show possible discrimination.

Chapter 5

Methodology

5.1 Introduction

In previous sections we have analyzed the area of Data Science Ethics and in particular the Machine Learning studies in the solution of "unfair datasets". Our research wants to use a particular type of constraints, that are Functional Dependencies, to give a general overview of the data, showing, if present, bias and discrimination. Our system needs, as input, only the dataset that we want to analyze, then, the framework is applied as a technique to obtain a list of unethical dependencies ordered by importance.
In this section, we will provide a description of the methodology that we followed in order to develop our tool to discover already existing bias in data. For this reason, we introduce the framework and we apply it to a running example showing the obtained results. Finally, we focus on two different studies in which we present a practical unified flow of the framework with two different datasets.

5.2 Our framework

Our framework is called ***FAIR-DB*** that stands for ***F****unction****A****l Depen-denc****I****es to discove****R*** ***D****ata* ***B****ias*. It is based on the discovery and analysis of Approximate Conditional Functional Dependencies or ACFDs, that are Functional Dependencies (FDs) that hold on almost all tuples, and that use conditions to specify the subset of tuples on which they hold.

As already said in chap. 3.3, ACFDs can be seen as a kind of Association Rules. In particular, Association Rules and Approximate Conditional Functional Dependencies are two different concepts that came from different

areas (the first came from Data Mining and the second was born in the Database field) but, practically, finding ARs means finding ACFDs and vice-versa. For this reason, we will use the term *rule* as synonym of *dependency*, aware about the conceptual difference and conscious of the concrete sameness.

Figure 5.1 represents the workflow with the main phases. The framework is divided in six main steps:

1. **Data Preparation and Exploration:** in this phase we import the data, we perform data integration and we apply all the data preprocessing steps needed (solve missing values, apply discretization etc) to clean and prepare the data. During this phase, we also visualize the attribute distributions using different *Data Visualization* techniques.

2. **CFDDiscovery**: taken the prepared dataset and fixed the input parameters that are support, confidence and a maximum size, we apply *CFDDiscovery* algorithm (presented in chap. 3.2.4) to extract the dependencies from the dataset.

3. **ACFDs Filtering:** from the output of *CFDDiscovery*, we discard the Approximate Functional Dependencies, for brevity AFDs, from the output list and we create a dictionary for ACFDs. In this phase, we apply also other constraints and we rewrite the rules into a more easy and understandable form to optimize the future computations.

4. **ACFDs Selection:** for each rule, we compute the metrics that show the ethical aspect of the dependency, in particular the *Difference* metric and for each protected attribute p we compute the *p-Difference*. According to values of the metrics, we select the most interesting rules.

5. **ACFDs Completion:** given the list of interesting rules, for each one of them, we compute all the possible combinations over the protected attributes and class variable; this process may add also rules that do not show bias.

6. **ACFDs Ranking:** analyzing all the resulting rules, we select the ones that show the unethical aspect. After that, we build a ranking composed by the ACFDs put in descending order of importance with their relative metrics.

7. **ACFDs User Selection and Scoring:** from the ranking, the user selects N ACFDs, then, we present the ranked selected dependencies

and the summarizing metrics that give a final score about the unfairness of the dataset.

Obviously according to the obtained results the process can restart tuning differently the parameters and selecting other rules, so that the user can restart the process until she is satisfied with the results.
In next paragraphs we will describe every phase more in depth explaining the details of our framework, in particular, for each step we apply it to the running example.

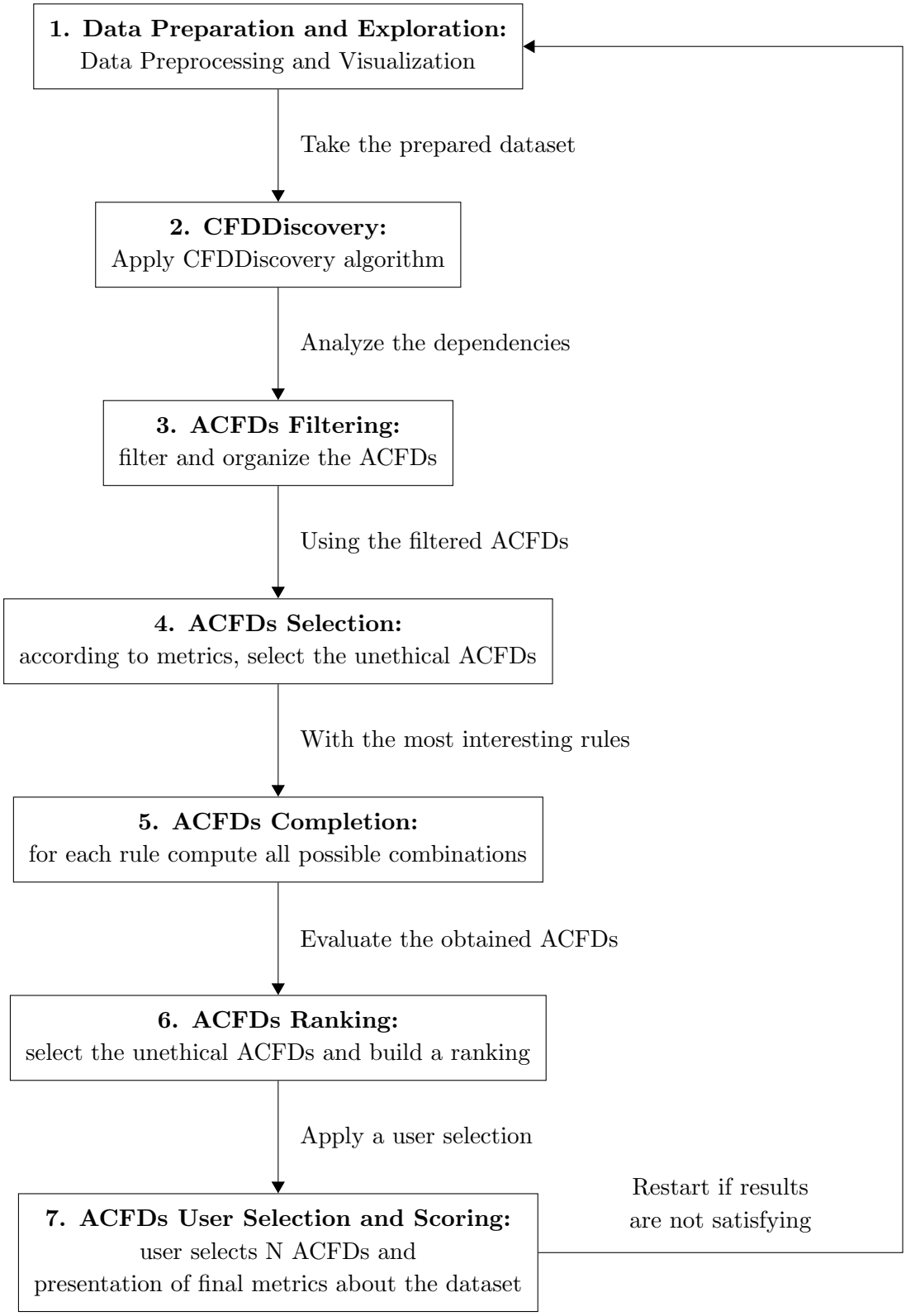


Figure 5.1: FAIR-DB schema

5.3 Running Example: U.S. Census Adult Dataset

Firstly, we describe the dataset that we used and that is significant for our research. We apply preprocessing operations to prepare the dataset accordingly to domain knowledge and optimization, then we apply the steps of our framework to extract the various types of Functional Dependencies to recognize bias and discriminatory behaviours.

The running example that we used is **UCI Machine Learning Repository Adult**[1] [12] a dataset containing the census information of U.S. people. This dataset relates people income to social factors such as age, education, race etc.
The *U.S. Census Adult Income* dataset was extracted by Barry Becker from the 1994 U.S. Census Database. The dataset consists of anonymous information such as occupation, age, native country, race, capital gain, capital loss, education, workclass and more.
The version that we considered contains 32561 tuples and 13 attributes, 5 of which are numerical and 8 are categorical, including the target variable *'Income'*. The main task for which this dataset is used for is to predict if a person earns more or less than 50,000 dollars/year, based on other features like the degree of education and the age.

We now provide a brief description of the attributes. We highlight that we refer to the values actually contained in this specific dataset, without any further assumption.

- **Age:** a numerical variable representing the age of a person. The values belong to the range [17,90].
- **Workclass:** a categorical variable representing the workclass of a person. Examples are *Private* and *Federal-gov*.
- **Education:** a categorical variable representing the educational degree. Examples are *HS-grad*, *Bachelors* and *Doctorate*. There are 16 distinct values.
- **Education-num:** a numerical variable representing a numeric encoding of the attribute education. The values range from 1 to 16 and examples of encoding with education are *Bachelors: 13* and *HS-grad: 9*.

[1] https://archive.ics.uci.edu/ml/datasets/Adult

- **Marital-status:** A categorical variable representing the marital status of a person. Examples are *Married-civ-spouse* and *Divorced*. There are 7 distinct values.

- **Occupation:** a categorical variable representing the type of job. Examples are *Sales* and *Tech-support*. There are 14 distinct values.

- **Relationship:** a categorical variable representing the family status. Examples are *Husband, Wife* and *Not-in-family*. There are 6 distinct values.

- **Race:** a categorical variable representing the race of a person. Examples are *White* and *Black*. There are 5 distinct values.

- **Sex:** a binary categorical variable representing the sex of a person. There are only the values *Female* and *Male*.

- **Capital-gain:** a numerical variable representing the capital gain of a person. It is the profit one earns on the sale of an asset like stocks, bonds or real estate. The values belong to the range [0, 99999]. The value 0 is the most frequent (95.27%).

- **Capital-loss:** a numerical variable representing the capital loss of a person. It arises when the cost price of stocks, bonds or real estate is higher than the selling price. The values belong to the range [0, 4356]. The value 0 is the most frequent (91.59%).

- **Hours-per-week:** a numerical variable representing how many hours a person work during each week. The values belong to the range [1, 99].

- **Native-country:** a categorical variable representing the country of origin. The most frequent value is *United-States* (91.19%), examples of other values are *Mexico, Philippines* and *Germany*. There are 41 distinct values.

- **Income:** a binary categorical variable representing if the annual income of a person is greater or less than 50'000 dollars/year. This is the target variable of the problem and it can assume the following values: $> 50K$ and $\leq 50K$.

Figure 5.2 presents the first five tuples of the U.S. Census Adult dataset.

Figure 5.2: U.S. Census Adult dataset

	age	workclass	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	90	?	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356	40	United-States	<=50K
1	82	Private	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18	United-States	<=50K
2	66	?	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356	40	United-States	<=50K
3	54	Private	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	40	United-States	<=50K
4	41	Private	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	40	United-States	<=50K

5.3.1 Data Preparation and Exploration

Firstly, Data Preparation starts with **Data Acquisition**, in this phase the user gathers one or more datasets that she will use in the research.
After Data Acquisition, **Summary Statistics** operation is usually performed. Summary statistics consist in *quantities, as mean for example, that capture various characteristics of a potentially large set of values with a single number or a small set of numbers*[39, p.98]. Summary Statistics are usually computed for the attributes of the dataset, for each of them we compute the relevant statistics presented in tabular form. This primary phase gives a *general idea* of the dataset and during this step, we can hypothesize the *protected columns* and we can identify (if present) the *target variable*.

Data Cleaning, that means solving data errors, is an operation that should be performed as soon as data is available, follows before any integration procedure. During the development of our framework, we considered the following assumptions:

- The user needs to clean a single data source.
- The dataset is in a relational form.

After Data Cleaning phase we apply **Data Integration**. In this step, multiple datasets are merged into a single one. It is important to notice that poor data quality from the singular sources will lead to poor quality even in the final dataset. Moreover, the user should be aware that Data Integration, also when the singular sources are clean, can lead to errors in the final dataset.

Data Preparation is a very important phase before performing any algorithm or application, in particular *Data Cleaning*. Poor data quality could lead to wrong or incomplete results. It is very unlikely to automatically resolve all the conflicts within a dataset. This is due to the fact that several

types of errors may be present and it is difficult software wise to correctly recognize all of them.

In [30], **Data Errors** are classified under three main categories:

- **Syntactical Anomalies**: they involve the format and the values which are used in order to represent real entities. In this category we find *Lexical Errors, Domain Format Errors* and *Irregularities*.
- **Semantic Anomalies**: they limit the capability of the data samples of being a comprehensive representation of the real world. This category includes *Integrity Constraints Violations, Contradictions, Duplicates* and *Invalid Tuples*.
- **Coverage Anomalies**: they limit the amount of information that can be represented in the recorded data. Errors of this type are *Missing Values* and *Missing Tuples*.

Depending on the dataset that we are considering, we may need to face several different error types. On top of that, the type of the error strongly influences the cleaning procedure.
We decided to solve primarily the problem of *missing values* because they can be easily identified and the loss of information that they introduce leads to worse results for the analysis which is being carried out.
In general, to solve missing values data analysts use imputation, so, if the suggestion of an imputation is accurate enough the replacement is automatically performed. However, it has to be noticed that there may be missing values that are *valuable information* [30] to take into consideration for the analysis, for example a person may have refused to fill a certain field in a survey and we may wonder why she chose to do so.
The dataset that we used during this thesis contains missing values that in general do not express valuable information, but there could be other cases that can lead to different considerations during the workflow of the framework. Moreover, applying imputation is not a good choice in all cases, in fact, substituting missing values could may change bias in data.

After Data Cleaning and Data Integration we may need to perform other steps as: *Feature Selection*, *Data reduction* and *Discretization* [39].

Feature selection refers to choosing the set of features to use that is appropriate for the subsequent analysis. The goal of feature selection is to

come up with the *smallest set of features* that best captures the characteristics of the problem being addressed [39]. The smaller the number of features used, the simpler the analysis will be. Of course, the set of features used must include all features relevant to the problem. So, there must be a balance between *expressiveness* and *compactness* of the feature set. There are several methods to consider when selecting features: new features can be added, some features can be removed, features can be re-coded or features can be combined. All these operations affect the final set of features that will be used for analysis. Of course, some features can be kept as is as well or new features can be derived from existing features.
Irrelevant features are those that contain no information that is useful for the analysis task and they should be removed. Good understanding of the domain application is essential in deciding which features to add, drop or modify.

A dataset could have a large number of attributes: this is what is called **highly dimensional data** [39]. Most of these dimensions may or may not matter in the context of our application with the questions we are asking. Reducing such high dimensions to a more manageable set of related and useful variables improves the *performance* and *accuracy* of our analysis. As the dimensionality increases, the complexity increases. As the space grows, data becomes increasingly sparse, this is called *Curse of Dimensionality*. To avoid the curse of dimensionality, you want to reduce the dimensionality of your data. Reducing the dimensionality means finding a smaller subset of features that can effectively capture the characteristics of your data.
To apply dimensionality reduction there are many techniques, the most famous are *Principal Component Analysis, t-Distributed Stochastic Neighbor Embedding* and *Singular Value Decomposition*. All these methods are useful to reduce the dimensionality of your data, but they can also make the resulting analysis models more difficult to interpret. The original features in the dataset have specific meanings such as *income, age* and *occupation*. By mapping the data to a new coordinate system, the dimensions in transformed data no longer have natural meanings. For this reason, we suggest to not apply dimensionality reduction, but instead perform a *manual feature elimination* to reduce dimensionality.

Another important step is **Discretization**, that means transforming data *from numeric to nominal data type* [39, p.57]. There are important effects of discretization among which smooth things of data, reducing noise, reducing data size etc.
For our purpose, Discretization is necessary in many cases, because Functional

Dependencies built on numerical attributes that have many different values are very precise, but they do not give an overview over the attribute values. For example, if there is a rule that contains a specific value of attribute *'age'* (let us assume that it is 18) we do not know if the rule could be valid also for values near to 18, like 20 or 16; furthermore, the rule could be not useful if involves only 18-years old people. For this reason, we suggest to use Discretization for numerical attributes that have many different values.
Discretization methods include *manual methods* or supervised and *automatic methods* or unsupervised [39]. If we have some domain knowledge is better to use the supervised approach in which we create the bins accordingly to what we know. If we do not have any domain knowledge, we can use the previously plotted distribution of the attributes values to create the bins. *Equal-width binning, equal-depth binning, regression analysis, cluster analysis, natural partitioning* would all be possible methods for discretization.

- For **equal-width binning**, given a range of values min, max, we divide in intervals of approximately same width.
- The **equal-depth binning** refers to having the same number of rows in each bin. Thus, given again a range of values min and max, we place, approximately, the same number of instances, in each bin, by dividing the total number of samples, by the desired number of samples in each bin.

Once you have the bins, if you want to **smooth your data**, you can *replace the values in a bin by statistical indicator*, such as the average or the median for numerical data and, for categorical data, we can use the mode. We will not use this technique that may eliminate existing bias, that are our main focus.
When we have created the different bins for the variables, it could be interesting to apply clustering to original data and afterwards decision trees to each cluster to analyze the obtained groups. If we have a target class we can delete it from the data before clustering and use it when we apply decision trees to analyze the goodness of the obtained groups.

During of Data Preparation steps, we can go through Data Visualization [39]. **Data Visualization** is the technique of conversion of data into a *visual or tabular format*, in this way the characteristics of the data and the relationships among data items and attributes can be analyzed. Humans have a well-developed ability to analyze large amounts of information that is presented visually, so they can detect *general or unusual patterns.*

For numerical attributes the techniques that are usually adopted are **Histograms** [39]. They are a graphical representation of the distribution of data and they are representation of tabulated frequencies erected or discrete intervals. They can also show a *density function*, a curve that simply summarizes over the individuals.
Instead, to plot categorical columns **Bar Plots** are used [39]. They compare *categories* and they can also be grouped by class, so that every attribute can be plotted with the target one leading to useful considerations.
Another useful technique to plot continuous variables is **Box Plot** [39]. It is another way of displaying the distribution of data and it gives an idea of the *skeweness of the population.*
This first data exploration phase is very important because the user understands from the plots if there are groups in the dataset and if there is majority class and, if present, how many are the small groups.

Data Visualization can continue more in deep using other plots as **Heatmaps** and **Clustermaps** [39]. Both techniques are used to see more dimensions at once projecting data into a smaller space. These two methods show the *correlation* between all attributes or a specific subset of them. In particular, Clustermap is an Heatmap that adds clustering algorithm on attribute values to see which ones are more correlated using clustering.

5.3.2 Running example: Data Preparation

Before being able to gather useful insights from running example of U.S. Census Adult dataset, we have to perform some preprocessing operations as *Data Cleaning*, *Feature Selection* and *Discretization.*

Before starting, it is interesting to perform *Summary Statistics.* Figure 5.3 displays the resulting statistics applied to numerical attributes and figure 5.4 to categorical ones.
For numerical attributes the table presents:

- **Count**: that is the frequency of the attribute.
- **Mean** and **Standard Deviation**.
- **Minimum** and **Maximum values** of every attribute.
- The most interesting **Percentiles**.

From the table values we can see that *'Capital-gain'* and *'Capital-loss'* attributes are skewed and their values are very different from other variables.

For categorical variables the figure 5.4 displays four values:

- **Count**: it represents the frequency of the column.
- **Unique**: it presents how many unique values are present for every attribute.
- **Top**: it indicates the mode of the attribute.
- **Freq**: it shows how many times appears the mode of every attribute.

Looking at values of *'Race'* and *'Native-country'* we can easily detect a majority class composed by *'White'* and *'United-States'* people. Furthermore, more than an half of the dataset represents people having *'Male'* sex and also more people gain *'*$\leq 50K$*'* dollars/year.

Figure 5.3: Summary Statistics Numerical Variables

	age	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	10.080679	1077.648844	87.303830	40.437456
std	13.640433	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	48.000000	12.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	4356.000000	99.000000

Figure 5.4: Summary Statistics Categorical Variables

	workclass	education	marital-status	occupation	relationship	race	sex	native-country	income
count	32561	32561	32561	32561	32561	32561	32561	32561	32561
unique	9	16	7	15	6	5	2	42	2
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	United-States	<=50K
freq	22696	10501	14976	4140	13193	27816	21790	29170	24720

After that, we compute the **missing ratio** to understand how many columns have missing values and for each one how much it is. The columns that are not too relevant in the search of ethical bias could be deleted. The attributes with a missing ratio too high are also removed. If a column (that is not a protected attribute) has a very little missing ratio, the missing values can be imputed, for example, by substituting them with the mode if the attribute is categorical, or using the median if the attribute is numerical. We do not suggest to use imputation or other methods with protected attributes because they can change the original biases obtaining wrong results.

We noticed that three attributes contained some missing values, reported as *'?'*. In particular, there were 1836 missing values for the attribute *workclass*, 1843 for the attribute *occupation* and 583 for the attribute *native-country*, for a total of 4262 missing values (less than 1% of the total number of cells). The total number of rows, containing at least one missing value, were 2399 (about 7.4% of the total number of samples). Therefore, we decided to remove the columns that are not interesting in this research from an ethical point of view and then delete also the rows that contain missing values. The deleted columns are: *'Marital-status', 'Occupation', 'Relationship', 'Capital-gain', 'Capital-loss'* that are not meaningful from an ethical point of view. In this way, computations of Functional Dependencies become faster, results are easier to read and analyze because they will only show relations between relevant attributes discarding uninteresting patterns. Hence, the dataset that we actually used for our analysis contained 30169 samples.

We noticed that two of the columns, *'Education'* and *'Education-num'*, represented the same information. The latter was simply obtained through a numerical encoding of the first one. It was clear that a Functional Dependency linked these two attributes, therefore we decided to remove the *'Education'*.
Furthermore, we group by *'Education-num'* using 8 bins and transforming the numbers into categorical values to optimize computations and make results more readable. The bins are: *'Preschool', 'Elementary', 'MiddleSchool', 'HS-College', 'Assoc', 'Bach', 'Mast', 'Postmaster'*.

To extract Functional Dependencies that do not depend on specific values of an attribute, it is useful to group into bins the values of an attribute to obtain relevant rules. In particular we created five bins for *'Hours-per-week'* attribute that are: *'0-20', '21-40', '41-60', '61-80', '81-100'*.

Reasoning over *'Race'* and *'Native-country'* attributes is more difficult than other attributes because there are factors, as migrations, that make it more difficult.

5.3.3 Migrants study

What is **Race**?
The data on attribute *'Race'* were derived from answers to the question on race that was asked of individuals in the United States. The Census Bureau collects racial data in accordance with guidelines provided by the U.S. Office of Management and Budget (OMB[2]), and these data are based on *self-identification.*
The racial categories included in the census questionnaire generally reflect a social definition of race recognized in this country and not an attempt to define race biologically, anthropologically, or genetically. In addition, it is recognized that the categories of the race item include **racial** and **national origin** or **socio-cultural groups**. People may choose to report more than one race to indicate their racial mixture, such as "American Indian" and "White." People who identify their origin as Hispanic, Latino, or Spanish may be of any race.
OMB requires five minimum categories: White, Black or African American, American Indian or Alaska Native, Asian, and Native Hawaiian or Other Pacific Islander.
According to the U.S. Census Bureau, every race has some native country as:

- Race: *Hispanic.* Native countries: *Mexico, Cuba, Dominican Republic, El Salvador, Guatemala, Central America, Argentina, Colombia, Ecuador, Peru, South America, Spain.*

- Race: *Non-Hispanic White.* Native countries: *Canada, France, Germany, Greece, Ireland, Italy, Netherlands, Poland, Portugal, USSR, United Kingdom, Yugoslavia, Other Europe, Iran, Israel, Other Middle East, Australia.*

- Race: *Black.* Native countries: *Haiti, Jamaica, Trinidad and Tobago, Africa.*

- Race: *Asian and Pacific Islander.* Native countries: *China, India, Japan, Korea, Philippines, Other South and East Asia, Other Oceania.*

[2]https://www.census.gov/topics/population/race/about.html

So according to the list, attribute *'Race'* is strictly correlated to *'Native-country'*.

Grouping these attributes into bins is not easy, in fact there are some *measurement issues* related to the **Ambiguity of the Race** [8].
Race has and may continue to have different meanings for different groups, sometimes overlapping and sometimes not. For instance, some Hispanics, who can be of any race in the OMB classification system, identify themselves primarily by ethnic or national origin (e.g., Mexican, Cuban, or Puerto Rican). In contrast, other Hispanics consider Hispanic or Latino to be a race on a par with Black, White, Asian, and American Indian.
Thus, there is research evidence that many Latin American and Latin Caribbean immigrants who come to the United States see themselves as being of mixed origin, most commonly European and American Indian or European and African.

Furthermore, it happens what is called **Inconsistent Reporting**. Population groups and individuals *vary in their consistency of reporting race when comparing surveys across time and with each other* [8, p.34]. In particular, there has always been considerable confusion regarding the responses of Latin American and Caribbean groups to the separate race question used in the census and surveys. Many Latin Americans and Caribbeans reject the racial categories on the census, select "other race," and write in a word that to them best describes their racial identity.

To conclude, there is **no single concept of race**. Rather, race is a *complex concept, best viewed for social science purposes as a subjective social construct based on observed or ascribed characteristics that have acquired socially significant meaning*[8, p.37]. Indeed, for the purpose of measuring racial discrimination, a social–cognitive concept of race is integral to meaningful analysis. For the purpose of understanding and measuring racial discrimination, race should be viewed as a social construct that evolves over time.

Given all these considerations we decided to keep both *'Race'* and *'Native-country'* attributes because they are correlated, but not in all cases, for example the offspring of migrants have the same race of their parents, but different native country because in many cases they may be born in U.S.. To have readable and more effective Functional Dependencies, we keep the attribute *'Race'* as it is in the original dataset and we group the values of

the attribute *'Native-country'* using 4 different values:

- **NC-White**: in this value we place people coming from *United-States*.
- **NC-Hispanic**: in this value we insert people coming from *Mexico, Cuba, Dominican Republic, El Salvador, Guatemala, Other North and Central America, Colombia, Ecuador, Peru, Argentina, Other South America, Spain*.
- **NC-Non-Hisp-White**: in this value we place people coming from *Canada, France, Germany, Greece, Ireland, Italy, Netherlands, Poland, Portugal, USSR, United Kingdom, Yugoslavia, Other Europe, Iran, Israel, Other Middle East, Australia*.
- **NC-Asian-Pacific**: in this value we insert people coming from *China, India, Japan, Korea, Philippines, Other South and East Asia, Other Oceania*.

We report in figure 5.5 how dataset appears after all the preprocessing steps done in this paragraph.
After the preprocessing section, we will describe some techniques that we

Figure 5.5: Data after the preprocessing phase

	workclass	race	sex	hours-per-week	native-country	income	age-range	education-degree
0	Private	White	Female	0-20	NC-White	<=50K	75-100	HS-College
1	Private	White	Female	21-40	NC-White	<=50K	45-60	Elementary
2	Private	White	Female	21-40	NC-White	<=50K	30-45	Assoc
3	Private	White	Female	41-60	NC-White	<=50K	30-45	HS-College
4	Private	White	Male	21-40	NC-White	<=50K	30-45	MiddleSchool
5	State-gov	White	Female	0-20	NC-White	>50K	60-75	PostMaster

have used to perform Data Exploration in order to have a general overview of the values contained in the dataset and their distribution.

5.3.4 Running example: Data Exploration

After having performed the preprocessing phase that comprehends the research and the deletion of missing values and having applied the feature selection, before running our framework it could be very useful to perform Data Exploration through **Data Visualization** to our dataset.

In our example, after the preprocessing phase, we deal only with categorical attributes that are: *'workclass'*, *'race'*, *'sex'*, *'hours-per-week'*, *'native-country'*, *'age-range'*, *'education-degree'*, *'income'*. For this type of attribute it is usual to use **Bar Plots**. Bar plot can also group attribute values by class, in this dataset every attribute can be grouped by the target variable that is *'income'*.
For example, in figure 5.6 the majority of people gain less than *50K dollars/year* and the majority of people that gain this amount is composed by *"Male"* people. We can easily see that in proportion the female category gains less that the men category.
Another technique that we suggest is the usage of **Clustermaps**. In figure

Figure 5.6: Bar Plot of "Income" and "Sex" attributes

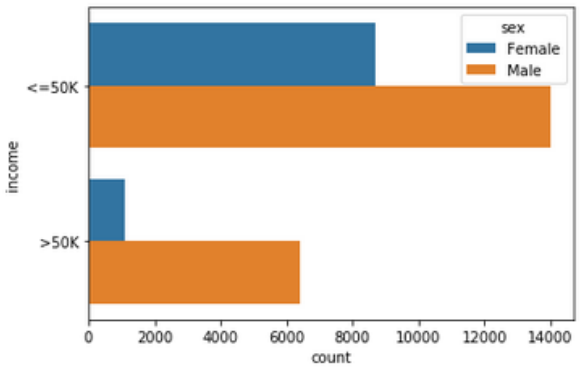
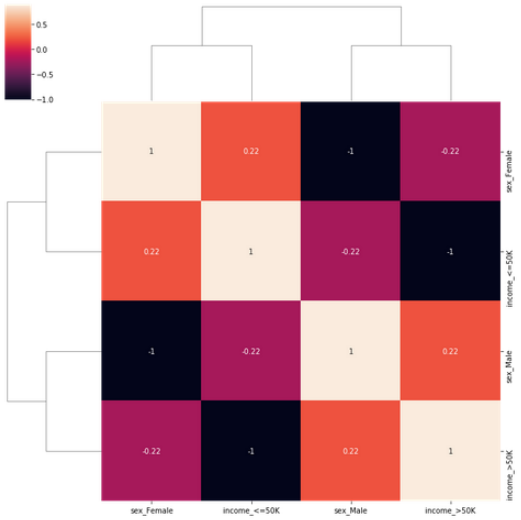


Figure 5.7: ClusterMap Plot of "Income" and "Sex" attributes



5.7 we can detect the same relationship between *Sex* and *Income* attributes that we have already seen in the Figure 5.6. The ClusterMap make the values *'Female'* and *'Income* $\leq 50K$*'* in the same cluster, and the values *'Male'* and *'Income* $> 50K$*'* in another one. This means that men gain in proportion more than women.
We suggest to apply these techniques comparing each protected attribute with the target class, in this way the user can build some ideas on the possible discrimination and bias that could be in the dataset.

5.3.5 CFDDiscovery

After data preparation, in this paragraph we will explain how to extract Approximate Conditional Functional Dependencies (ACFDs) from data using ***CFDDiscovery*** algorithm [34].
In previous section 3.2.1, we presented several algorithms to discover Association Rules (ARs) and consequently also ACFDs, as *A-priori*, *Eclat* or *FP-growth*.
We said that these algorithms could be slightly changed to discover ACFDs, but with several complications.

The algorithms that discover ARs are based on the central idea of finding *Frequent Itemsets*. An itemset is said to be frequent if its elements have support greater than a minimum support threshold. Finding correct frequent itemsets implies that all the attribute domains contain different values.
This constraint is due to the fact that, the algorithms that find ARs take into account only the value losing the reference to the attribute. Specifically, if two columns contain the same value, that value has not the same meaning in the two cases. For this reason, all the possible values of all the columns must be different one from the other.
For example, if the attribute *"Age"* has values in the domain [15-80] and attribute *"Hours-per-week"* has values in the domain [0-60], the algorithms that find ARs are not able to distinguish between the values of the two attributes, so this is true for all the values between [15-60], and this issue could generate wrong or puzzling dependencies.

We can divide the process in two steps:

1. Establish the **protected attributes** and the **target variable**.

2. Tune the parameters: **minSupport, minConfidence** and **maxSize**.

Firstly, according to the application needs, we suggest to memorize all protected attributes in a structure to save them and detect the target class. This first step is usually done also in Data Preparation and Exploration phase, but if the prepared dataset is imported, then the user has to detect and save these attributes.

Afterwards, it is central to extract the appropriate number of dependencies from the dataset, so tuning the *CFDDiscovery* parameters is crucial. The algorithm takes four mandatory arguments and one optional as input:

- A *dataset* in csv format.
- *minSupport*: the minimum support threshold, it represents the minimum number of tuples that verifies the dependency.
- *minConfidence*: the confidence leeway threshold, that represents how frequently the dependency is verified knowing that the LHS of the dependency is verified.
- *maxSize*: the maximum antecedent size of the discovered dependency, so the maximum number of attributes that can compare in LHS of the dependency.
- *[Optional]* The algorithm/implementation to be used.

Given an instance D of a schema R, support threshold δ, confidence leeway threshold ϵ, and maximum antecedent size α, the approximate CFDs discovery problem is to find all ACFDs ϕ: $(X \rightarrow Y, t_p)$ over R with:

- $\text{support}(\phi, D) \geq \delta$
- $\text{confidence}(\phi, D) \geq \epsilon$
- $|X| \geq \alpha$.

The obtained rules are in this form:

$$(lhsAttr1 = valueLhsAttr1, ..., lhsAttrN = valueLhsAttrN) \rightarrow (rhsAttr = valueRhsAttr).$$

A possible example could be:

$$(native - country = \text{“}NC - Hispanic\text{”}) \rightarrow (income = \text{“} \leqslant 50K\text{”}).$$

The algorithm prints all the dependencies that respect all the aforementioned criteria, but we report some notable considerations:

- All the rules have **only one item on the RHS** of the dependency.
- It can happen that there are dependencies that are true, but they **do not involve any protected attribute or class variable** in the rule.
- The algorithm produces both **AFDs** and **ACFDs**, so there could be dependencies in which one or more attribute values are not specified.

We have to be aware of these considerations to perform next operations, as the filtering phase and the creation a of a structure that will contain all the interesting ACFDs.

The *CFDDiscovery* step should also be guided by the Data Exploration phase. In fact, plotting every attribute separately helps the user to detect the different groups in the dataset. With the detection of the groups and their frequency, we can establish a good *minimum support threshold*. In fact, setting a minimum support threshold it is not an easy task: if it is too high we lose information about small groups, if it is too low we have too many dependencies to analyze. According to the cardinality of the groups present in the dataset, the support parameter can be set approximately as the cardinality of the smallest group that is interesting for the user. The user can repeat this step decreasing *minSupport*, then she looks at the rules printed out and stops the process when the dependencies involve the meaningful groups.

The *minConfidence* is computed as the ratio between the frequency of the dependency over the frequency of the LHS of the rule. We suggest to put this parameter very high, almost 1, and then, repeat, only this step, or all the process, decreasing it a little at every round. In this way, we obtain more and more dependencies at every round and we can stop decreasing when we are satisfied with the quality and the number of rules. Obviously the lower is this parameter, the more dependencies are generated at each round increasing the computational complexity.

The *maxSize* is the cardinality of the LHS of the dependency. This parameter should be set according to the total number of attributes in the dataset and to the desired complexity of the resulting rules. If the user wants to see the relationship between few attributes, this parameter should be set as 2 or 3. If the user is interested in the correlation between more attributes, she can increase this parameter keeping in mind that this will lead to an

higher computational complexity.

Finally, the software allows the choice of different algorithm implementations to be used to extract the dependencies from the dataset. As already said in chap. 4, there are three different approaches: Integrated, Item-First and FD-First.

According to the paper section related to the experiments [34], the FD-First option scales better when the number of tuples in the dataset increases, so it is faster with respect to other methods. Furthermore, decreasing the minimum support and increasing the maximum antecedent size, FD-First outperforms the other approaches. Thus, according to our research, we decide to take the last approach, the FD-First.

An alternative to *CFDDiscovery* algorithm could be using the technique proposed by [25], that finds rules involving interesting infrequent itemsets and also leads to the discovery of negative association rules (NARs). Small groups can be seen as infrequent itemsets because they have a small support, while negative association rules are rules in which some couples "attribute-value" can compare negated. As already explained before, dealing with Association Rules introduces some issues, so for this reason, we have chosen an algorithm that finds Functional Dependencies instead of Association Rules.

5.3.6 Running Example: Apply CFDDiscovery algorithm

In this section we apply the *CFDDiscovery* algorithm to our dataset. After Data Preparation and Exploration, we import the dataset and we save three important variables:

1. **all-tuples**: this variable contains the cardinality of the dataset. In this specific example it has value 30169.
2. **protected-attributes**: this is an array that contains all the protected attributes comparing in the dataset. For the U.S. Census Adult dataset, we identified as protected attributes: *'Race'*, *'Sex'* and *'Native-country'*.
3. **target-class**: this variable saves the target attribute of the dataset, in this specific case it is *'Income'*.

At this point, we can apply the *CFDDiscovery* algorithm, establishing the input parameters.

1. We save the path of the prepared dataset in *csv* format.

2. We fixed the *minSupport*, that is the minimum number of tuples that verifies the dependency, so that is considered valid. The algorithm needs this parameter as a count, not as a percentage. We fixed it at 900, so the minSupport has value $\frac{900}{30169} = 0.03$.

3. We fixed the *minConfidence*, that is the frequency of dependency given that the LHS of the rule is valid; it is fixed at 0.86.

4. For the algorithm option, by default, the *FD-First* implementation is chosen by the software, as it is typically the fastest.

We append an additional constraint to the command of the algorithm, that imposes to the each dependency to contain the target class (could be either on the LHS or on the RHS of the rule), in this way we can already discard the not interesting dependencies.

The algorithm, given these inputs, finds 118 dependencies from the U.S. Census Adult dataset, both AFDs and ACFDs. In figure 5.8 there are the first 12 dependencies.

Figure 5.8: CFDDiscovery output, first 12 rules found

```
Total number of dependencies found:  118

Dependency n. 0 :  (education-degree=MiddleSchool) => income=<=50K
Dependency n. 1 :  (age-range=15-30) => income=<=50K
Dependency n. 2 :  (education-degree=Bach, age-range=15-30) => income=<=50K
Dependency n. 3 :  (education-degree=MiddleSchool, age-range=15-30) => income=<=50K
Dependency n. 4 :  (education-degree=Assoc, age-range=15-30) => income=<=50K
Dependency n. 5 :  (education-degree=HS-College, age-range=15-30) => income=<=50K
Dependency n. 6 :  (native-country=NC-Hispanic) => income=<=50K
Dependency n. 7 :  (income=>50K) => native-country=NC-White
Dependency n. 8 :  (income=<=50K) => native-country=NC-White
Dependency n. 9 :  (native-country=NC-White, education-degree=MiddleSchool) => income=<=50K
Dependency n. 10 :  (education-degree, income=>50K) => native-country
Dependency n. 11 :  (income=>50K, education-degree=Mast) => native-country=NC-White
```

We can easily detect the AFDs, for example dependency number 10:

$$\phi_{10} : (education - degree, income => 50K) \rightarrow (native - country)$$

is an Approximate Functional Dependency because the rule does not specify the values of attributes *'education-degree'* and *'native-country'*. The other dependencies are ACFDs and all have only one item on the RHS of the rule. As the user can notice, there could also be rules that should be removed

from the list because they do not contain any protected attribute. We will solve all these aspects in the next section, the ACFDs Filtering.

5.3.7 ACFDs Filtering

In previous paragraph we have seen the *CFDDiscovery* output: it is a simply list of strings, so, for our research, we need a structure in which organize and save the strings as a list of dependencies.
For this reason we use a **Dictionary**, that is a collection which is unordered, changeable and indexed. Dictionaries are structures having keys and corresponding values associated to that keys.

We can divide the ACFDs Filtering step in three main phases:

1. **Create a parser**: given the list of rules, the parser splits the list extracting every dependency. Then, from each dependency we divide the LHS and the RHS of the rule. From these two parts, we extract every couple of *"attribute-value"* and we save them in a structure.

2. **Filter the ACFDs**: for each rule, we check each couple of *"attribute-value"*; if at least one value is missing, the rule is a AFD, so, we can discard it. In this way, we keep only the ACFDs. Then, for each rule, we also check if at least one protected attribute and the class variable appear. If the attributes in the dependency do not respect the constraint, we discard such dependency.

3. **Create the dictionary**: given all these ACFDs we build a dictionary that is a list of rules in which there are two distinct fields: *'lhs'* and *'rhs'*. Every field contains a list of one or more couples "attribute-value", in which each attribute is the key and the relative value associated to the key is the corresponding attribute value.

In the first step, the creation of the parser, is important to correctly interpret the list of strings, in particular extract each dependency is essential. Using the *CFDDiscovery* algorithm, the output list already contains in each position one different dependency. After that, the parser detect the LHS and the RHS of the dependency using the value '$\rightarrow$' to split the two parts of the rule.

In the second step, we filter the rules discarding the ones that do not satisfy both the following two constraints:

1. all the attributes in the dependency must have the corresponding value (otherwise we deal with AFDs),

2. there must be at least one protected attribute and the target variable in the dependency, so that the resulting ACFDs could show bias regarding protected attributes.

To check the first constraint we analyze each side of the dependency, then we detect the couples *"attribute-value"* (they are simply divided by a comma) and, for each couple, we search for the '=' symbol that indicates that the attribute has a corresponding value. Applying the first constraint we filter the dependencies obtaining only ACFDs; with the second check we filter the ACFDs obtaining rules that could be biased on the protected attributes and the target class. To accomplish the second constraint we simply check all the attributes comparing in each dependency.

The third stage simply organizes these filtered ACFDs in a more adequate structure that is the dictionary, that appears as a list of rules.

During the ACFDs Filtering phase, the user can also add some constraints:

- Decide which **values must appear** in the rules. An example could be setting that all the dependencies must involve only the *'Female'* people, because the researcher is interested only in peculiar aspects of women.

- Decide which **values must not appear** in the rules. For example, some *age ranges* could be not interesting for the research, or the user could want to discard rules that interest a specific group.

If the user is interested in studying small groups, we do not suggest to use these constraints. In next sections, we will describe more in details how to study small groups more in deep.

5.3.8 Running Example: ACFDs Filtering

Given the output of the algorithm that discovers the ACFDs we have the following steps:

1. Use the **parser**,

2. Filter the **ACFDs**,

3. Create the **dictionary**.

Firstly, we analyze each string of the output and we analyze the dependency. We start the process by removing the useless characters as parenthesis, then we split the LHS from the RHS of the dependency. After that, we detect the AFDs and we discard them keeping only the ACFDs. In figure 5.9 there is the output of this first process.

Figure 5.9: First split, dividing each rule into RHS and LHS.

```
Dependency n. 0 :  [['education-degree=MiddleSchool'], ['income=<50K']]
Dependency n. 1 :  [['age-range=15-30'], ['income=<50K']]
Dependency n. 2 :  [['education-degree=Bach', 'age-range=15-30'], ['income=<50K']]
Dependency n. 3 :  [['education-degree=MiddleSchool', 'age-range=15-30'], ['income=<50K']]
Dependency n. 4 :  [['education-degree=Assoc', 'age-range=15-30'], ['income=<50K']]
Dependency n. 5 :  [['education-degree=HS-College', 'age-range=15-30'], ['income=<50K']]
Dependency n. 6 :  [['native-country=NC-Hispanic'], ['income=<50K']]
Dependency n. 7 :  [['income=>50K'], ['native-country=NC-White']]
Dependency n. 8 :  [['income=<50K'], ['native-country=NC-White']]
Dependency n. 9 :  [['native-country=NC-White', 'education-degree=MiddleSchool'], ['income=<50K']]
Dependency n. 10 :  [['income=>50K', 'education-degree=Mast'], ['native-country=NC-White']]
Dependency n. 11 :  [['income=>50K', 'education-degree=Bach'], ['native-country=NC-White']]
```

Figure 5.10: Creating the LHS and RHS of each dependency.

```
Total number of dependencies in the dictionary:  100
Dependency n. 0 :  {'lhs': {'education-degree': 'MiddleSchool'}, 'rhs': {'income': '<=50K'}}
Dependency n. 1 :  {'lhs': {'age-range': '15-30'}, 'rhs': {'income': '<=50K'}}
Dependency n. 2 :  {'lhs': {'education-degree': 'Bach', 'age-range': '15-30'}, 'rhs': {'income': '<=50K'}}
Dependency n. 3 :  {'lhs': {'education-degree': 'MiddleSchool', 'age-range': '15-30'}, 'rhs': {'income': '<=50K'}}
Dependency n. 4 :  {'lhs': {'education-degree': 'Assoc', 'age-range': '15-30'}, 'rhs': {'income': '<=50K'}}
Dependency n. 5 :  {'lhs': {'education-degree': 'HS-College', 'age-range': '15-30'}, 'rhs': {'income': '<=50K'}}
Dependency n. 6 :  {'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}
Dependency n. 7 :  {'lhs': {'income': '>50K'}, 'rhs': {'native-country': 'NC-White'}}
Dependency n. 8 :  {'lhs': {'income': '<=50K'}, 'rhs': {'native-country': 'NC-White'}}
Dependency n. 9 :  {'lhs': {'native-country': 'NC-White', 'education-degree': 'MiddleSchool'}, 'rhs': {'income': '<=50K'}}
Dependency n. 10 :  {'lhs': {'income': '>50K', 'education-degree': 'Mast'}, 'rhs': {'native-country': 'NC-White'}}
Dependency n. 11 :  {'lhs': {'income': '>50K', 'education-degree': 'Bach'}, 'rhs': {'native-country': 'NC-White'}}
```

After that we divide each couple "attribute-value" contained in the LHS and RHS of the dependency, and we create a structure in which save the two dependency sides and the relative couples of "attribute-value". Figure 5.10 displays the result of this step.

Lastly we discard the dependencies that do not contain protected attributes or the target class, then we build the dictionary of all the ACFDs

Figure 5.11: Dictionary with only interesting ACFDs

```
Total number of ACFDs in the dictionary:  84
ACFD n. 0 :  {'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}
ACFD n. 1 :  {'lhs': {'income': '>50K'}, 'rhs': {'native-country': 'NC-White'}}
ACFD n. 2 :  {'lhs': {'income': '<=50K'}, 'rhs': {'native-country': 'NC-White'}}
ACFD n. 3 :  {'lhs': {'native-country': 'NC-White', 'education-degree': 'MiddleSchool'}, 'rhs': {'income': '<=50K'}}
ACFD n. 4 :  {'lhs': {'income': '>50K', 'education-degree': 'Mast'}, 'rhs': {'native-country': 'NC-White'}}
ACFD n. 5 :  {'lhs': {'income': '>50K', 'education-degree': 'Bach'}, 'rhs': {'native-country': 'NC-White'}}
ACFD n. 6 :  {'lhs': {'education-degree': 'Assoc', 'income': '>50K'}, 'rhs': {'native-country': 'NC-White'}}
ACFD n. 7 :  {'lhs': {'education-degree': 'HS-College', 'income': '>50K'}, 'rhs': {'native-country': 'NC-White'}}
ACFD n. 8 :  {'lhs': {'income': '<=50K', 'education-degree': 'Bach'}, 'rhs': {'native-country': 'NC-White'}}
ACFD n. 9 :  {'lhs': {'income': '<=50K', 'education-degree': 'MiddleSchool'}, 'rhs': {'native-country': 'NC-White'}}
ACFD n. 10 :  {'lhs': {'income': '<=50K', 'education-degree': 'Assoc'}, 'rhs': {'native-country': 'NC-White'}}
ACFD n. 11 :  {'lhs': {'income': '<=50K', 'education-degree': 'HS-College'}, 'rhs': {'native-country': 'NC-White'}}
```

that respect the constraint.
We can notice that some rules disappeared from figure 5.10 to figure 5.11. The first six rules present in figure 5.10 do not involve any protected attribute, so they are not interesting and they are omitted. In fact, the original total number of dependencies was 100, and after this step, we keep only 84 ACFDs.

5.3.9 ACFDs Selection

After the ACFDs Filtering phase we obtain a dictionary that contains all the ACFDs, thus, in this phase, we start investigating on the dependencies, in particular on the "ethical" aspect.
In the dictionary appear the ACFDs that satisfy the *CFDDiscovery* constraints and the other two constraints that we have previously established, but they could show bias in the dataset or not. So, the next step consists in the selection of the dependencies that reveal unfairness in the dataset.

Before performing the selection, we build a table containing all the dependencies and the relative metrics.
The table is composed by different columns:

- **Rule**: it simply contains the rule.
- **Support**: it contains the percentage of the samples in the dataset that verifies the dependency, it indicates how much the rule is *"pervasive"*; a high support expresses a big number of tuples involved, so the rule is more "universal".
- **Confidence**: it shows how frequently the dependency is verified knowing that the LHS part is verified. A high confidence demonstrates that, the rule is verified in almost all cases in which the LHS of the rule is verified, so there are few tuples that should be removed in order for the dependency to hold.
- **Difference**: it indicates how much a dependency is *"ethical"*. The more this metric is high, the more the CFD shows a behaviour that is not in line the general one.
- **ProtectedAttributeDifference**: it consists in a different column for each protected attribute; it indicates how much the rule shows bias, but paying attention to the specific value of the protected attribute. The more this value is high, the more the rule is *discriminatory with respect to the specific protected attribute value.*

We decide to focus on these metrics because an ACFD has importance according to three main aspects: the coverage or pervasiveness, the fairness index and the discrimination due to the protected attribute. These aspects could be expressed by three metrics:

- **Support** that is the percentage of tuples that verifies the dependency. It indicates the *pervasiveness* of the ACFD.

- **Difference** that express the group fairness, so it can be seen as *fairness index* of the dependency.
- **Protected Attribute Difference** that is a disparity measure, related to the protected attribute. It can be seen as *protected attribute fairness index*.

We already defined in chap. 3 the **Support** metric and it is computed for the rule $X \to Y$ as:

$$Support(X \to Y) = sup(X, Y) = \frac{\#(X, Y)}{all - tuples},$$

where $\#(X, Y)$ is the count of how many times X and Y appears together in the dataset and *all-tuples* is the total number of tuples in the dataset.
The support is the percentage of the samples in the dataset that verifies the dependency $X \to Y$. For this reason we can say that the support indicates the *pervasiveness* of the ACFD.

For each rule, we define the **Difference** metric, that is the difference between the rule confidence and the confidence computed without the protected attributes of the LHS of the ACFD.
Given a dependency in the form:

$$\phi : (X \to Y)$$

X represent the LHS of the rule, Y the RHS of the rule and we define Z as $Z = (X - \{ProtectedAttributes\})$ so is the LHS of the dependency without the couples "protected attribute-value".
We define the *Difference* as:

$$Difference = Confidence - NoProtectedAttributeConfidence$$

where

$$Confidence(X \to Y) = Conf(\phi) = \frac{sup(X, Y)}{sup(X)}$$

and

$$NoProtectedAttributeConfidence(X \to Y) = NoProtAttrConf(\phi) = \frac{\sup(Z, Y)}{\sup(Z)}.$$

So, the *Difference* becomes:

$$Difference(X \to Y) = Diff(\phi) = \frac{sup(X, Y)}{sup(X)} - \frac{\sup(Z, Y)}{\sup(Z).}$$

if Z is the empty set the *Difference* becomes:

$$Difference(X \rightarrow Y) = Diff(\phi) = \frac{sup(X,Y)}{sup(X)} - \sup(Y).$$

The *Difference* metric indicates how much the dependency respects the global behaviour.
If the *Confidence* is high, the rule is valid in almost all the cases in which the LHS of the dependency holds. If the *No-Protected Attributes Confidence* is low, there are few tuples that contains the both the LHS part without protected attributes and the RHS.
The difference between these two confidence measures can be seen as the difference between two distributions.
Let us propose an example of a possible rule to explain more in detail this concept:

$$\phi_1 : (sex = Female, workclass = Private) \rightarrow (income =\leq 50K)$$

The confidence is:

$$Conf(\phi_1) = \frac{\sup(sex = Female, workclass = Private, income =\leq 50K)}{\sup(sex = Female, workclass = Private)}$$

and *No-Protected Attribute Confidence* is computed as:

$$NoProtAttrConf(\phi_1) = \frac{\sup(workclass = Private, income =\leq 50K,)}{\sup(workclass = Private)}$$

So, the *Difference* becomes:

$$Diff(\phi_1) : Conf(\phi_1) - NoProtAttrConf(\phi_1)$$

The *Difference* value represents the distance between two distributions.
The first element is the confidence, so it is the support of all the female people that work by themselves and gain less than 50K dollars/year divided by the support of all the female people that work by themselves. The second element is the *No-Protected Attribute Confidence* that is the support of all the people that work privately and gain less than 50K dollars/year divided by the support of all the population that work privately.

- If the two distributions are almost equal, the **Difference is almost zero** and fairness is respected. In fact, in this case the group fairness is guaranteed because the group is treated equally with respect to the elements of the population that have the same characteristics without specifying the protected attribute.

- If the first element of the difference is greater than the second one, the **Difference is positive**. In this case, group and subgroup fairness are no more guaranteed. Taking into account the aforementioned example, if there are more women that work privately and gain less than 50K dollars/year with respect to all people that work privately and gain less than 50K dollars/year, this means that the "female" group is not treated equally.

- If the second element is greater than the first one, the **Difference is negative**. Looking at the example, in this case the "female" group has a better treatment. In fact, women that work privately and gain less than 50K dollars/year are in proportion less than all the people that work privately and gain less than 50K dollars/year, demonstrating a bias in the other way.

Moreover, a dependency could contain on the LHS more than one protected attribute at the same time. For this reason, we introduce the last metric: the Protected Attribute Difference. The **Protected Attribute Difference** is very similar to the Difference measure, but it is computed for each protected attribute p considering only p as protected at each time.
Let us define this metric starting from a general rule in the form:

$$\phi : (X \rightarrow Y).$$

We define W as $W = (X - \{p\})$ so W is the LHS of the dependency without the couple: "protected attribute p-value".
We define the Protected Attribute Difference, shortly, *p-Difference*, as:

$$pDifference = Confidence - NoPConfidence$$

where the confidence is computed as conventional:

$$Confidence(X \rightarrow Y) = Conf(\phi) = \frac{sup(X,Y)}{sup(X)}$$

and the *No-P Confidence* is the confidence metrics in which ϕ does not contain the protected attribute p:

$$NoPConfidence(X \rightarrow Y) = NoPConf(\phi) = \frac{\sup(W,Y)}{\sup(W)}.$$

So the *p-Difference* becomes:

$$pDifference(X \rightarrow Y) = pDiff(\phi) = \frac{sup(X,Y)}{sup(X)} - \frac{\sup(W,Y)}{\sup(W).}$$

Looking at the example:

$$\phi_2 : (sex = Female, race = Black) \rightarrow (income =\leq 50K)$$

The *Difference* is computed as:

$$Diff(\phi_2) = Conf(\phi_2) - NoProtAttrConf(\phi_2)$$

where:

$$Conf(\phi_2) = \frac{\sup(sex = Female, race = Black, income =\leq 50K)}{\sup(sex = Female, race = Black)}$$

and

$$NoProtAttrConf(\phi_2) = sup(income =\leq 50K)$$

If we compute the Difference respect to the protected attribute *'Sex'*, we compute the difference metric assuming that only *'Sex'* is protected. Thus, the *Sex-Difference* is:

$$SexDiff(\phi_2) = Conf(\phi_2) - NoSexConf(\phi_2)$$

where the confidence is the same as before and *No-Sex Confidence* is:

$$NoSexConf(\phi_2) = \frac{\sup(race = Black, income =\leq 50K)}{\sup(race = Black)}.$$

In this case, even if *'Race'* is protected, in the *Sex-Difference* computation, we do not consider it protected.

Similar, the *Race-Difference* is computed as:

$$RaceDiff(\phi_2) = Conf(\phi_2) - NoRaceConf(\phi_2)$$

where the confidence is the same as before and *No-Race Confidence* is:

$$NoRaceConf(\phi_2) = \frac{\sup(sex = Female, income =\leq 50K)}{\sup(sex = Female)}$$

To recap, the *'Protected Attribute Difference'* of p is the Difference computation in which we perform a difference between the confidence and the confidence computed using the LHS of the dependency in which we remove

only the protected attribute p. This value indicates how much the dependency shows bias on the protected attribute p domain.
The reader can notice that in the computation of the *p-Difference*, the protected attributes that are different from p are not considered as protected ones. This choice is due to the fact that computing the *p-Difference* means computing fairness only looking at p, instead the general difference metric checks the fairness with respect to all the population considering all the protected attributes. Obviously, the *p-Difference* is computed only when p appears in the dependency.

Given the table, we want to find ACFDs that demonstrate discrimination, so, to obtain an initial subset of interesting dependencies, we select all ACFDs that have:

$$Difference > minThreshold.$$

In this way, we can select all the rules in which the difference metric is relevant.
In our research the difference metric is our definition of fairness based on the notions of group and subgroup fairness.
In fact, when the difference is above the minimum threshold, there is a significant inequality between the group involved by the ACFD and the general behaviour of the population.
Obviously, the *minThreshold* parameter is set according to the research needs. According to the value of the parameter, the quality and the number of the ACFDs change.

When the 'unethical" dependencies are selected, we can build again the table that contains all the rules and for each one of them report all the relevant information. This table contains the dependencies and the relative metrics.

5.3.10 Running Example: ACFDs Selection

In this step, we analyze each Approximate Conditional Functional Dependency and we compute the relative metrics, then we select the ones that are interesting from the ethics point of view.

Figure 5.12 displays the first 12 rows of table with the ACFDs and the relative metrics. In this step, we analyze the ACFDs contained in the dictionary previously computed, and for each rule we compute the support, the confidence, the difference and for each protected attribute, if present,

also the protected attribute difference.
Then we apply the selection constraint: $Difference > minThreshold$ where the $minThreshold = 0.1$.

Figure 5.12: Table with ACFDs and relative metrics

	Rule	Support	Confidence	Diff	RaceDiff	SexDiff	NativeCountryDiff
0	{'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.043919	0.908156	0.157021	0	0	0.157021
1	{'lhs': {'income': '>50K'}, 'rhs': {'native-country': 'NC-White'}}	0.231861	0.931673	0.019777	0	0	0.000000
2	{'lhs': {'income': '<=50K'}, 'rhs': {'native-country': 'NC-White'}}	0.680036	0.905344	-0.006552	0	0	0.000000
3	{'lhs': {'native-country': 'NC-White', 'education-degree': 'MiddleSchool'}, 'rhs': {'income': '<=50K'}}	0.074016	0.933138	-0.003229	0	0	-0.003229
4	{'lhs': {'income': '>50K', 'education-degree': 'Mast'}, 'rhs': {'native-country': 'NC-White'}}	0.028141	0.924837	0.012728	0	0	0.000000
5	{'lhs': {'income': '>50K', 'education-degree': 'Bach'}, 'rhs': {'native-country': 'NC-White'}}	0.065465	0.928975	0.013431	0	0	0.000000
6	{'lhs': {'education-degree': 'Assoc', 'income': '>50K'}, 'rhs': {'native-country': 'NC-White'}}	0.060691	0.945764	0.008132	0	0	0.000000
7	{'lhs': {'education-degree': 'HS-College', 'income': '>50K'}, 'rhs': {'native-country': 'NC-White'}}	0.051079	0.952999	0.017119	0	0	0.000000
8	{'lhs': {'income': '<=50K', 'education-degree': 'Bach'}, 'rhs': {'native-country': 'NC-White'}}	0.087606	0.905757	-0.009786	0	0	0.000000
9	{'lhs': {'income': '<=50K', 'education-degree': 'MiddleSchool'}, 'rhs': {'native-country': 'NC-White'}}	0.074016	0.882260	-0.003053	0	0	0.000000
10	{'lhs': {'income': '<=50K', 'education-degree': 'Assoc'}, 'rhs': {'native-country': 'NC-White'}}	0.218867	0.935402	-0.002230	0	0	0.000000
11	{'lhs': {'income': '<=50K', 'education-degree': 'HS-College'}, 'rhs': {'native-country': 'NC-White'}}	0.254201	0.932515	-0.003366	0	0	0.000000

As it can be noticed from the table 5.12, there are many rules that should be removed because they do not satisfy the constraint, as rule number 2:

$$(income =\leq 50K) \rightarrow (native - country = NC - White).$$

This rule is a valid ACFD, but its difference is negative, so the "NC-White" group is treated better with respect to the rest of the population. Thus, this rule shows discrimination in a puzzling way and not a direct bias. In fact, the corresponding "discriminatory" ACFD is the rule number 1:

$$(income => 50K) \rightarrow (native - country = NC - White)$$

that has small positive difference.

The table 5.13 is a filtered version of the previous table where the ACFDs are chosen if they satisfy the criterion:

$$Difference > minThreshold,$$

and we established the $minThreshold = 0.1$. The starting length of the dictionary was 84, after this phase the selected ACFDs are in total 17.

From the figure 5.13, we want to highlight some interesting rules:

$$\phi_0 : (native - country = NC - Hispanic) \rightarrow (income =\leq 50K)$$

Figure 5.13: First table with selected ACFDs

	Rule	Support	Confidence	Diff	RaceDiff	SexDiff	NativeCountryDiff
0	{'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.043919	0.908156	0.157021	0	0	0.157021
19	{'lhs': {'hours-per-week': '21-40', 'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.033909	0.928312	0.122855	0	0	0.122855
26	{'lhs': {'sex': 'Female'}, 'rhs': {'income': '<=50K'}}	0.287447	0.886345	0.135210	0	0.13521	0.000000
28	{'lhs': {'sex': 'Female', 'education-degree': 'Assoc'}, 'rhs': {'income': '<=50K'}}	0.101362	0.910932	0.126163	0	0.126163	0.000000
30	{'lhs': {'education-degree': 'HS-College', 'income': '>50K'}, 'rhs': {'sex': 'Male'}}	0.046538	0.868275	0.183995	0	0	0.000000
33	{'lhs': {'age-range': '45-60', 'income': '>50K'}, 'rhs': {'sex': 'Male'}}	0.084159	0.877636	0.165426	0	0	0.000000
38	{'lhs': {'sex': 'Male', 'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.028738	0.881994	0.130859	0	-0.0261624	0.195754
39	{'lhs': {'sex': 'Female', 'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}	0.262190	0.885382	0.134246	0	0.139644	-0.000963
44	{'lhs': {'race': 'Black'}, 'rhs': {'income': '<=50K'}}	0.081309	0.870167	0.119031	0.119031	0	0.000000
61	{'lhs': {'native-country': 'NC-White', 'race': 'Black'}, 'rhs': {'income': '<=50K'}}	0.075839	0.869631	0.118496	0.123893	0	-0.000535
62	{'lhs': {'race': 'White', 'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.032252	0.916196	0.165061	0.00803959	0	0.179863
68	{'lhs': {'sex': 'Female', 'race': 'Black'}, 'rhs': {'income': '<=50K'}}	0.043588	0.939286	0.188150	0.0529407	0.069119	0.000000

$$\phi_{26} : (sex = Female) \rightarrow (income =\leq 50K)$$

$$\phi_{44} : (race = Black) \rightarrow (income =\leq 50K)$$

$$\phi_{68} : (sex = Female, race = Black) \rightarrow (income =\leq 50K).$$

At the end of this step, the user can already have an overview of the bias present in the dataset, looking at the obtained ACFDs.
For example, "Hispanic","Female" and "Black" groups suffer from discrimination with respect to the rest of the population, in fact, people that belong to one or more of these groups have an income that is below the 50'000 dollars/year because of the nationality, the sex or the race.
We want to highlight the fact that rule ϕ_{68} is not a direct consequence of ACFDs ϕ_{26} and ϕ_{44}. From these last two dependencies, the user can state that the "Female" and "Black" groups suffer from discrimination, but it could be not true that the subgroup "Female-Black" suffers from discrimination.
Let us introduce two additional hypothetical ACFDs:

$$\beta : (race = Other) \rightarrow (income =\leq 50K).$$

$$\gamma : (sex = Female, race = Other) \rightarrow (income =\leq 50K).$$

If ϕ_{26} and β are verified, so the $Difference > minThreshold$, this means that these ACFDs show bias in the "Female" and "Other" groups, we can not state that also γ is true. In fact in the "Other" group, women could be treated better than men, so γ could be not verified.
The framework works with ACFDs, and due to the "Approximate" term, we can not infer γ from ϕ_{26} and β; only is γ has $Difference > minThreshold$ we can say that this rule shows discrimination to the "Female-Other" group.

Moreover, looking at the figure, we can analyze ACFD number 38:

$\phi_{38} : (sex = Male, native{-}country = NC{-}Hispanic) \rightarrow (income =\leq 50K)$.

From this ACFD, the reader could think that, in the Hispanic group, men are treated worse than women. Although, the difference computed with respect to attribute *'Sex'* is not positive. This means that, in the Hispanic minorities, men are not treated worse women.
Actually, if we recall again rule number 68:

$$\phi_{68} : (sex = Female, race = Black) \rightarrow (income =\leq 50K)$$

in this case the *"Sex-Difference"* is slightly positive, so in the black race group, women are treated worse than men.

We could be satisfied with these results, but continuing the research could lead to some other interesting results, so in the next paragraph we analyze more in depth the dependencies.

5.3.11 ACFDs Completion

At this step of the framework, we use the table containing all the interesting ACFDs and we analyze them to discover other meaningful dependencies.
One alternative is that the user could stop the research and save the most interesting dependencies from the table. In the other alternative, the user could continue the research analyzing more in deep the selected ACFDs.

The last part of the framework is composed by three main steps:

1. For each rule, compute the **possible combinations** over the protected attributes and the target class.
2. **Select the obtained rules** and rank them.
3. Let the **user select** the most meaningful rules and compute a final overview of the rules with a summarizing scoring about the fairness of the dataset.

In this paragraph we concentrate on the first step, the computation of the possible combinations for the ACFDs.

We start extracting from each rule the protected attributes and the target variable. Then, we compute the Cartesian product between all these

attribute domain values. For each combination we create a new ACFD.

Given a generic rule:

$$\phi : (X \rightarrow Y),$$

we define A as $A = (p_1, p_2, .., p_N)$ that is the set of all protected attributes appearing in X, and $D_p = (D_{p_1}, D_{p_2}, .., D_{p_N})$ that is composed by the domains of the protected attributes in A.
The framework computes all possible combinations of values in the domains of all the protected attributes and the target variable. We define $D = (D_{p_1}, D_{p_2}, .., D_{p_N}, D_Y)$ where D contains all the domains of all protected attributes and the target variable. From D we compute all the possible combinations:

$$C = (D_{p_1} \times D_{p_2} \times .. \times D_{p_N} \times D_Y).$$

Recalling again the example:

$$\phi_1 : (sex = Female) \rightarrow (income =\leq 50K).$$

Assuming that ϕ_1 is a selected rule, at this step the framework computes all its possible combinations of values in the domains of all the protected attributes and target variable present. In the example, there is only one protected attribute *"Sex"*, so $A = \{Sex\}$, and the target class is *"Income"*, so $Y = \{Income\}$.
For the attribute *"Sex"* the domain contains only two values: *"Female"* and *"Male"*; for class variable *"Income"* the domain is composed by values:"$\leq 50K$" and "$> 50K$", so $D = \{(Female, Male), (\leq 50K, > 50K)\}$.
All the possible combinations are:

$$C = \{(Female, \leq 50K), (Female, > 50K), (Male, \leq 50K), (Male, > 50K)\}.$$

So, from ϕ_1, the framework generates four rules:

$$\phi_1 : (sex = Female) \rightarrow (income =\leq 50K)$$

$$\phi_2 : (sex = Female) \rightarrow (income => 50K)$$

$$\phi_3 : (sex = Male) \rightarrow (income =\leq 50K)$$

$$\phi_4 : (sex = Male) \rightarrow (income => 50K).$$

To recap, given the selected ACFDs, the framework computes all the possible combinations for each rule. This computation allows the user to study all the domain of the protected attributes and of the target class, allowing the ACFDs completion and bringing to some considerations.

- If an original rule is chosen , it is because it shows discrimination, as for example ϕ_1. From this dependency we understand that female people in proportion gain less than 50'000 dollars/year with the rest of population. So, in the population there should be a group that is treated better than women, so that in proportion, gain more than 50'000 dollars/year. From ϕ_1, it seems intuitive to study also ϕ_2, ϕ_3 and ϕ_4, from which we expect to be valid only ϕ_4.

- With the completion phase, the combination computation allows to study also small groups that could be not studied. For example, if men are fewer than women ϕ_3 and ϕ_4 would not be studied.
  The support of dependencies obtained by the combination process could be lower than the 'minSupport' established for *CFDDiscovery* algorithm. Thus, the obtained ACFDs could interest a small number of tuples as for a group that contains few elements. Analyzing the domain of the protected attributes, we can discover more specific dependencies. On one side, with the completion process, we increase the total number of rules, but on the other side we expect to see bias and discrimination in small groups.

After that, for each obtained ACFD we compute metrics and we build again the table with all ACFDs to analyze the resulting dependencies.

5.3.12 Running Example: ACFDs Completion

Coming back to our example, the starting ACFDs are 17. At this step, we compute all the possible combinations over the protected attributes and the target class, and then, we build the table with all the ACFDs and the relative metrics.
In figure 5.14 there are all the possible combinations computed from the first dependency.

From the original 17 ACFDs, after the completion process, we obtain a list of 224 rules. Figure 5.15 displays the first 12 dependencies and the relative metrics.

At this phase, we do not have to check if the ACFDs contain a protected attribute and the target, because the obtained dependencies are combinations of already checked rules.

Figure 5.14: Combination output of a ACFD

```
Original ACFD:  {'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}

ACFD n. 0 :  {'lhs': {'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}
ACFD n. 1 :  {'lhs': {'native-country': 'NC-White'}, 'rhs': {'income': '>50K'}}
ACFD n. 2 :  {'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}
ACFD n. 3 :  {'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '>50K'}}
ACFD n. 4 :  {'lhs': {'native-country': 'NC-Non-Hisp-White'}, 'rhs': {'income': '<=50K'}}
ACFD n. 5 :  {'lhs': {'native-country': 'NC-Non-Hisp-White'}, 'rhs': {'income': '>50K'}}
ACFD n. 6 :  {'lhs': {'native-country': 'NC-Asian-Pacific'}, 'rhs': {'income': '<=50K'}}
ACFD n. 7 :  {'lhs': {'native-country': 'NC-Asian-Pacific'}, 'rhs': {'income': '>50K'}}
```

Figure 5.15: Table with ACFDs after the completion process

	Rule	Support	Confidence	Diff	RaceDiff	SexDiff	NativeCountryDiff
0	{'lhs': {'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}	0.680036	0.745738	-0.005397	0	0	-0.005397
1	{'lhs': {'native-country': 'NC-White'}, 'rhs': {'income': '>50K'}}	0.231861	0.254262	0.005397	0	0	0.005397
2	{'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.043919	0.908156	0.157021	0	0	0.157021
3	{'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '>50K'}}	0.004442	0.091844	-0.157021	0	0	-0.157021
4	{'lhs': {'native-country': 'NC-Non-Hisp-White'}, 'rhs': {'income': '<=50K'}}	0.014551	0.683801	-0.067335	0	0	-0.067335
5	{'lhs': {'native-country': 'NC-Non-Hisp-White'}, 'rhs': {'income': '>50K'}}	0.006729	0.316199	0.067335	0	0	0.067335
6	{'lhs': {'native-country': 'NC-Asian-Pacific'}, 'rhs': {'income': '<=50K'}}	0.012629	0.684022	-0.067114	0	0	-0.067114
7	{'lhs': {'native-country': 'NC-Asian-Pacific'}, 'rhs': {'income': '>50K'}}	0.005834	0.315978	0.067114	0	0	0.067114
8	{'lhs': {'hours-per-week': '21-40', 'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}	0.443866	0.800801	-0.004656	0	0	-0.004656
9	{'lhs': {'hours-per-week': '21-40', 'native-country': 'NC-White'}, 'rhs': {'income': '>50K'}}	0.110411	0.199199	0.004656	0	0	0.004656
10	{'lhs': {'hours-per-week': '21-40', 'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.033909	0.928312	0.122855	0	0	0.122855
11	{'lhs': {'hours-per-week': '21-40', 'native-country': 'NC-Hispanic'}, 'rhs': {'income': '>50K'}}	0.002619	0.071688	-0.122855	0	0	-0.122855

As the reader can notice, the support and the confidence of some dependencies are very low. For example, rules that involve "NC-Hisp-White" group or "NC-Asian-Pacific" group, have low support and sometimes low confidence. These rules are obtained with the completion process and they could be obtained from the *CFDDiscovery* algorithm only if we decrease the input parameters. Thus, the combination process allows us to detect rules that are not directly output of the *CFDDiscovery* algorithm.

Observing the table 5.15 and analyzing the first eight ACFDs, we can compare the dependencies that involve one group at a time and we select the ones that have positive difference metric:

$$\phi_1 : (native - country = NC - White) \rightarrow (income => 50K)$$

$$\phi_2 : (native - country = NC - Hispanic) \rightarrow (income =\leq 50K)$$

$\phi_5 : (native{-}country = NC{-}Non{-}Hispanic{-}White) \rightarrow (income => 50K)$

$\phi_7 : (native - country = NC - Asian - Pacific) \rightarrow (income => 50K)$.

The ACFDs can be relevant on two aspect:

1. the ACFD has a high support metric, so it involves a large number of tuples, as for example rule ϕ_1;
2. the ACFD has a high difference metric, so it shows relevant bias and discrimination, as for example rules ϕ_2.

All these ACFDs have positive difference metrics, so they are meaningful from an "ethical" point of view. The "native-country" group that is more discriminated in terms of "income" is composed by Hispanic people; in fact, for the other groups, the dependencies show that they gain more than 50'000 dollars/year on average.

Looking at the groups that are different from "NC-Hispanic", we can notice that they have diverse positive *Difference* metrics, in particular, the "NC-White" group has the smallest positive difference. We know that "NC-White" is the group composed only by people coming from United States. Looking at its difference value, we can suppose that people that were born in U.S. are treated, in terms of income, worse than "NC-Non-Hispanic-White" group or "NC-Asian-Pacific" group.
This is true, but we have to notice that, in this dataset, people that were born in United States compose the majority group. "NC-White" is composed by people of various races, thus people coming from U. S. do not correspond to white people, but there are people of various races, for example the offspring of migrants that are born in United States.
Thus, in general, the "NC-White" group is treated worse with respect to "NC-Non-Hispanic-White" people or "NC-Asian-Pacific" people, but we cannot say that this is true for each race subgroup that was born in United States.

5.3.13 ACFDs Ranking

The ACFDs Ranking is the step of the framework that starts from the selection of the meaningful ACFDs, and, finally, it shows a ranking of the dependencies ordered by a criterion chosen by the user.

For the selection phase we use again the previous criterion:

$$Difference > minThreshold.$$

Thus, the framework discards the rules that have difference below the threshold, so that do not exhibit discrimination.
According to the user needs, she can decide to use the same value for the threshold parameter of the previous selection or to modify it.

Next, the user can order the obtained rules according to one of these three different criteria:

- **Support option**: means sorting the ACFDs in decreasing order of support. The support metric indicates the number of tuples involved by the dependency, so the higher is the number, the more tuples are involved by the ACFD. Ordering dependencies by support highlights the *pervasiveness* of the rule.

- **Difference option**: means sorting the ACFDs using the Difference metric. Using this criterion implies that the dependencies that show higher difference are put first in the ranking. This method highlights the *ethical aspect* of the rules.

- **Mean option**: for each ACFD, the mean is computed as the mean of the support and the difference metric of the rule. This method tries to combine both aspects of the dependency: the ethical perspective and the pervasiveness of it. Sorting using this criterion means putting first the dependencies that have the *best trade-off* between difference and support.

At this point the user can choose one of these criteria to sort the ACFDs and she obtains the ranking.

Let us express the importance of the **ranking**.
In a real world large dataset, the ACFDs mined at this step could be many, in order of thousands. For the user, looking at all these dependencies is an impossible task and a waste of time. In a concrete scenario, this phase has a cost that is proportional to the number of tuples that the user has to look and choose. Thus, for the user is necessary to have the rules ordered by a criterion, so that the selection process speeds up and the cost could be maintained low.
Furthermore, the criterion should fit the user necessities, for this reason we have created three different criteria, so that the user can choose the most adequate one.

5.3.14 Running Example: ACFDs Ranking

This step of the framework is composed by two phases:

1. Select the interesting rules with the *Difference* criterion.
2. Presents the ranking with selected ACFDs and metrics.

As output of the previous phase we have a list of all the combinations of the ACFDs, and they could be "unethical" or not. From this list, we perform a selection using the criterion:

$$Difference > minThreshold.$$

We decided to keep the same minimum threshold as before, so it has value 0.1. The total number of dependencies derived from the completion process is 224, after the selection phase the obtained ACFDs are 64.

After that, the user can select a method to order the ACFDs using the *Support* or the *Difference* or the *Mean* option. We decide to create the ranking using the "Mean" preference, so the dependencies are ordered using the mean between the support and the difference metric. Figure 5.16 shows the first 12 rules in the ranking.

Figure 5.16: Ranking of first 12 ACFDs after first selection with "Mean" criterion

	Rule	Support	Confidence	Diff	RaceDiff	SexDiff	NativeCountryDiff	Mean
16	{'lhs': {'sex': 'Female'}, 'rhs': {'income': '<=50K'}}	0.287447	0.886345	0.135210	0	0.13521	0.000000	0.211329
32	{'lhs': {'sex': 'Female', 'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}	0.262190	0.885382	0.134246	0	0.139644	-0.000963	0.198218
48	{'lhs': {'sex': 'Female', 'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}	0.262190	0.885382	0.134246	0	0.139644	-0.000963	0.198218
174	{'lhs': {'race': 'White', 'sex': 'Female'}, 'rhs': {'income': '<=50K'}}	0.229540	0.877026	0.125891	-0.00931871	0.140694	0.000000	0.177716
154	{'lhs': {'sex': 'Female', 'race': 'White'}, 'rhs': {'income': '<=50K'}}	0.229540	0.877026	0.125891	-0.00931871	0.140694	0.000000	0.177716
220	{'lhs': {'workclass': 'Private', 'sex': 'Female'}, 'rhs': {'income': '<=50K'}}	0.229408	0.905653	0.124445	0	0.124445	0.000000	0.176926
111	{'lhs': {'native-country': 'NC-Asian-Pacific', 'race': 'Other'}, 'rhs': {'income': '>50K'}}	0.000099	0.500000	0.251135	0.184022	0	0.409091	0.125617
145	{'lhs': {'race': 'Other', 'native-country': 'NC-Asian-Pacific'}, 'rhs': {'income': '>50K'}}	0.000099	0.500000	0.251135	0.184022	0	0.409091	0.125617
31	{'lhs': {'age-range': '45-60', 'income': '>50K'}, 'rhs': {'sex': 'Male'}}	0.084159	0.877636	0.165426	0	0	0.000000	0.124793
152	{'lhs': {'race': 'Amer-Indian-Eskimo', 'native-country': 'NC-Asian-Pacific'}, 'rhs': {'income': '<=50K'}}	0.000066	1.000000	0.248865	0.315978	0	0.118881	0.124466
112	{'lhs': {'native-country': 'NC-Asian-Pacific', 'race': 'Amer-Indian-Eskimo'}, 'rhs': {'income': '<=50K'}}	0.000066	1.000000	0.248865	0.315978	0	0.118881	0.124466
150	{'lhs': {'race': 'Amer-Indian-Eskimo', 'native-country': 'NC-Non-Hisp-White'}, 'rhs': {'income': '<=50K'}}	0.000033	1.000000	0.248865	0.316199	0	0.118881	0.124449

5.3.15 ACFDs User Selection and Scoring

In this last phase of the process, the user selects N dependencies from the ranking list, that are interesting for the research needs. Using only the N selected ACFDs, the framework computes a final scoring outline that is composed by three measures that are:

- **Cumulative Support**: is the percentage of tuples of the dataset involved by the selected ACFDs. The more this value is similar to 1, the more rules have covered the dataset.
- **Difference Mean**: is the mean of all the *'Difference'* columns of the selected ACFDs. It indicates how much the dataset is ethical according to rules selected. Greater is the value, more the ACFDs have detected greater bias in the dataset.
- **Protected Attribute Difference Mean**: for each protected attribute p, it is the mean of *p-Difference* over all the selected ACFDs. It indicates how much the dataset is ethical over p according to dependencies selected. Greater is the value, more ACFDs have detected greater bias regarding the protected attribute in the dataset.

We decide to focus on these summarizing metrics because, as already explained in previous paragraphs, an ACFD has importance according to three main aspects: Support, Difference and Protected Attribute Difference. For these three aspects, using ACFDs selected by the user, we compute the mean value; thus, the *Cumulative Support* gives the pervasiveness of all the list of ACFDs, the *Difference Mean* gives a general overview over the fairness, and the *Protected Attribute Difference Mean* is the measure of fairness with respect to the protected attribute.

These summarizing metrics entirely depend on the specific ACFDs selected by the user, so the metrics could change according to the chosen dependencies and so, on the user needs.
Thus, according to the selected rules, the user can highlight different aspects of the dataset and leading to different conclusions.

Lastly, the user can see some examples of the tuples more involved by the selected ACFDs.
To extract these *"problematic tuples"* the framework performs some operations. During the computation of the metrics, we add to the original dataset an additional column *"marked"* for each tuple, that indicates how

many ACFDs are verified by the tuple. Thus, to select the "problematic tuples" we select the ones that are verified for more than M ACFDs, so that $marked > M$.
In the future, the user could correct the dataset according to the rules and the tuples more involved by the ACFDs. There are already developed techniques to solve bias in dataset based on the discovery of CFDs as [26], so our framework could be applied before this repair passage.

5.3.16 Running example: ACFDs User Selection and Scoring

Given the ranking in figure 5.16, in this last step, the user chooses N rules that are interesting according to her needs. In the running example, we choose $N = 15$ ACFDs over the total number that is 64.
Figure 5.17 displays the final table composed by the chosen dependencies and a final scoring outline.

The total number of tuples involved by the ACFDs is 13296 and the total number of tuples in the dataset is 30169; this means that the *Cumulative Support* is 0.44. The 44% of the dataset is covered by the chosen dependencies, so almost half of the dataset.
Looking at the Difference metrics: the *Difference Mean* is 0.16 that is meaningful, in fact, we can notice for all ACFDs the difference is above 0.1 and there are some dependencies that have difference metric above 0.2. These last rules have a meaningful difference, but their support is very low, so the tuples involved by them are few.
Finally, the *p-Difference Mean* metrics are pretty low, because there are many ACFDs with low or negative values.

As the table reports and looking also at the ACFDs mentioned in previous sections, the groups more discriminated are: 'Female', 'Black', 'NC-Hispanic' and from this last ranking also 'Amer-Indian-Eskimo'. Instead, the groups that have more privileges are: 'Male', 'White', 'NC-White', 'NC-Asian-Pacific', 'NC-Non-Hisp-White'.

Finally, the user could also see tuple examples involved by the dependencies. She can decide only the most problematic ones or see all of them, in order to correct the dataset in the future.
To select the *"problematic" tuples* we choose the ones that are verified for more than *M=3* ACFDs. Figure 5.18 displays the total number of tuples

Figure 5.17: Table with chosen final rules

Number of tuples interested by the rules: 13296 . Total number of tuples: 30169

Cumulative Support: 0.4407172925851039 . Difference Mean: 0.1695548087719456

Race-Difference Mean: 0.08669383850171641 . Sex-Difference Mean: 0.03890844137846591 . Native Country- Difference Mean: 0.05854767971034243

Total number of ACFDs selected: 15

	Rule	Support	Confidence	Diff	RaceDiff	SexDiff	NativeCountryDiff	Mean
16	{'lhs': {'sex': 'Female'}, 'rhs': {'income': '<=50K'}}	0.287447	0.886345	0.135210	0	0.13521	0.000000	0.211329
32	{'lhs': {'sex': 'Female', 'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}	0.262190	0.885382	0.134246	0	0.139644	-0.000963	0.198218
154	{'lhs': {'sex': 'Female', 'race': 'White'}, 'rhs': {'income': '<=50K'}}	0.229540	0.877026	0.125891	-0.00931871	0.140694	0.000000	0.177716
111	{'lhs': {'native-country': 'NC-Asian-Pacific', 'race': 'Other'}, 'rhs': {'income': '>50K'}}	0.000099	0.500000	0.251135	0.184022	0	0.409091	0.125617
31	{'lhs': {'age-range': '45-60', 'income': '>50K'}, 'rhs': {'sex': 'Male'}}	0.084159	0.877636	0.165426	0	0	0.000000	0.124793
152	{'lhs': {'race': 'Amer-Indian-Eskimo', 'native-country': 'NC-Asian-Pacific'}, 'rhs': {'income': '<=50K'}}	0.000066	1.000000	0.248865	0.315978	0	0.118881	0.124466
150	{'lhs': {'race': 'Amer-Indian-Eskimo', 'native-country': 'NC-Non-Hisp-White'}, 'rhs': {'income': '<=50K'}}	0.000033	1.000000	0.248865	0.316199	0	0.118881	0.124449
156	{'lhs': {'sex': 'Female', 'race': 'Black'}, 'rhs': {'income': '<=50K'}}	0.043588	0.939286	0.188150	0.0529407	0.069119	0.000000	0.115869
34	{'lhs': {'sex': 'Female', 'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.015181	0.962185	0.211050	0	0.0540286	0.075840	0.113115
160	{'lhs': {'sex': 'Female', 'race': 'Other'}, 'rhs': {'income': '<=50K'}}	0.002751	0.954023	0.202888	0.0676779	0.0449321	0.000000	0.102819
2	{'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.043919	0.908156	0.157021	0	0	0.157021	0.100470
66	{'lhs': {'race': 'Black'}, 'rhs': {'income': '<=50K'}}	0.081309	0.870167	0.119031	0.119031	0	0.000000	0.100170
122	{'lhs': {'race': 'Black', 'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}	0.075839	0.869631	0.118496	0.123893	0	-0.000535	0.097168
28	{'lhs': {'age-range': '45-60', 'income': '<=50K'}, 'rhs': {'sex': 'Female'}}	0.058504	0.394855	0.107064	0	0	0.000000	0.082784
72	{'lhs': {'race': 'Amer-Indian-Eskimo'}, 'rhs': {'income': '<=50K'}}	0.008353	0.881119	0.129984	0.129984	0	0.000000	0.069168

involved by more than 3 ACFDs, and the first five tuples as example.

Figure 5.18: Table with some problematic tuples of the dataset

Problematic tuples: 1384

	workclass	race	sex	hours-per-week	native-country	income	age-range	education-degree	marked
111	Private	Black	Female	0-20	NC-White	<=50K	15-30	Assoc	5
132	Local-gov	Black	Female	21-40	NC-White	<=50K	45-60	HS-College	6
136	Private	Black	Female	21-40	NC-Hispanic	<=50K	30-45	HS-College	5
183	Local-gov	Black	Female	21-40	NC-White	<=50K	60-75	Bach	5
189	Private	Black	Female	21-40	NC-White	<=50K	30-45	Assoc	5

5.4 Minorities study

In this section we propose an additional workflow that could be done after the aforementioned process to examine more in deep the dataset, in particular when data contains different groups.

It is difficult to study the nature of imbalanced data. **Imbalanced data** typically refers to a problem (usually in classification tasks) where the classes are not represented equally. It means that the target class contains a much smaller number of instances of one class than the other classes, and this can deeply influence the application performance.

Looking at our framework, we use the *CFDDiscovery* algorithm that needs three parameters: support, confidence and maxSize. These parameters, in particular **support** and **confidence** are difficult to be tuned in order to catch the **dependencies that deal with imbalanced data**.
In fact, datasets could have a lot of groups, in particular minorities. A *minority* is a group with low cardinality, so by definition the user must set low support and low confidence to extract dependencies that involve them. Decreasing these two parameters usually implies a result that is composed by a lot of rules involving the majority group and, in some cases, by other rules involving the different minorities. The quality and the number of the ACFDs that affect the minorities depend only on the dataset distribution.

There are mainly two classes of methods to solve the problem of imbalanced data: ***Data level preprocessing methods*** *(that rely on transforming the original data to change the distribution of classes) and* ***methods modifying the algorithms*** [31, p.564].

The first strategy consists in transforming the data distribution, but in our research this means also changing the bias in the dataset, making impossible the detection of unfairness and discrimination.
According to the second branch of techniques, we could use another software, for example the one that finds rules involving infrequent datasets [25] cited in previous section 5.3.5.
We introduce another technique to solve the imbalanced problem that consists in using a modified version of the dataset and continuing using the framework and the *CFDDiscovery* algorithm as before.

We propose to analyze the relevant minorities separately, one at a time. The work [31] is very important to guide the research of minorities. To

verify whether minority examples can be observed in real-world datasets, the authors used visualization methods, which project multi-dimensional data points into the low-dimensional space such that the structural properties of the data are preserved.

In our research, projecting multi-dimensional data points to a low-dimensional space means changing the data, and thus might change the bias. Instead, the extraction of the groups can be done during the preprocessing phase. In fact, in the Data Exploration part, we recommended to plot the attribute values for every column, especially for the protected ones. The user can fix a **minimum support threshold** which establishes a minimum cardinality for a group to be studied as minority or not.
Obviously according to the precision that the user wants to achieve, she should decide if it is reasonable to study such minority or not.

In the next paragraph, we will present the results of our framework applied to a minority of the U.S. Census Adult dataset to observe possible dependencies and to examine more in deep the entire process.

5.4.1 Running Example: Minority study

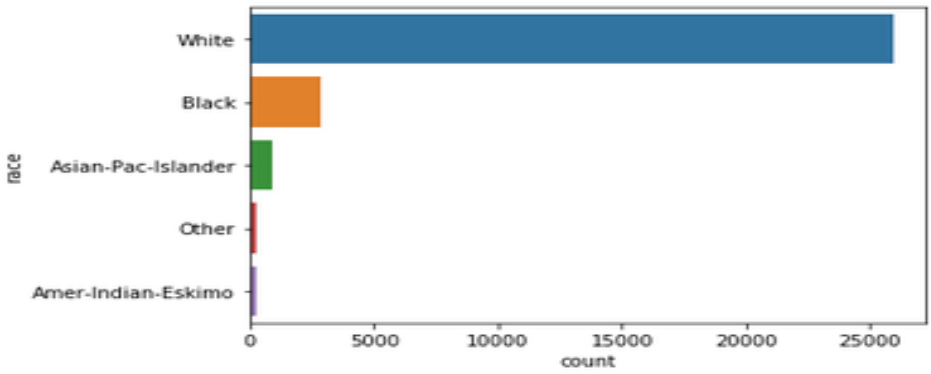
Continuing the study with the U.S. Census Adult dataset, we present an example of a study on a small group and we present the dependencies that we obtained from such group.
We will present the study on the **Black people** minority.

To start the research, we apply Data Visualization to the entire dataset computing Bar plots and Histograms, so that we are able to detect the meaningful minorities.

Figure 5.19 displays a Bar plot that displays the race distribution of the population. There is one majority group composed by *'White'* people and four minorities that are *'Black'*, *'Asian-Pac-Islander'*, *'Other'* and *'Amer-Indian-Eskimo'* people.
According to the cardinality of the groups we can establish to study all the minorities that contain at least 2500 tuples, so we can concentrate on the black and white population.

After the Data Preparation and Exploration, we decide to analyze only the black group, so we extract all the people that have the attribute "Race" equal to "Black". We suggest to perform Data Exploration also to the new

Figure 5.19: Bar Plot of "Race" attribute



dataset in order to build new hypothesis on the dataset.

We save the protected attributes and the target class, then we applied the *CFDDiscovery* algorithm with minimum support count = 200, so the minimum support is computed as $\frac{200}{2819} = 0.07$, the minimum confidence has value 0.86 and maximum antecedent size is equal to 2.

The output is composed by 86 dependencies, but only 74 are ACFDs, of which 52 are ACFDs that include at least one protected attribute and the target class.

Figure 5.20: First Selection phase

	Rule	Support	Confidence	Diff	RaceDiff	SexDiff	NativeCountryDiff
21	{'lhs': {'sex': 'Female'}, 'rhs': {'income': '<=50K'}}	0.466477	0.939286	0.069119	0	0.069119	0.000000
24	{'lhs': {'sex': 'Female', 'education-degree': 'Assoc'}, 'rhs': {'income': '<=50K'}}	0.163533	0.948560	0.076908	0	0.0769079	0.000000
25	{'lhs': {'sex': 'Female', 'education-degree': 'HS-College'}, 'rhs': {'income': '<=50K'}}	0.174175	0.976143	0.057122	0	0.0571224	0.000000
27	{'lhs': {'sex': 'Female', 'age-range': '30-45'}, 'rhs': {'income': '<=50K'}}	0.196878	0.923461	0.065580	0	0.0655798	0.000000
29	{'lhs': {'sex': 'Female', 'age-range': '45-60'}, 'rhs': {'income': '<=50K'}}	0.103938	0.907121	0.145145	0	0.145145	0.000000
30	{'lhs': {'sex': 'Female', 'hours-per-week': '21-40'}, 'rhs': {'income': '<=50K'}}	0.377794	0.946667	0.055632	0	0.0556322	0.000000
31	{'lhs': {'sex': 'Female', 'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}	0.436325	0.937500	0.067333	0	0.0678687	-0.001786
45	{'lhs': {'race': 'Black', 'sex': 'Female'}, 'rhs': {'income': '<=50K'}}	0.466477	0.939286	0.069119	0	0.069119	0.000000
50	{'lhs': {'workclass': 'Private', 'sex': 'Female'}, 'rhs': {'income': '<=50K'}}	0.353672	0.950429	0.053762	0	0.0537623	0.000000

At this point, we apply the selection criterion $Difference > minThreshold$, where the minimum threshold has value 0.05, obtaining 9 dependencies

present in figure 5.20.

From these 9 ACFDs we apply the combination process, then we perform again the selection phase using the criterion $Difference > minThreshold$, where the minimum threshold has the previous value, so it is 0.05, obtaining 23 ACFDs.
At this point, we select N=7 ACFDs. Figure 5.21 displays the final ranking.

Figure 5.21: Final Ranking, Black Minority

Number of tuples interested by the rules: 1596 . Total number of tuples: 2819

Cumulative Support: 0.5661582121319617 . Difference Mean: 0.11270938850675931

Race-Difference Mean: 0.0 . Sex-Difference Mean: 0.011705324459590468 . Native Country- Difference Mean: 0.005675007367681033

Total number of ACFDs selected: 7

	Rule	Support	Confidence	Diff	RaceDiff	SexDiff	NativeCountryDiff	Mean
1	{'lhs': {'sex': 'Female'}, 'rhs': {'income': '<=50K'}}	0.466477	0.939286	0.069119	0	0.069119	0	0.267798
36	{'lhs': {'sex': 'Male', 'native-country': 'NC-Non-Hisp-White'}, 'rhs': {'income': '>50K'}}	0.001419	0.400000	0.270167	0	0.114286	0.201973	0.135793
5	{'lhs': {'sex': 'Female', 'education-degree': 'Assoc'}, 'rhs': {'income': '<=50K'}}	0.163533	0.948560	0.076908	0	0.076908	0	0.120221
2	{'lhs': {'sex': 'Male'}, 'rhs': {'income': '>50K'}}	0.099681	0.198027	0.068194	0	0.068194	0	0.083937
6	{'lhs': {'sex': 'Male', 'education-degree': 'Assoc'}, 'rhs': {'income': '>50K'}}	0.031926	0.219512	0.091164	0	0.091164	0	0.061545
27	{'lhs': {'sex': 'Female', 'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.028024	0.963415	0.093248	0	0.070557	0.0241289	0.060636
38	{'lhs': {'sex': 'Male', 'native-country': 'NC-Asian-Pacific'}, 'rhs': {'income': '>50K'}}	0.000355	0.250000	0.120167	0	0.083333	0.0519732	0.060261

The total number of tuples involved by the dependencies is 1596, the total number of tuples analyzed is 2819, so the *Cumulative Support* is 0.57, that means that more than half of the dataset of black population is affected by discrimination. The *Difference Mean* has value of 0.11; in the *p-Difference Mean* metrics, the one relative to 'Sex', is above 0.1, meaning that the inequality between women and men is high also in the black group.

Let us highlight some ACFDs:

$$\phi_1 : (sex = Female) \rightarrow (income =\leq 50K).$$

$$\phi_2 : (sex = Male) \rightarrow (income => 50K).$$

It is meaningful that also in the black group the women gain less than 50'000 dollars/year with respect to men. Clearly group fairness is no respected in this group and also general subgroup fairness.
If we analyze more in deep the ACFDs, we see that also subgroup fairness

in this minority is not respected. In fact, also if women and men reach the same education degree the dependencies do not change:

$$\phi_1 : (sex = Female, education - degree = Assoc) \rightarrow (income =\leq 50K)$$

$$\phi_2 : (sex = Male, education - degree = Assoc) \rightarrow (income => 50K).$$

Another consideration can be formalized by looking at these dependencies:

$$\phi_{36} : (sex = Male, native{-}country = NC{-}Non{-}Hisp{-}White) \rightarrow (income => 50K)$$

$$\phi_{38} : (sex = Male, native{-}country = NC{-}Asian{-}Pacific) \rightarrow (income => 50K).$$

The ACFDs involving "NC-Non-Hisp-White" group and "NC-Asian-Pacific" group have low support, but they show again the same type of bias as one found in the analysis done for the general dataset, highlighting again these privileges with respect to other groups.

Finally, figure 5.21 presents the some *"problematic tuples"* that are valid for more than *M=2* rules.

Figure 5.22: Problematic Tuples

Problematic tuples: 34

	workclass	race	sex	hours-per-week	native-country	income	age-range	education-degree	marked
115	Private	Black	Male	21-40	NC-Non-Hisp-White	>50K	30-45	Assoc	3
405	Private	Black	Female	21-40	NC-Hispanic	<=50K	45-60	Assoc	3
445	Private	Black	Female	21-40	NC-Hispanic	<=50K	45-60	Assoc	3
453	Private	Black	Female	21-40	NC-Hispanic	<=50K	15-30	Assoc	3
470	Private	Black	Female	21-40	NC-Hispanic	<=50K	30-45	Assoc	3

5.5 Titanic study

This section presents an overview of the framework applied to another dataset: the **Titanic**[3]. This dataset presents a list of passengers on Titanic, a British passenger liner operated by the 'White Star Line' that sank in the North Atlantic Ocean in the early morning hours of 15 April 1912, after striking an iceberg during her maiden voyage from Southampton to New York City.
Of the estimated 2,224 passengers and crew aboard, more than 1,500 died, making the sinking one of modern history's deadliest peacetime commercial marine disasters.

The version that we considered contains 891 samples and 12 attributes, 8 of which are categorical and 4 are numerical. The target variable *'Survived'* is a binary categorical variable. The main task for which this dataset is used for is to predict if a person survived the famous disaster, based on other features like sex and age and whether the passenger travelled alone or not.
We now provide a brief description of the attributes. We highlight that we refer to the values actually contained in this specific dataset, without any further assumption.

- **PassengerId**: a categorical variable representing the Id of the passenger.
- **Survived**: a categorical binary variable representing if the passenger survived (1) or not (0).
- **Pclass**: a categorical variable representing if the passenger was in the first class (1), in the second class (2) or in the third class (3).
- **Name**: a categorical variable containing the title and the name of a person.
- **Sex**: a categorical variable representing the sex of the passenger. The only possible values are male and female.
- **Age**: a numerical variable representing the age of the passenger. The values belong to the range [0.42,80.0]. It has to be noticed that the age of children with less than one year was reported as a float number instead of an integer.

[3]https://www.kaggle.com/c/titanic

- **SibSp**: a numerical variable representing the number of siblings and eventually the spouse who embarked with the passenger. The values belong to the range [0,8].
- **Parch**: a numerical variable representing the number of parents and children who embarked with the passenger. The values belong to the range [0,6].
- **Ticket**: a categorical variable representing the ticket code of the passenger.
- **Fare**: a numerical variable representing the price of the ticket owned by the passenger. The values belong to the range [0.0,512.3292].
- **Cabin**: a categorical variable representing the cabin of the passenger. Many of them refer to similar sections of the ship.
- **Embarked**: a categorical variable representing the port of embarkation of the passenger. The domain of the attribute contains the three values: S = Southampton (72%), C = Cherbourg (18.9%) and Q = Queenstown (9.1%).

Figure 5.23 displays the first tuples of the dataset.

Figure 5.23: Titanic Dataset

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

5.5.1 Data Preparation and Exploration

Before being able to gather useful insights from this dataset we had to perform some preprocessing operations.
After Data Acquisition, because there is only one source we do not need to perform Data Integration and so we can focus on **Data Cleaning**.

As we can notice from Summary Statistics (figures 5.24 and 5.4), three attributes contain some missing values. In particular, there are 177 missing

Figure 5.24: Titanic Summary Statistics Numerical Attributes

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Figure 5.25: Titanic Summary Statistics Categorical Attributes

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	McEvoy, Mr. Michael	male	1601	B96 B98	S
freq	1	577	7	4	644

values for the attribute *'Age'*, 687 for the attribute *'Cabin'* and 2 for the attribute *'Embarked'*, for a total of 866 missing values (8% of the total number of cells). The total number of rows containing at least one missing value is 708 (about 79.5% of the total number of samples). This percentage is extremely high, hence we choose to perform the following operations.
Since it would not have been wise to research dependencies on the full dataset with so many missing information we choose to remove *'Age'* and *'Cabin'* columns from our analysis.
For the *'Embarked'* column, since the missing values are fewer, we decide to use the mode to substitute the missing values, so we impute the missing values substituting the most frequent value that is 'S' (Southampton).

In **Feature Selection** phase, we notice that the attributes *'PassengerId'*, *'Name'* and *'Ticket'* don not provide useful information, hence we choose to remove them, reducing Data Dimensionality.
Moreover, there are 15 rows that contain people that paid 0 dollar; they represent people that are in the Titanic crew or assistant of people of first class. To not change bias in resulting dependencies, we decide to remove these rows.
Therefore, our final dataset contains 876 rows and 7 features.

After that, we perform **Data Discretization** on the numerical attribute

'Fare', grouping it into 5 bins: *'0-8','9-20','21-40','41-80','81-500'*.
Finally, from the dataset, we can easily understand that *'Fare'*, *'Class'* and *'Embarked'* are correlated. The user can also delete two of these attributes to avoid obvious rules, we decided to keep them in order to understand the relation between these attributes and the other ones in the dataset.

Figure 5.26 displays the dataset after the preprocessing phase.

Figure 5.26: Titanic Dataset after Preprocesssing phase

	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked
0	0	3	male	1	0	0-8	S
1	1	1	female	1	0	41-80	C
2	1	3	female	0	0	0-8	S
3	1	1	female	1	0	41-80	S
4	0	3	male	0	0	9-20	S

During this phase we also perform **Data Visualization**.
Figure 5.27 displays the distribution of the target class *'Survived'*, and, as we can notice, the majority of people did not survived.

Figure 5.27: 'Survived' attribute BarPlot

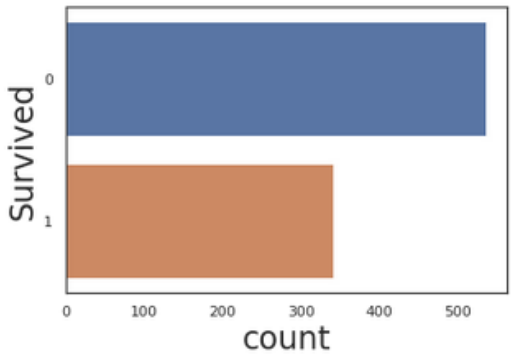
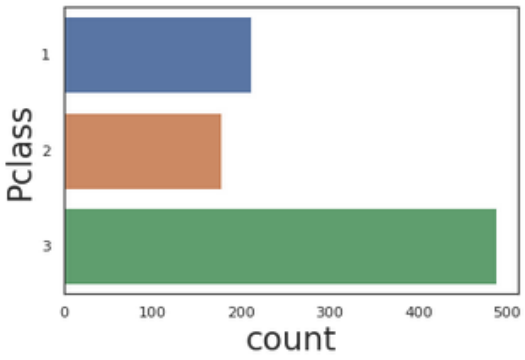


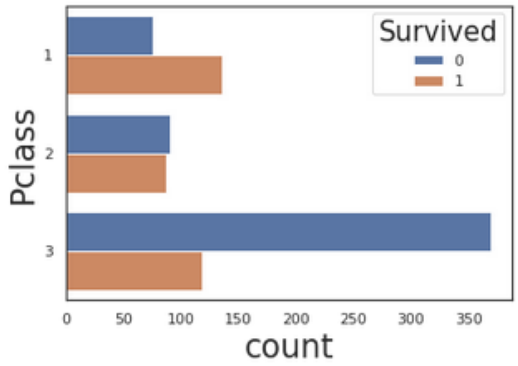
Figure 5.28 presents the people distribution over the three classes: the first class, the second class and the third class. It is interesting to see that the second class has fewer people than the first one, and the third class hosts the majority of people.

Figure 5.28: 'Pclass' attribute BarPlot



In last plot 5.29, we present the attributes *'Pclass'* and *'Survived'*. From this figure, we can notice that in the third class the majority of people did not survived, instead in the first class more than half people survived.

Figure 5.29: 'Pclass' and 'Survived' attributes BarPlot



It is also interesting to see the relation between the attributes *'Fare'* and *'Embarked'*. The figure 5.30 indicates the port from which people depart: 'S' stays for Southampton, 'C' is Cherbourg, 'Q' is Queenstown. It is interesting that from the city of Queenstown people paid the lowest rates. In fact, if we see figure 5.31, we can easily see that almost none of the people coming from Queenstown is in first class.

Figure 5.30: 'Fare' and 'Embarked' attributes BarPlot

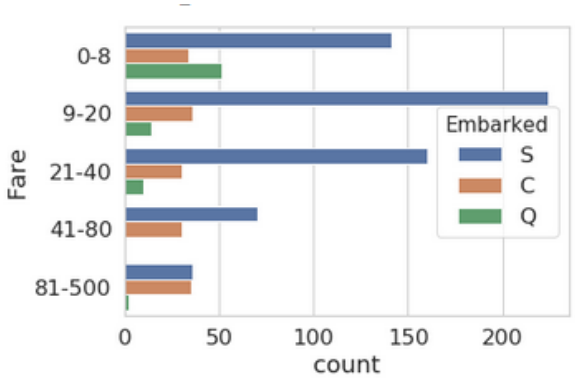
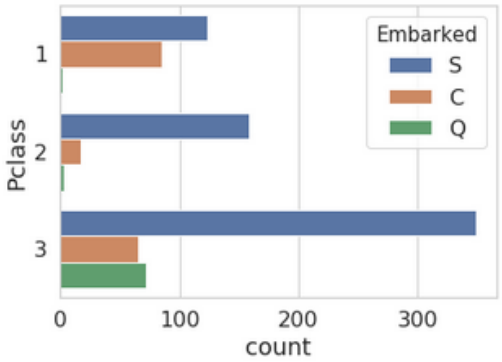


Figure 5.31: 'Fare' and 'Pclass' attributes BarPlot



5.5.2 CFDDiscovery

At this stage, we establish the protected attributes *'Pclass'*, *'Sex'* and *'Embarked'* and the target class *'Survived'*. We choose as protected attribute *'Embarked'* instead of *'Fare'*, because *'Fare'* is strictly correlated to attribute *'Pclass'*, so analyzing the last attribute, we also evaluate the attribute *'Fare'*. Furthermore, *'Pclass'* and *'Embarked'* are less correlated to each other than *'Embarked'* and *'Fare'*, so decided to keep *'Embarked'* and to not consider *'Fare'* as protected.

Figure 5.32 displays some dependencies extracted from the prepared Titanic dataset *'D'*. The figure reports the first ACFDs in which two of them are AFDs and the others are ACFDs.

To obtain such rules we set the parameters as:

- minSupport threshold: $\delta = 0.09$

- minConfidence threshold $\epsilon = 0.85$
- maxSize or maximum antecedent size $\alpha = 2$

All the obtained ACFDs ϕ: $(X \rightarrow Y, t_p)$ satisfy these constraints:

- $\text{support}(\phi, D) \geq \delta = 0.09$
- $\text{confidence}(\phi, D) \geq \epsilon = 0.85$
- $|X| \geq \alpha = 2$.

The total number of obtained rules in this step is 18.

Figure 5.32: CFDDiscovery Output

```
Dependency n. 0 :  (SibSp, Survived=0) => Parch
Dependency n. 1 :  (Survived=0, SibSp=0) => Parch=0
Dependency n. 2 :  (Survived=0, Fare=0-8) => SibSp=0
Dependency n. 3 :  (Survived=0, Fare=9-20) => Parch=0
Dependency n. 4 :  (Survived=0, Fare=0-8) => Parch=0
Dependency n. 5 :  (Survived=0, Pclass=2) => Parch=0
Dependency n. 6 :  (Survived=0, Fare=0-8) => Pclass=3
Dependency n. 7 :  (Survived=0, Fare=9-20) => Embarked=S
Dependency n. 8 :  (Survived=1, Pclass=2) => Embarked=S
Dependency n. 9 :  (Survived=0, Pclass=2) => Embarked=S
Dependency n. 10 :  (SibSp, Survived=0) => Sex
```

5.5.3 ACFDs Filtering

Given the list of dependencies, in this step we filter the ACFDs and we create a structure in which organize and save the dependencies. The main phases are:

1. The **use of the parser** to divide every dependency extracting the LHS and RHS and then each couple of attribute-value.
2. **Filter the ACFDs**. Discard all the approximate FDs and keep only the approximate CFDs. Furthermore, check if in each rule appears at least one protected attribute and the class variable.
3. Given the parsed ACFDs, **build the dictionary** that contains all the dependencies.

In figure 5.33 we present the final appearance of the first ACFDs in the dictionary. The dictionary contains a total of 14 ACFDs.

Figure 5.33: Dictionary Output

```
CFD n. 0 :  {'lhs': {'Survived': '0', 'SibSp': '0'}, 'rhs': {'Parch': '0'}}
CFD n. 1 :  {'lhs': {'Survived': '0', 'Fare': '0-8'}, 'rhs': {'SibSp': '0'}}
CFD n. 2 :  {'lhs': {'Survived': '0', 'Fare': '9-20'}, 'rhs': {'Parch': '0'}}
CFD n. 3 :  {'lhs': {'Survived': '0', 'Fare': '0-8'}, 'rhs': {'Parch': '0'}}
CFD n. 4 :  {'lhs': {'Survived': '0', 'Pclass': '2'}, 'rhs': {'Parch': '0'}}
CFD n. 5 :  {'lhs': {'Survived': '0', 'Fare': '0-8'}, 'rhs': {'Pclass': '3'}}
CFD n. 6 :  {'lhs': {'Survived': '0', 'Fare': '9-20'}, 'rhs': {'Embarked': 'S'}}
CFD n. 7 :  {'lhs': {'Survived': '1', 'Pclass': '2'}, 'rhs': {'Embarked': 'S'}}
CFD n. 8 :  {'lhs': {'Survived': '0', 'Pclass': '2'}, 'rhs': {'Embarked': 'S'}}
CFD n. 9 :  {'lhs': {'Survived': '0', 'SibSp': '0'}, 'rhs': {'Sex': 'male'}}
CFD n. 10 :  {'lhs': {'Survived': '0', 'Fare': '0-8'}, 'rhs': {'Sex': 'male'}}
```

5.5.4 ACFDs Selection

After the dictionary creation, we can start investigating on the ACFDs. In the list appears all rules that satisfy the aforementioned criteria, so we have to select the ones that show discrimination and bias.
We start by computing a table that contains all the ACFDs and the metrics. After that, we select all the rules that have the difference metric above the minimum threshold fixed as 0.1, obtaining 12 dependencies.
From this first selection, we can build table 5.34 that contains the 12 ACFDs ordered by "Mean" option.
For each dependency we compute the support, confidence, difference and, for each protected attribute p, the *p-Difference*.

Figure 5.34: Table ordered using 'Mean' option

	Rule	Support	Confidence	Diff	Score	PclassDiff	SexDiff	PortDiff
0	{'lhs': {'Survived': '0', 'Pclass': '2'}, 'rhs': {'Parch': '0'}}	0.091324	0.879121	0.122272	0	0.073513	0	0
1	{'lhs': {'Survived': '0', 'Fare': '0-8'}, 'rhs': {'Pclass': '3'}}	0.198630	0.994286	-0.001290	0	0.000000	0	0
2	{'lhs': {'Survived': '0', 'Fare': '9-20'}, 'rhs': {'Embarked': 'S'}}	0.180365	0.863388	0.045870	0	0.000000	0	0
3	{'lhs': {'Survived': '1', 'Pclass': '2'}, 'rhs': {'Embarked': 'S'}}	0.086758	0.873563	0.153244	0	0.234267	0	0
4	{'lhs': {'Survived': '0', 'Pclass': '2'}, 'rhs': {'Embarked': 'S'}}	0.093607	0.901099	0.180779	0	0.129136	0	0
5	{'lhs': {'Survived': '0', 'SibSp': '0'}, 'rhs': {'Sex': 'male'}}	0.396119	0.903646	0.197069	0	0.000000	0	0
6	{'lhs': {'Survived': '0', 'Fare': '0-8'}, 'rhs': {'Sex': 'male'}}	0.182648	0.914286	0.126675	0	0.000000	0	0
7	{'lhs': {'Sex': 'male', 'Fare': '0-8'}, 'rhs': {'Survived': '0'}}	0.182648	0.898876	0.124540	0	0.000000	0.12454	0
8	{'lhs': {'Survived': '0', 'Sex': 'male'}, 'rhs': {'Parch': '0'}}	0.445205	0.859031	0.102182	0	0.000000	0.0534234	0
9	{'lhs': {'Survived': '0', 'Parch': '0'}, 'rhs': {'Sex': 'male'}}	0.445205	0.904872	0.197482	0	0.000000	0	0
10	{'lhs': {'Survived': '0', 'Pclass': '2'}, 'rhs': {'Sex': 'male'}}	0.097032	0.934066	0.292513	0	0.085468	0	0
11	{'lhs': {'Survived': '0', 'Sex': 'female'}, 'rhs': {'Pclass': '3'}}	0.082192	0.888889	0.332953	0	0.000000	0.199169	0

This first table is very useful to get an idea of which are the dependencies that show bias.
At this point, the user could select the ACFDs that show bias according to her needs or she could continue the process going to the last phases of the framework.
We suggest to continue the research computing all the possible combinations.

5.5.5 ACFDs Completion and ACFDs Ranking

In the last part of the framework, there are three main steps:

1. For each dependency, compute the **possible combinations** over the protected attributes and the target class.

2. **Select** the obtained rules and **rank** them.

3. Let the user selecting the most interesting rules and compute **final metrics**.

Figure 5.35 displays for one example rule, all the possible combinations.

Figure 5.35: CFD combination process

```
Originary CFD:  {'lhs': {'Pclass': '1', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}
CFD n.  0 : {'lhs': {'Pclass': '3', 'Sex': 'male'}, 'rhs': {'Survived': '0'}}
CFD n.  1 : {'lhs': {'Pclass': '3', 'Sex': 'male'}, 'rhs': {'Survived': '1'}}
CFD n.  2 : {'lhs': {'Pclass': '3', 'Sex': 'female'}, 'rhs': {'Survived': '0'}}
CFD n.  3 : {'lhs': {'Pclass': '3', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}
CFD n.  4 : {'lhs': {'Pclass': '1', 'Sex': 'male'}, 'rhs': {'Survived': '0'}}
CFD n.  5 : {'lhs': {'Pclass': '1', 'Sex': 'male'}, 'rhs': {'Survived': '1'}}
CFD n.  6 : {'lhs': {'Pclass': '1', 'Sex': 'female'}, 'rhs': {'Survived': '0'}}
CFD n.  7 : {'lhs': {'Pclass': '1', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}
CFD n.  8 : {'lhs': {'Pclass': '2', 'Sex': 'male'}, 'rhs': {'Survived': '0'}}
CFD n.  9 : {'lhs': {'Pclass': '2', 'Sex': 'male'}, 'rhs': {'Survived': '1'}}
CFD n.  10 : {'lhs': {'Pclass': '2', 'Sex': 'female'}, 'rhs': {'Survived': '0'}}
CFD n.  11 : {'lhs': {'Pclass': '2', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}
```

The total number of obtained combinations from all the ACFDs is 110.
For each combination rule, we compute the metrics and we select the interesting ones using the previous constraint: $Difference > minThreshold$.
We decided to keep the same value of previous step for the minimum threshold that is 0.1 and we obtain 39 ACFDs.
After that, we rank the rules using the *'Mean'* criterion ordering the ACFDs in decreasing order. Figure 5.36 displays the first part of the ranking list of

the selected combination rules.

Figure 5.36: Ranking of ACFDs after combination process ordered by 'Mean' option

	Rule	Support	Confidence	Diff	Score	PclassDiff	SexDiff	PortDiff	Mean
105	{'lhs': {'Pclass': '1', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}	0.103881	0.968085	0.578816	0	0.226047	0.323535	0	0.341348
93	{'lhs': {'Pclass': '1', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}	0.103881	0.968085	0.578816	0	0.226047	0.323535	0	0.341348
58	{'lhs': {'Survived': '0', 'Parch': '0'}, 'rhs': {'Sex': 'male'}}	0.445205	0.904872	0.197482	0	0.000000	0	0	0.321344
109	{'lhs': {'Pclass': '2', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}	0.079909	0.921053	0.531783	0	0.179014	0.432289	0	0.305846
97	{'lhs': {'Pclass': '2', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}	0.079909	0.921053	0.531783	0	0.179014	0.432289	0	0.305846
86	{'lhs': {'Pclass': '3', 'Sex': 'male'}, 'rhs': {'Survived': '0'}}	0.339041	0.865889	0.255159	0	0.058060	0.108189	0	0.297100
98	{'lhs': {'Pclass': '3', 'Sex': 'male'}, 'rhs': {'Survived': '0'}}	0.339041	0.865889	0.255159	0	0.058060	0.108189	0	0.297100
42	{'lhs': {'Survived': '0', 'SibSp': '0'}, 'rhs': {'Sex': 'male'}}	0.396119	0.903646	0.197069	0	0.000000	0	0	0.296594
54	{'lhs': {'Survived': '0', 'Sex': 'male'}, 'rhs': {'Parch': '0'}}	0.445205	0.859031	0.102182	0	0.000000	0.0534234	0	0.273694
61	{'lhs': {'Survived': '1', 'Parch': '0'}, 'rhs': {'Sex': 'female'}}	0.174658	0.659483	0.366873	0	0.000000	0	0	0.270765
73	{'lhs': {'Survived': '1', 'Pclass': '2'}, 'rhs': {'Sex': 'female'}}	0.079909	0.804598	0.446150	0	0.121313	0	0	0.263029
45	{'lhs': {'Survived': '1', 'SibSp': '0'}, 'rhs': {'Sex': 'female'}}	0.156393	0.655502	0.362079	0	0.000000	0	0	0.259236

5.5.6 ACFDs User Selection and Scoring

Last step consists in performing a selection of the N most interesting rules. We select N=6 dependencies and figure 5.37 displays the final ranking.

Figure 5.37: Final ranking of user selected ACFDs ordered by "Mean"

Number of tuples interested by the rules: 647 . Total number of tuples: 876

Cumulative Support: 0.7385844748858448 . Difference Mean: 0.35437190832094806

PClass diff mean: 0.0018055850473224606 . Sex diff mean: 0.00500473096013565 . Port diff mean: 0.0

Total number of ACFDs selected: 6

	Rule	Support	Confidence	Diff	Score	PclassDiff	SexDiff	PortDiff	Mean
93	{'lhs': {'Pclass': '1', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}	0.103881	0.968085	0.578816	0	0.226047	0.323535	0	0.341348
97	{'lhs': {'Pclass': '2', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}	0.079909	0.921053	0.531783	0	0.179014	0.432289	0	0.305846
86	{'lhs': {'Pclass': '3', 'Sex': 'male'}, 'rhs': {'Survived': '0'}}	0.339041	0.865889	0.255159	0	0.058060	0.108189	0	0.297100
77	{'lhs': {'Survived': '0', 'Sex': 'female'}, 'rhs': {'Pclass': '3'}}	0.082192	0.888889	0.332953	0	0.000000	0.199169	0	0.207572
69	{'lhs': {'Survived': '1', 'Pclass': '3'}, 'rhs': {'Sex': 'female'}}	0.082192	0.610169	0.251722	0	-0.073115	0	0	0.166957
81	{'lhs': {'Survived': '1', 'Sex': 'male'}, 'rhs': {'Pclass': '1'}}	0.051370	0.416667	0.175799	0	0.000000	0.0178397	0	0.113584

From the table 5.37 we choose some meaningful dependencies:

$$\phi_{93} : (Pclass = 1, Sex = Female) \rightarrow (Survived = 1)$$

$$\phi_{97} : (Pclass = 2, Sex = Female) \rightarrow (Survived = 1)$$

$$\phi_{77} : (Survived = 0, Sex = Female) \rightarrow (Pclass = 3).$$

From the first two ACFDs and, in particular looking at the *Sex-Difference* metric, we can tell that women have a larger probability of surviving with respect to men, but this is partly true until the class changes. From ACFD number 77 we can infer that if a woman did not survive, she is in third class with 0.88 value of confidence. The difference metric is pretty high, it has values 0.33, so the dependency shows a relevant discrimination in the dataset.

Analyzing other ACFDs as:

$$\phi_{86} : (Pclass = 3, Sex = Male) \rightarrow (Survived = 0)$$

$$\phi_{81} : (Survived = 1, Sex = Male) \rightarrow (Pclass = 1)$$

we can understand that previous consideration about the classes for women could be valid also for men. If a man is in the third class with a confidence

of 0.86 he will not survive, instead if he is in the first class he has an higher probability of surviving.
From these ACFDs and from the final metrics we can state that the Titanic does not respect group fairness between women and men, and also the subgroup fairness with respect to classes.

Finally, in figure 5.38 there are some examples of tuples that are problematic, in the sense that at least one dependency verifies it.

Figure 5.38: Problematic tuples

Problematic tuples: 647

	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked	marked
0	0	3	male	1	0	0-8	S	1
1	1	1	female	1	0	41-80	C	1
2	1	3	female	0	0	0-8	S	1
3	1	1	female	1	0	41-80	S	1
4	0	3	male	0	0	9-20	S	1

Chapter 6

Experimental Results

6.1 Introduction

In this section we present a comparison between our framework, *FAIR-DB* and the one presented in paper *"Nutritional Labels for Data and Models"* [37] called *Ranking Facts*. We start presenting this tool, then with two concrete examples, we enumerate the main differences and similarities between this work and our framework.

6.2 Ranking Facts Basics

In paper *"Nutritional Labels for Data and Models"* [37] the authors J. Stoyanovich and B. Howe developed a tool based on the concept of a **Nutritional Labels**, drawing an analogy to the food industry, where simple, standard labels convey information about the ingredients and production processes.
Nutritional labels are derived, automatically or semi-automatically, as part of the complex process that gave rise to the data or model they describe, embodying the paradigm of interpretability-by-design.
Fairness measures are based on a generative process for rankings, sorted lists of items, that meet a particular fairness criterion and are drawn from a dataset with a given proportion of members of a binary protected group.

The final developed system is called **Ranking Facts** and it automatically derives nutritional labels for rankings.
The tool is based on the idea of *algorithmic ranker* that takes a collection of items as input and produce a ranking as output. The simplest kind of ranker is a **score-based ranker** which computes a score for each item independently and then sorts the items according to their scores. The concept

of fairness is related to rankers because these tools can discriminate against individuals and protected groups. Furthermore, ranked results are often **unstable**, that means that small changes in the input may lead to drastic changes in the output.
The authors developed this tool based on the concept of nutritional labels, in which labels convey information about the algorithm chosen and the attributes that are involved by it.

6.3 Ranking Facts Tool

Ranking Facts is a collection of visual widgets that are based on stability, fairness and diversity concepts.
The tool needs numerical attributes to compute the score-based ranker, and so to compute the widgets.

The two starting widgets that explain the ranking methodology are: the **Recipe** and the **Ingredients**.
The *Recipe* widget describes the ranking algorithm while the *Ingredients* widget lists attributes used in ranking in order of importance.

The following widget is the **Stability** that explains whether the ranking methodology is robust on the particular chosen dataset.

Fairness widget quantifies whether the ranked output exhibit statistical parity (a particular definition of group fairness very similar to subgroup fairness cited in chap. 4.2) with respect to one or more protected attributes. The notion of fairness is defined specifically for rankings and it can be computed comparing only binary categorical attributes.

The last widget is **Diversity** and it ensures that different kinds of objects are represented in the output of the algorithmic process.

Stability and Diversity can be analyzed from two different aspects: **top-10** and **over-all**. Applying the first aspect, the tool shows the results analyzing only the first 10 items of the ranking, instead using the second one, the result is computed using all items appearing in the ranking.

6.4 Ranking Facts Experiment

6.4.1 U.S. Census Adult dataset

In this paragraph, *Ranking Facts* is applied to the U.S. Census Adult dataset. We do not use the online demo because the dataset has too many tuples (more than 30'000) and the demo would return a time-out error, thus, we opt for the notebook version[4] using a similar version of the U.S. Census Adult dataset.
In this version, very similar to the one used in our previous analysis, we do not discretize attributes *'Age'*, *'Education-num'* and *'Hours-per-week'*, so that the algorithm could use these attributes to compute the **score-based ranker**. Thus, to specify the **ranking function** we choose the numerical attributes: *'Age'*, *'Education-num'* and *'Hours-per-week'*.
For the **fairness check** the algorithm needs at least one binary attribute, so we choose *'Sex'*.
For the **diversity check** we choose three categorical attributes that are *'Sex'*, *'Race'* and *'Native-country'*.

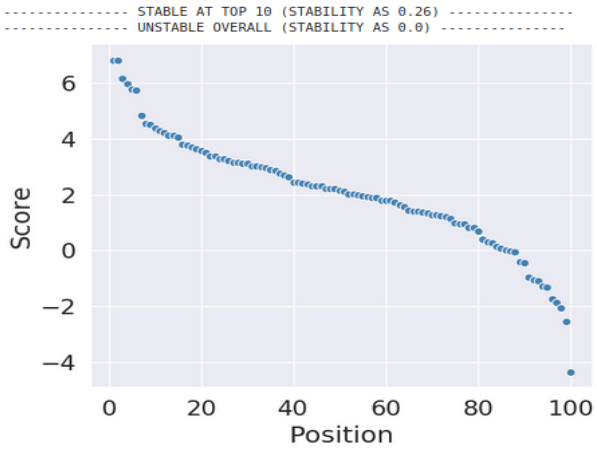
After having established the input parameters, we apply the algorithm to the dataset. Using the notebook version we do not have all the widgets, but we can still present a meaningful analysis.

Firstly, we create the ranking function using the three numerical attributes: *'Age'*, *'Education-num'* and *'Hours-per-week'*. After that, we analyze the **Stability** of the ranking function as figure 6.1 reports.
The plot displays an *unstable ranking*, so slight changes to the data (e.g., due to uncertainty and noise), or to the methodology (e.g., by slightly adjusting the weights in a score-based ranker) could lead to a significant change in the output.

The second point of the analysis is centered on **Fairness**.
Fairness measures analyze one binary attribute at the time, computing a statistical test to verify the **group fairness** with three different definitions: *FA*IR, pairwise comparison* and *proportion*.
A ranking is considered unfair when the p-value of the corresponding statistical test falls below 0.05.
Firstly, we compute the fairness measure using the binary attribute *'Sex'*. For both three measures, the result indicates that group fairness is verified

[4] https://github.com/DataResponsibly/RankingFacts

Figure 6.1: Stability plot



for *'Male'* group and it is not verified for *'Female'* group. Figure 6.2 shows the result of this check for the first measure of fairness ($FA*IR$), similar outcomes are printed for the *pairwise comparison* and *proportion* notions. This result on group fairness over the attribute *'Sex'* was also confirmed by our framework analysis using the U.S. Census Adult dataset.

Figure 6.2: 'Sex' Fairness test

```
---------- GROUP FAIRNESS VERIFICATION CONSIDERING sex ----------
---------- FAIR FOR Male (p=0.9998385193208591, alpha=0.79) ----------
---------- UNFAIR FOR Female (p=0.00040043116055571307, alpha=0.81) FAIL AT RANK POSITION 17 ---------
```

In the analysis of this dataset with our framework, we computed fairness measures for other two categorical attributes: *'Race'* and *'Native-country'*. These two attributes are not binary, so to perform the fairness check with *Ranking Facts* is necessary Data Preprocessing. After that, we apply the tool on these two attributes obtaining the following results.

Figure 6.3: 'Race' Fairness test

```
---------- GROUP FAIRNESS VERIFICATION CONSIDERING race ----------
---------- FAIR FOR White (p=1.0, alpha=0.05) ----------
---------- FAIR FOR Asian-Pac-Islander (p=1.0, alpha=0.05) ----------
---------- UNFAIR FOR Black (p=0.0, alpha=0.05) ----------
---------- UNFAIR FOR Other (p=0.0, alpha=0.05) ----------
---------- UNFAIR FOR Amer-Indian-Eskimo (p=0.01, alpha=0.05) ----------
```

Figure 6.3 and 6.4 display the outputs of the *pairwise comparison* analyzing respectively the attributes *'Race'* and *'Native-country'*. Similar results

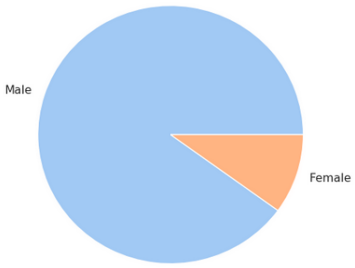
Figure 6.4: 'Native-country' Fairness test

```
---------- GROUP FAIRNESS VERIFICATION CONSIDERING native-country ----------
---------- FAIR FOR NC-White (p=1.0, alpha=0.05) ----------
---------- UNFAIR FOR NC-Hispanic (p=0.0, alpha=0.05) ----------
---------- FAIR FOR NC-Non-Hisp-White (p=1.0, alpha=0.05) ----------
---------- FAIR FOR NC-Asian-Pacific (p=1.0, alpha=0.05) ----------
```

can be derived also for the other two measures of fairness.
As before, for these two attributes, the results obtained with *Ranking Facts* are in accordance to the ones obtained with our framework.

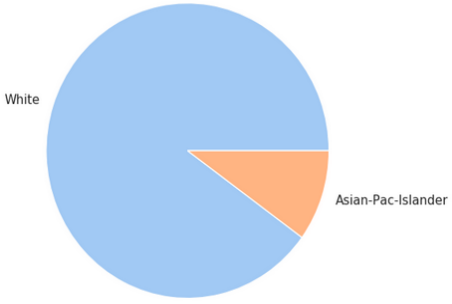
Finally, the last aspect that this tool analyzes is the **Diversity**. We compute it for three categorical attributes: *'Sex'*, *'Race'* and *'Native-country'*. We report for each one the diversity check at *top-10* level.

Figure 6.5: 'Sex' Diversity widget



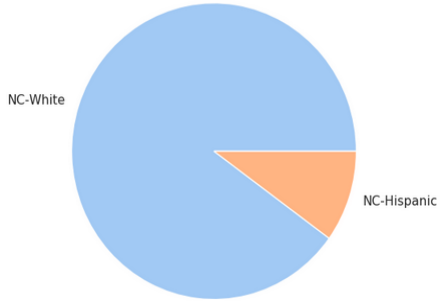
The figures 6.5, 6.6 and 6.7 confirmed the aforementioned fairness results. Analyzing the plot 6.5, the *'Male'* group is predominant in the ranking, in fact the women did not pass the fairness check.

Figure 6.6: 'Race' Diversity widget



In the *'Race'* Diversity widget the main groups are *'White'* and *'Asian-Pac-Islander'*, so the *'Black'*, *'Other'* and *'Amer-Indian-Eskimo'* are discrim-

Figure 6.7: 'Native-country' Diversity widget



inated.

Finally, in figure 6.7 the *'NC-White'* group is predominant over the *'NC-Hispanic'* group.

To compare the results, we present the final outcome of our framework applied to the U.S. Census Adult dataset. In figure 6.8, the more discriminated groups are: 'Female', 'Black', 'NC-Hispanic' and also 'Amer-Indian-Eskimo'. Instead, the groups that have more privileges are: 'Male', 'White', 'NC-White', 'NC-Asian-Pacific', 'NC-Non-Hisp-White'.

Figure 6.8: U.S. Census Adult dataset FAIR-DB outcome

Number of tuples interested by the rules: 13296 . Total number of tuples: 30169

Cumulative Support: 0.4407172925851039 . Difference Mean: 0.1695548087719456

Race-Difference Mean: 0.08669383850171641 . Sex-Difference Mean: 0.03890844137846591 . Native Country- Difference Mean: 0.05854767971034243

Total number of ACFDs selected: 15

	Rule	Support	Confidence	Diff	RaceDiff	SexDiff	NativeCountryDiff	Mean
16	{'lhs': {'sex': 'Female'}, 'rhs': {'income': '<=50K'}}	0.287447	0.886345	0.135210	0	0.13521	0.000000	0.211329
32	{'lhs': {'sex': 'Female', 'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}	0.262190	0.885382	0.134246	0	0.139644	-0.000963	0.198218
154	{'lhs': {'sex': 'Female', 'race': 'White'}, 'rhs': {'income': '<=50K'}}	0.229540	0.877026	0.125891	-0.00931871	0.140694	0.000000	0.177716
111	{'lhs': {'native-country': 'NC-Asian-Pacific', 'race': 'Other'}, 'rhs': {'income': '>50K'}}	0.000099	0.500000	0.251135	0.184022	0	0.409091	0.125617
31	{'lhs': {'age-range': '45-60', 'income': '>50K'}, 'rhs': {'sex': 'Male'}}	0.084159	0.877636	0.165426	0	0	0.000000	0.124793
152	{'lhs': {'race': 'Amer-Indian-Eskimo', 'native-country': 'NC-Asian-Pacific'}, 'rhs': {'income': '<=50K'}}	0.000066	1.000000	0.248865	0.315978	0	0.118881	0.124466
150	{'lhs': {'race': 'Amer-Indian-Eskimo', 'native-country': 'NC-Non-Hisp-White'}, 'rhs': {'income': '<=50K'}}	0.000033	1.000000	0.248865	0.316199	0	0.118881	0.124449
156	{'lhs': {'sex': 'Female', 'race': 'Black'}, 'rhs': {'income': '<=50K'}}	0.043588	0.939286	0.188150	0.0529407	0.069119	0.000000	0.115869
34	{'lhs': {'sex': 'Female', 'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.015181	0.962185	0.211050	0	0.0540286	0.075840	0.113115
160	{'lhs': {'sex': 'Female', 'race': 'Other'}, 'rhs': {'income': '<=50K'}}	0.002751	0.954023	0.202888	0.0676779	0.0449321	0.000000	0.102819
2	{'lhs': {'native-country': 'NC-Hispanic'}, 'rhs': {'income': '<=50K'}}	0.043919	0.908156	0.157021	0	0	0.157021	0.100470
66	{'lhs': {'race': 'Black'}, 'rhs': {'income': '<=50K'}}	0.081309	0.870167	0.119031	0.119031	0	0.000000	0.100170
122	{'lhs': {'race': 'Black', 'native-country': 'NC-White'}, 'rhs': {'income': '<=50K'}}	0.075839	0.869631	0.118496	0.123893	0	-0.000535	0.097168
28	{'lhs': {'age-range': '45-60', 'income': '<=50K'}, 'rhs': {'sex': 'Female'}}	0.058504	0.394855	0.107064	0	0	0.000000	0.082784
72	{'lhs': {'race': 'Amer-Indian-Eskimo'}, 'rhs': {'income': '<=50K'}}	0.008353	0.881119	0.129984	0.129984	0	0.000000	0.069168

To conclude, analyzing the U.S. Census Adult dataset, the results obtained

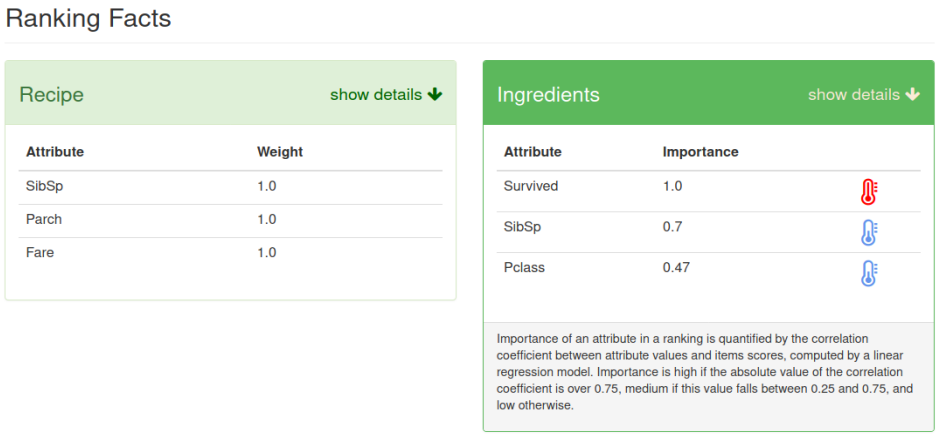
with our framework and *Ranking Facts* notebook are in completely accordance on the fairness aspect.

6.4.2 Titanic dataset

In this paragraph, we apply *Ranking Facts* to another dataset that we studied with our framework: the Titanic dataset.
In this case, the dataset has a smaller number of tuples (876 tuples) so we can use the online demo[5] uploading a similar version of Titanic that we used with our framework. In particular, in this version, we do not discretize the attribute *'Fare'*, so that this attribute could be used to compute the **score-based ranker**.
In fact, to specify the **ranking function** we must choose only numerical attributes, thus we opt for *'Fare'*, *'SibSp'* and *'Parch'*.
For **fairness check** the algorithm needs at least one binary attribute, so we choose only *'Sex'*, because *'Pclass'* is not binary.
For the **diversity check** we choose two categorical attributes: *'Pclass'* and *'Embarked'*.
After having established the input parameters, we apply the algorithm to the dataset. For each widget, we present the obtained results.

In figure 6.9, the first two widgets **Recipe** and **Ingredients** are presented.

Figure 6.9: Recipe and Ingredient widgets

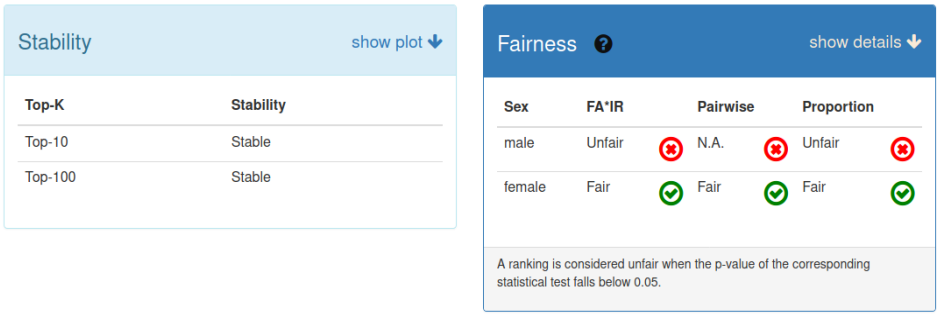


[5] http://demo.dataresponsibly.com/rankingfacts

The first widget specifies the attributes that we used to create the ranking function and their relative weight.
The *Ingredients* widget lists the attributes in order of importance, so clearly the target variable *'Survived'* has the maximum importance, *'SibSp'* attribute follows with a value of 0.7 and then *'Pclass'* attribute takes the third position.

The figure 6.10 presents the following two widgets: **Stability** and **Fairness**.

Figure 6.10: Stability and Fairness widgets



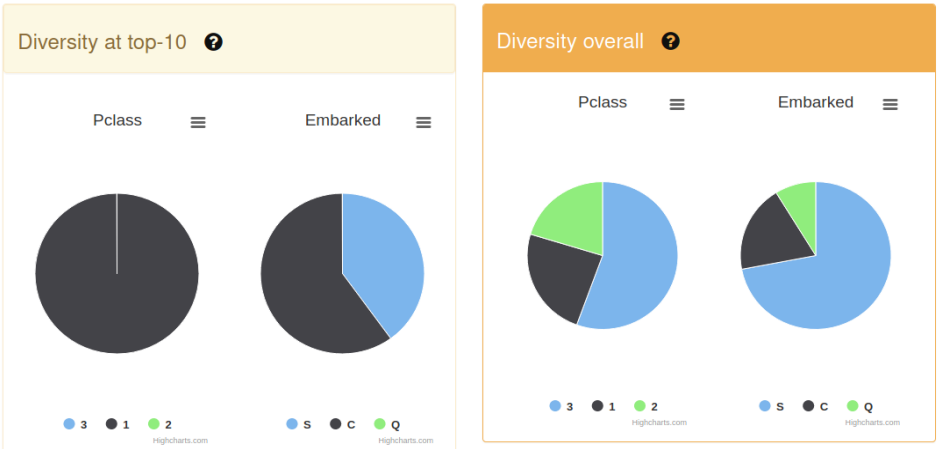
The Stability widget indicates that the ranking created is stable at both two level: *top-10* and *over-all*. Thus, the ranking methodology is robust on the Titanic dataset.
The Fairness widget analyzes the binary attribute *'Sex'* using three different fairness measures. A ranking is considered unfair when the p-value of the corresponding statistical test falls below 0.05. This widget displays that the *'Male'* group is discriminated with respect to *'Female'* group, in fact analyzing the tests more in details, they return $p-value = 0$ for the *'Male'* group.

The result obtained with *Ranking Facts* is very similar to our result, in fact the Titanic dataset more women than men survived.
Because *'Pclass'* attribute is not binary, we can not analyze it with the fairness widget without a Data Preprocessing, but we can study it with the diversity aspect.

Figure 6.11 displays the **Diversity** plots for the attributes *'Pclass'* and *'Embarked'*.

Analyzing the widget at *top-10* level, we can easily detect that the first

Figure 6.11: Diversity widget



10 elements of the ranking consist in people coming only from first class. Moreover, there are people embarked from ports S = Southampton or C = Cherbourg, but not from Q = Queenstown.
Analyzing the *over-all* situation, the plot regarding the class is more distributed over the three classes. For the attribute *'Embarked'*, there are elements coming from Queenstown, but they are still few.

We present the final outcome of our framework, *FAIR-DB*, applied to the Titanic dataset (figure 6.12), and the groups more discriminated that are: men and people in the third class. Instead, the groups that have more privileges are: women and people coming from the first or the second classes. Note that, in our framework, we did not choose ACFDs that involve the *'Embarked'* attribute, so nothing can be said about this attribute.

We can conclude that, also for this dataset, the results obtained with our framework and *Ranking Facts* demo are very similar and comparable.

Figure 6.12: Titanic dataset FAIR-DB outcome

Number of tuples interested by the rules: 647 . Total number of tuples: 876

Cumulative Support: 0.7385844748858448 . Difference Mean: 0.35437190832094806

PClass diff mean: 0.0018055850473224606 . Sex diff mean: 0.00500473096013565 . Port diff mean: 0.0

Total number of ACFDs selected: 6

	Rule	Support	Confidence	Diff	Score	PclassDiff	SexDiff	PortDiff	Mean
93	{'lhs': {'Pclass': '1', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}	0.103881	0.968085	0.578816	0	0.226047	0.323535	0	0.341348
97	{'lhs': {'Pclass': '2', 'Sex': 'female'}, 'rhs': {'Survived': '1'}}	0.079909	0.921053	0.531783	0	0.179014	0.432289	0	0.305846
86	{'lhs': {'Pclass': '3', 'Sex': 'male'}, 'rhs': {'Survived': '0'}}	0.339041	0.865889	0.255159	0	0.058060	0.108189	0	0.297100
77	{'lhs': {'Survived': '0', 'Sex': 'female'}, 'rhs': {'Pclass': '3'}}	0.082192	0.888889	0.332953	0	0.000000	0.199169	0	0.207572
69	{'lhs': {'Survived': '1', 'Pclass': '3'}, 'rhs': {'Sex': 'female'}}	0.082192	0.610169	0.251722	0	-0.073115	0	0	0.166957
81	{'lhs': {'Survived': '1', 'Sex': 'male'}, 'rhs': {'Pclass': '1'}}	0.051370	0.416667	0.175799	0	0.000000	0.0178397	0	0.113584

6.5 Ranking Facts Comparison

In this section we focus on the main aspects of both tools: our system *FAIR-DB* and *Ranking Facts*; in particular we highlight similarities and differences to present a final comparison between them.

Starting from our framework, *FAIR-DB*, the most important strengths are:

- the tool generates a list of rules **semi-automatically**; in the final phase, **user interaction** is required to select the appropriate dependencies according to the research scopes;
- the resulting rules are clear, explicit and are **self-explanatory**, in particular they give many insights about bias in dataset;
- given the final ranking the user can solve the bias in the dataset applying techniques based on **dependency-repair methods**.

Instead, the main properties of *Ranking Facts* are [37]:

- the tool generates labels/widgets **automatically** or **semi-automatically**;
- the information that it gives is **comprehensible** because it is short, simple and clear;
- the resulting labels are not only metadata, but they could be **consultative** providing actionable information.

Analyzing the main properties in common, both tools have comparable features and some of them are also related. Although, the main differences between *FAIR-DB* and *Ranking Facts* are:

1. *Ranking Facts* requires numerical attributes to build the score-based ranker and binary categorical variables to compute the fairness measures. *FAIR-DB* has no constraints on the **attributes types**, we can both check fairness or numerical or categorical attributes.

2. *FAIR-DB* requires **Data Preprocessing**, in particular it is important to solve missing values and apply discretization. *Ranking Facts* requires that the input dataset is complete, so without missing values and, in some cases, data preprocessing for the attribute types.

3. *Ranking Facts* is based on the paradigm of **interpretability-by-design** using widgets and plots that are are very intuitive and clear; *FAIR-DB* visualizes a ranking in tabular form with all the dependencies and the related metrics. Our ranking reports the ACFDs in meaningful way, because the user has the possibility to choose the ordering option.

4. In *Ranking Facts* only the *Ingredient* widget can partially catch the relationship between attributes. In *FAIR-DB*, the ACFDs report the **correlation** between the attributes and visualize it in a concise way.

5. Our tool discovers ACFDs that could involve more than one group at a time and can report information about subgroups. For example, one rule can consider both women and black minority, obtaining more insights about bias in the dataset. The *Fairness* widget in *Ranking Facts* can check fairness only between one attribute domain each time, so the results do not contain information about the existing **bias in subgroups** or in the minorities.

6. *FAIR-DB* promotes **user interaction** allowing customization and stimulating the user by the choice of parameters and the final selection of the dependency. *Ranking Facts* uses automated pipelines, so it requires a user communication only for establishing the input parameters.

7. Our tool is not based on **statistical definitions** of fairness or **statistical test** that involve a classifier previously computed on the dataset, but only on the distribution analysis. *Ranking Facts* is based on group fairness checks that need the computation of statistical test.

8. *FAIR-DB* is **scalable**, it can analyze dataset with more that 30'000 tuples; the *Ranking Fact* demo can handle fewer tuples than our

framework (because of the time-out error), although the notebook version has not this limit.

To conclude, both approaches, *Ranking Facts* and *FAIR-DB*, analyze fairness and discover bias in datasets. They are both in accordance with the results showing similarities and common aspects as the comprehensibility and the consultative form of data, but also differences as the user interaction or the customization aspect that show the uniqueness of each approach.

Chapter 7

Conclusions and Future Works

7.1 Conclusive Summary

We developed a novel framework, ***FAIR-DB***, that, through the extraction of a particular type of Functional Dependencies, it discovers already existing bias and discrimination in datasets.
We used this framework in order to retrieve the dependencies among the attributes of two well known datasets (*U.S. Census Adult* dataset and *Titanic* dataset) after some preprocessing operations on the datasets.
We applied filtering and selection operations to the discovered dependencies, aiming to extract only the unethical ones. The tool pipeline continued by performing a completion process and it encouraged user communication: the user chooses the dependencies that are more interesting for her type of research and we show a final ranking with summarizing scoring measures. These last metrics allow to understand how much the dataset is fair and favours the detection of groups in the dataset that are discriminated. Lastly, regarding the groups that suffer from bias, we reported some tuple examples. In this research, we also focused on small groups, named minorities, and we proposed an additional workflow to analyze the already existing bias when the dataset contains minorities.

Moreover, we compared our results with an already existing tool *Ranking Facts* [37] in order to evaluate the main differences and similarities between the two tools.
FAIR-DB proved to be able to identify the groups that verifies fairness/unfairness checks, and also analyze the subgroup fairness thanks to the metrics.

Our tool also suggests possible future steps, by showing the tuples of the dataset more involved by the dependencies. In fact, the user could correct the dataset by using some already existing algorithm as, for example, automatic tools of CFD-repair [26].

7.2 Future Works

As we highlighted during the previous sections of this thesis, the discovery of data bias thought Functional Dependencies is organized into several phases and involves many complications and limits.
First of all, we focused on *missing values* as our main source of data errors, but in real world datasets there could be different types of data errors.
Another relevant limitation of our work is the focus on *relational databases* only. However, there are other tools that can perform the transformation from non relational to relational database, hence we believe that our framework may be integrated with them in order to function properly.
Future works based on this thesis may deepen these aspects.

An interesting topic that may be addressed in the future is the *evaluation of different metrics* for the fairness check of the dependencies, like the statistical tests or similarity measures.
Furthermore, it would be beneficial to perform a *CFD-repair algorithm* [26] after the application of our framework to further evaluations.

On a final note, we believe that it would be beneficial to *integrate FAIR-DB* with other tools of Data Science Ethics, like *Ranking Facts* [37], whose focus is on different notions and other aspects of fairness. This would lead to an increase of the overall efficiency and completeness of the systems and it would allow us to tackle a broader variety of data bias and discrimination problems.

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Andrew Altman. Discrimination. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2020 edition, 2020.

[3] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning*. fairmlbook.org, 2019.

[4] Rachel K. E. Bellamy, Kuntal Dey, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilovic, Seema Nagar, Karthikeyan Natesan Ramamurthy, John T. Richards, Diptikalyan Saha, Prasanna Sattigeri, Moninder Singh, Kush R. Varshney, and Yunfeng Zhang. AI fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM J. Res. Dev.*, 63(4/5):4:1–4:15, 2019.

[5] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. Relaxed functional dependencies - A survey of approaches. *IEEE Trans. Knowl. Data Eng.*, 28(1):147–165, 2016.

[6] Alexandra Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big Data*, 5(2):153–163, 2017.

[7] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[8] National Research Council. *Measuring Racial Discrimination*. The National Academies Press, Washington, DC, 2004.

[9] Brian d'Alessandro, Cathy O'Neil, and Tom LaGatta. Conscientious classification: A data scientist's guide to discrimination-aware classification. *CoRR*, abs/1907.09013, 2019.

[10] Amit Datta, Michael Carl Tschantz, and Anupam Datta. Automated experiments on ad privacy settings: A tale of opacity, choice, and discrimination. *CoRR*, abs/1408.6491, 2014.

[11] Flávio du Pin Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R. Varshney. Optimized pre-processing for discrimination prevention. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3992–4001, 2017.

[12] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[13] Wenfei Fan, Floris Geerts, and Jef Wijsen. Determining the currency of data. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 71–82. ACM, 2011.

[14] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams, editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 259–268. ACM, 2015.

[15] Luciano Floridi and Mariarosaria Taddeo. What is data ethics? *Philosophical Transactions of The Royal Society A Mathematical Physical and Engineering Sciences*, 374:1–9, 12 2016.

[16] Chris Giannella and Edward L. Robertson. On approximation measures for functional dependencies. *Inf. Syst.*, 29(6):483–507, 2004.

[17] Sara Hajian, Francesco Bonchi, and Carlos Castillo. Algorithmic bias: From discrimination discovery to fairness-aware data mining. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 2125–2126. ACM, 2016.

[18] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[19] Lauren Kirchner Jeff Larson, Surya Mattu and Julia Angwin. How we analyzed the compas recidivism algorithm, 2016.

[20] Jolanda Jetten and Kim Peters. *The Social Psychology of Inequality*. Springer International Publishing, 01 2019.

[21] Ariana Tobin Julia Angwin and Madeleine Varner. Machine Bias: facebook (still) letting housing advertisers exclude users by race, 2017.

[22] Surya Mattu Julia Angwin, Jeff Larson and Lauren Kirchner. Machine Bias: there's software used across the country to predict future criminals. and it's biased against blacks., 2016.

[23] Faisal Kamiran and Toon Calders. Data preprocessing techniques for classification without discrimination. *Knowl. Inf. Syst.*, 33(1):1–33, 2011.

[24] David Madras, Elliot Creager, Toniann Pitassi, and Richard S. Zemel. Learning adversarially fair and transferable representations. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3381–3390. PMLR, 2018.

[25] Sajid Mahmood, Muhammad Shahbaz, and Aziz Guergachi. Negative and positive association rules mining from text using frequent and infrequent itemsets. *The Scientific World Journal*, 2014, 05 2014.

[26] Mirjana Mazuran, Elisa Quintarelli, Letizia Tanca, and Stefania Ugolini. Semi-automatic support for evolving functional dependencies. In Evaggelia Pitoura, Sofian Maabout, Georgia Koutrika, Amélie Marian, Letizia Tanca, Ioana Manolescu, and Kostas Stefanidis, editors, *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016*, pages 293–304. OpenProceedings.org, 2016.

[27] Raoul Medina and Lhouari Nourine. A unified hierarchy for functional dependencies, conditional functional dependencies and association rules. In Sébastien Ferré and Sebastian Rudolph, editors, *Formal Concept*

Analysis, 7th International Conference, ICFCA 2009, Darmstadt, Germany, May 21-24, 2009, Proceedings, volume 5548 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2009.

[28] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *CoRR*, abs/1908.09635, 2019.

[29] James Moor. What is computer ethics? *Metaphilosophy*, 16:266 – 275, 08 1985.

[30] Heiko Müller and Johann-Christoph Freytag. Problems, methods, and challenges in comprehensive data cleansing. pages 6–7, 01 2003.

[31] Krystyna Napierala and Jerzy Stefanowski. Types of minority class examples and their influence on learning classifiers from imbalanced data. *J. Intell. Inf. Syst.*, 46(3):563–597, 2016.

[32] The Editors of Encyclopaedia Britannica. Database. In Edward N. Zalta, editor, *Encyclopædia Britannica*. May 18, 2020 edition, 2020.

[33] Alexandra Olteanu, Carlos Castillo, Fernando Diaz, and Emre Kiciman. Social data: Biases, methodological pitfalls, and ethical boundaries. *Frontiers Big Data*, 2:13, 2019.

[34] Joeri Rammelaere and Floris Geerts. Revisiting conditional functional dependency discovery: Splitting the "c" from the "fd". In Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2018, Dublin, Ireland, September 10-14, 2018, Proceedings, Part II*, volume 11052 of *Lecture Notes in Computer Science*, pages 552–568. Springer, 2018.

[35] John Rawls. Justice as fairness: Political not metaphysical. *Philosophy and Public Affairs*, 14(3):223–251, 1985.

[36] Nripsuta Ani Saxena, Karen Huang, Evan DeFilippis, Goran Radanovic, David C. Parkes, and Yang Liu. How do fairness definitions fare? testing public attitudes towards three algorithmic definitions of fairness in loan allocations. *Artif. Intell.*, 283:103238, 2020.

[37] Julia Stoyanovich and Bill Howe. Nutritional labels for data and models. *IEEE Data Eng. Bull.*, 42(3):13–23, 2019.

[38] Harini Suresh and John V. Guttag. A framework for understanding unintended consequences of machine learning. *CoRR*, abs/1901.10002, 2019.

[39] Pang-Ning Tan, Michael S. Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[40] Sahil Verma and Julia Rubin. Fairness definitions explained. In Yuriy Brun, Brittany Johnson, and Alexandra Meliou, editors, *Proceedings of the International Workshop on Software Fairness, FairWare@ICSE 2018, Gothenburg, Sweden, May 29, 2018*, pages 1–7. ACM, 2018.

[41] Catharine M. Wyss, Chris Giannella, and Edward L. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract. In Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors, *Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001, Munich, Germany, September 5-7, 2001, Proceedings*, volume 2114 of *Lecture Notes in Computer Science*, pages 101–110. Springer, 2001.

[42] Lu Zhang, Yongkai Wu, and Xintao Wu. A causal framework for discovering and removing direct and indirect discrimination. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 3929–3935. ijcai.org, 2017.

Appendix A

User Manual

A.1 Pseudo-code of FAIR-DB

In this section we present the pseudocode of the algorithm on which is based our framework, *FAIR-DB*.

Algorithm 1 FAIR-DB Algorithm

1: $df \leftarrow dataset$ ▷ import dataset
2: $dependencies \leftarrow$ **CFDDiscovery**$(df, minSupp, minConf, maxSize)$
3: $ACFDs \leftarrow$ **removeAFDs**$(dependencies, df)$
4: $ACFDs \leftarrow$ **filterACFDs**$(ACFDs, df)$
5: $ACFDsTable \leftarrow$ **computeMetrics**$(ACFDs, df)$
6: $selectedACFDs \leftarrow$ **filterByDifference**$(ACFDsTable, minThreshold_1)$
7: $combinedACFDs \leftarrow$ **computeCombinations**$(selectedACFDs, df)$
8: $completionACFDsTable \leftarrow$ **computeMetrics**$(combinedACFDss, df)$
9: $selectedACFDs \leftarrow$ **filterByDifference**$(completionACFDsTable, minThreshold_2)$
10: $rankedACFDs \leftarrow$ **rankACFDs**$(selectedACFDs, option)$
11: $finalACFDs \leftarrow$ **selectByIndexes**$(rankedACFDs, ruleIndexes)$
12: $cumulativeSupp, differenceMean \leftarrow$ **computeMean**$(finalACFDs)$
13: **for** p in **protectedAttributes**(df) **do**
14: $\quad pDifferenceMean \leftarrow$ **computeMean**$(finalACFDs, p)$
15: [Optional:] $problematicTuples \leftarrow$ **extractProblematicTuples**$(df, finalACFDs, M)$

Here, there is the explanation for all the functions of the framework:

- **CFDDiscovery** is a given algorithm that mines the dependencies from the dataset given a *minSupport, minConfidence* and *maxSize*.
- **removeAFDs** is a function that given the dataset and the dependencies, removes the rules that have at least one attribute without the

corresponding value.

- **filterACFDs** is a function that, given the dataset as input, filters the ACFDs returning only the ones that contain at least one protected attribute and the target class.
- **computeMetrics** is a function that, given the ACFDs and the dataset, for each dependency computes the metrics: *support, confidence, difference* and, for each protected attribute p the *pDifference*. After that, it returns the ACFDs and the metrics as a table.
- **filterByDifference** is a function that simply filters the rules that have $difference > minThreshold$.
- **computeCombination** is a function that given the ACFDs and the dataset, it computes all the possible combinations over the protected attributes and the target class.
- **rankACFDs** is a function that given the selected ACFDs and an option (*Support, Difference* or *Mean*), it builds the ranking ordering using the chosen option.
- **selectByIndexes** is a function that given the ranked ACFDs and the indexes of the rules chosen by the user, it returns the final ranking composed by the selected ACFDs.
- **protectedAttributes**, given the dataset as input, it returns all the protected attributes.
- **computeMean**, it is function that given the final ACFDs, in one case, it computes:
  - the *cumulativeSupport* that is the percentage of tuples involved by the selected dependencies;
  - the *differenceMean* that is the mean of all the difference metrics;

  in the second case, it takes as input also the protected attribute p and it computes the mean of the *pDifference* of all the rules returning the *pDifferenceMean*.
- **extractProblematicTuples** is a function that given the dataset, the final ACFDs and a number M, it returns all the tuples in the dataset that are verified for more than M rules.

Algorithm computeMetrics for each ACFD compute metrics and return the table

```
1: procedure computeMetrics(ACFDs, df)
2:     for ACFD in ACFDs do
3:         ACFDsTable                                   ▷ create the table
4:         support, confidence ← computeSupport(ACFD, df)
5:         noProtAttrConf ← computeNoProtectedAttrConf(ACFD, df)
6:         difference ← confidence − NoProtAttrConf
7:         append ACFD, support, confidence, difference to ACFDsTable
8:         for p in protectedAttributes(df) do
9:             noPConf ← computeNoPConf(ACFD, p, df)
10:            pDifference = confidence − noPConf
11:            append pDifference to ACFDsTable
12:    return ACFDsTable
```

Algorithm computeSupport for each ACFD compute *support* and *confidence*

```
1: procedure computeSupport(ACFD, df)
2:     LHS ← getLHS(ACFD)
3:     RHS ← getRHS(ACFD)
4:     X, Y, XY ← 0
5:     for tuple in df do
6:         LHSFlag, RHSFlag ← False
7:         if (LHS in tuple) then
8:             X ← X + 1
9:             LHSFlag ← True
10:        if (RHS in tuple) then
11:            Y ← Y + 1
12:            RHSFlag ← True
13:        if (RHSFlag and LHSFlag) then
14:            XY ← XY + 1
15:    allTuples ← len(df)
16:    support ← XY / allTuples
17:    confidence ← XY / X
18:    return support, confidence
```

Algorithm computeNoProtectedAttrConf for each ACFD compute the *noProtectedAttributeConfidence*

```
1: procedure computeNpProtectedAttrConf(ACFD, df)
2:     noProtAttrLHS ← getNoProtectedAttrLHS(ACFD)
3:     RHS ← getRHS(ACFD)
4:     noProtAttrInX, noProtAttrInXY ← 0
5:     for (tuple in df) do
6:         noProtAttrLHSFlag, RHSFlag ← False
7:         if (noProtAttrLHS in tuple) then
8:             noProtAttrInX ← noProtAttrInX + 1
9:             noProtAttrLHSFlag ← True
10:        if (RHS in tuple) then
11:            RHSFlag ← True
12:        if (RHSFlag and noProtAttrLHSFlag) then
13:            noProtAttrInXY ← noProtAttrInXY + 1
14:    allTuples ← len(df)
15:    noProtectedAttrConf ← noProtAttrinXY / noProtAttrinX
16:    return noProtectedAttrConf
```