University of Ljubljana
Faculty of Computer and Information Science

Antolović Vanda

# Evaluation and Comparison of Data Mining and Machine Learning Capabilities Within Relational Database Management Systems

MASTER'S THESIS

THE 2ND CYCLE MASTER'S STUDY PROGRAMME
COMPUTER AND INFORMATION SCIENCE

Supervisor: assoc. prof. dr. Matjaž Kukar

Ljubljana, 2021

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Antolović Vanda

# Ovrednotenje in primerjava orodij za podatkovno rudarjenje in strojnega učenja znotraj sistemov za upravljanje relacijskih podatkovnih baz

magistrsko delo

magistrski študijski program druge stopnje
računalništvo in informatika

Mentor: assoc. prof. dr. Matjaž Kukar

Ljubljana, 2021

[1]V dogovorju z mentorjem lahko kandidat magistrsko delo s pripadajočo izvorno kodo izda tudi pod drugo licenco, ki ponuja določen del pravic vsem: npr. Creative Commons, GNU GPL. V tem primeru na to mesto vstavite opis licence, na primer tekst [1].

# Acknowledgments

*Antolović Vanda, 2021*

# Contents

# List of used acronmys

| **acronym** | **meaning** |
|---|---|
| **ML** | machine learning |
| **RDBMS** | relational database management system |
| **IT** | information technology |
| **DSS** | decision support system |
| **SQL** | structured query language |
| **OLTP** | online transaction processing system) |
| **GUI** | graphical user interface) |
| **LR** | logistic regression) |
| **GNB** | gaussian naive bayes) |
| **SVC** | linear support vector classification) |
| **KNN** | k-nearest neighbors classifier) |
| **DTC** | decision tree classifier) |
| **RTC** | random forest classifier) |
| **GBC** | gradient boosting classifier) |
| **LiR** | linear regression) |
| **BR** | bayesian ridge regressor) |
| **SVR** | support vector regressor) |
| **KNR** | k-nearest neighbors regressor) |
| **DTR** | decision tree regressor) |
| **EN** | elastic net) |
| **GBR** | gradient boosting regressor) |
| **RPM** | red hat package manager) |
| **GLM** | general linear model) |
| **SVM** | support vector machine |
| **DT** | decision tree) |
| **NB** | naive bayes) |
| **MSE** | mean squared error) |
| **RMSE** | root mean squared error) |
| **MAE** | mean absolute error) |

# Abstract

**Title:** Evaluation and Comparison of Data Mining and Machine Learning Capabilities Within Relational Database Management Systems

The evolution of data and the data science technology started bringing machine learning algorithms to the data, to ease the process of training and lessen the possibility of data corruption by transfers from system to system. We had here picked four combinations of Relational database management systems and integrated or semi integrated machine learning toolsets - Python and SQLite, MariaDB and MindsDB, PostgreSQL and MindsDB, and Oracle with Oracle Machine Learning. All four combinations were compared with the help of accuracies and training time they have achieved over seven datasets. Six of the dataset had classification problem setting, two were with multiple classification other with binary, and one dataset with regression problem setting. MariaDB had had the slowest training time, while MindsDB could not evaluate the dataset containing larger strings or produce qualitative measures for assessing dataset with regression target value. Oracle produced best results, as it was able to accurately evaluate all datasets with fast training time. Even though same is accurate for Python, data had to be optimized and transformed into numerical for the main Python's machine learning library to be able to process the data. Considering this all, a simple decision support system was created to help make a sensible decision which toolset to use to suit user's needs.

## Keywords

*machine learning, data mining, database, dataset, running time*

# Povzetek

**Naslov:** Ovrednotenje in primerjava orodij za podatkovno rudarjenje in strojnega učenja znotraj sistemov za upravljanje relacijskih podatkovnih baz

Razvoj podatkov in podatkovne znanosti sta začela prinašati algoritme strojnega učenja neposredno k podatkom, kar je olajšalo proces usposabljanja in zmanjšalo možnost poškodovanja podatkov s prenosom iz sistema v sistem. V tem magistrskem delu smo izbrali štiri kombinacije sitemov upravljanja relacijskih baz podatkov in integriranih ali pol integriranih naborov orodij za strojno učenje - Python in SQLite, MariaDB in MindsDB, PostgreSQl in MindsDB ter Oracle s Oracle Machine Learning. Vse štiri kombinacije so bile primerjane s pomočjo metrike natančnosti in časa usposabljanja, ki so ta sistemi ga dosegli nad sedmih naborih podatkov. Šest naborih podatkov je imelo klasifikacijsko nastavitev težave, dva sta imela več klasifikacij ostali pa binarno, in en nabor podatkov je imel nastavitvo regresijske težave. MariaDB je imel napočasnejši čas usposabljanja, medtem ko MindsDB ni mogel oceniti nabor podatkov, ki vsebuje večje nize, niti izdelati kakovostnih meril za oceno regresijeskega nabora podatkov. Oracle je dosegel najboljše rezulate, saj je natančno ocenil vse nabore podatkov s hitrim časom usposabljanja. Čeprav je Python enako, ampak je bilo treba podatke optimizirati in spremeniti v numerične, da lahko Pythoneva glavna knjžica za strojno učenje obdela podatke. Glede na vse je bil ustvarjen preprost sistem za podporo odločanju, ki je pomagal sprejeti razumno odločitev, kateri nabor orodij uporabiti za potrebe uporabnikov.

## Ključne besede

*strojno učenje, podatkovno rudarjenje, podatkovna baza, nabor podatkov, čas delovanja*

# Razširjeni povzetek

# Chapter 1

# Introduction

The main goal of this thesis was to evaluate and compare different database systems' capabilities; precisely the data mining and machine learning (ML) in-database subsystems. A global interest in making a crossover between these two is fairly new, there are continuing evaluations preformed by processing data with ML toolsets from within the database [2]. As the world is constantly changing, people have become more concerned about the privacy of their data, its rapid growth and its availability over various platforms. Therefore, standard extractions and data movement between different systems/programes are becoming more scrutinized for data safeness. Data growth has forced many organizations to down-sample, which affects the machine model's accuracy. The number of systems that process extensive amounts of data is getting smaller and the built of predictive models has become more complex [3]. All this implies that making ML model training inference an integral part of relational query processing could help [4]. This can extensively be useful to organizations and people in these organizations, but also to individuals, who do not have excessive IT knowledge in transforming and optimizing their data and for developing their own ML algorithms' predictions.

Relational database management systems (RDBMS) with accompanying integrated or semi integrated ML toolsets that were evaluated and compared are the following ones: SQLite with Python (a built in feature of Python with databases placed directly on our disks, with the help of libraries like pandas and scikit-learn [5] machine learning tasks are done partially in-database), MariaDB with MindsDB (made with goal to bring ML closer to the non-experts to predict and explain the data [6]), PostgreSQL with MindsDB, and Oracle with Oracle Machine Learning (the idea behind ML in Oracle was to move the algorithm not the data, meaning the analytics are preformed directly on the data inside the database, along with regulating results and availability of database's learning models [7]). Main emphasis was put on analysing predictive performance (accuracy as a typical classification machine learning metric and area under the curve, also known as ROC, and mean squared error, root mean squared error and mean absolute error as typical regression machine learning metrics) and the speed of training. Training time has been defined as evaluation metric, as the complexity of data challenges systems and lengthens training in general. Comparisons between running times are most often used in these kind of analysis as seen in [8]. The evaluated combinations were set up in a clean new virtual Linux environment for most substantive analysis. The capabilities of each toolset was tested through relational databases filled with different datasets. Ultimately seven datasets of different sizes and imbalanced sets were used, six with a classification problem setting (binary classification and multiclass classification) and one with a regression target value. The idea behind using such is to help determine how diversely organized datasets containing different data types also affect the main success indicators. Ultimate comparison between tools was over same dataset.

With the received analysis results a basic Decision Support System (DSS) was set up. Decision support systems are knowledge based and designed to help in crucial decision making in various organizational activities (planning, manufacturing, management decisions, etc.) [9]. The idea is that the DSS helps smaller organizations with new projects related to in-database data mining and ML, precisely in selection of the most suitable toolset depending on their needs and appointed evaluation markers. Various utility criteria was defined to use as input, as user's operating system and hardware dependence, data they needed to evaluate and prior knowledge were also considered.

The organization of next chapters is as follows, second chapter (2) has a short summary of related work, then in third chapter (3) we introduce the combination of toolsets and RDBMS we have used along with descriptions of each dataset, in the fourth (4) we disclose the environment set up process, data optimization that was done in certain cases, and data importation into the RDBMSs. In the fifth chapter (5) main results are visualized and compared, second to last (6) we explained the set up of DSS and lastly in seventh chapter (7) we listed our thoughts on the overall process and collected results.

# Chapter 2

# Related Work

Some of the recognizable scientific works this thesis could be compared to are those focused on analysing their own solutions opposed to others used in the same domain. One of these solutions is Vertica [10] where ML toolset is implemented inside a specifically created Vertica analytic database [8]. It offers numerous ML algorithms and tools to be used in-database. They have also included a special model object for storing training results, which allowed them to develop new tools for model management. These models are treated as first-class database objects in Vertica. By providing R and Python libraries Vertica's users are given the opportunity to do a complete data analysis in a language of their choice. Vertica's success was tested over two different datasets and compared to Spark-MLlib's results over the same data. Vertica gave highly comparable results to Spark-MLlib, which is a more often used library in the same domain. In translation Spark's overall training time was longer than Vertica's total running time.

Integration of gradient descent and tensor algebra in existing database systems as a new relational operator eliminates the need for data transfer [11]. The relational operator for specifying model functions use SQL lambda functions (anonymous SQL functions defined in the query; these are utilized from [12] where their implementation serves so data analytic operator can be used for multiple algorithms by changing the varying point with one specified by the user, exactly this injection is done with lambda functions). All mentioned allows data to be transformed inside of the database and the models to be trained using gradient descent. When running times were compared this solution gave better results in some aspects than widely recognized library TensorFlow.

MindsDB has provided a short comparison between its own system and a few others such as, Sklearn, TensorFlow and Ludwig [13] by building simple training models. They have explained that the pro of MindsDB over the others is its simplicity even though the accuracy does not always beat the other systems.

Another comparable system is MADlib analytics, an open-source library supporting numerous ML methods. MADlib creators have put library's focus on parallel databases - on shared-nothing parallelism that allows highly refined analytics, as explained [14]. The essence of its main functionalities were tested on PostrgeSQL and GreenPlum database. Both performed similarly in achieving sufficient parallel speedup and in minimization of performance overheads. Comparisons had not been done in [14] between MADlib and Apache Mahout (data mining framework running on Hadoop infrastructure which distributes huge amounts of data on several clusters, Mahout is essentially used to scale ML algorithms [15]). However, probable differences would be various algorithm implementations' difficulty or system's differences between relational database management systems and Hadoop [14]. But, Hadoop MapReduce has been tested against two parallel database management systems: Vertica and DBMS-X [16]. In essence database management systems were better in performance in tasks at all scaling levels, but proved to be lacking in adding user defined functions.

Another solution is Raven [17] that outperforms, with in-database ML inference, performance of dedicated frameworks as standalone ONNX Runtime (open source project focused on advancing performance and accelerating ML inference [18]). Raven allows users to use ML frameworks of their choice for building and training ML models. The frameworks along with preprocessing steps and different library dependencies form a model pipeline. When invoking the model pipeline with an SQL query it forms inference queries. To conclude, Raven's performance is higher by up to 24 times than standalone ONNX Runtime.

As [8] points out, it is extremely difficult to compare two systems accurately to get conclusive results on which system preforms better because the underlying optimization logics are implemented differently. However, by doing all of the processes in-database the time needed for data transfers between the database and another environment has been reduced. Therefore in-database functions have proven to be beneficial.

This thesis will present innovative detailed comparisons of in-database ML and data mining. There are different approaches for improving database management systems such as, implementation of ML toolsets and integrations of new functionalities. Those have been compared to popular widely used tools, e.g. above mentioned integration of gradient descent optimizer into database systems [11], implementations of multi-dimensional array-like indices in database tables and modifications of query optimizer on top of SimSQL (analytic relational database designed to handle stochastic analytics, specifically with simulated data [19]) [20], but not yet to other similar solutions as done here.

# Chapter 3

# Introducing ML and RDBM combinations and Datasets

To present a thorough analysis, different RDMS that offer in-database or partially in-database ML functions had to be compared. In the following section 3.1 each of the used tools is presented. Also, as was mentioned different datasets in the view of problem setting (classification and regression), quantity of data and balance of data were used. Each datasets' form is described in 3.2.

## 3.1 Systems with accompanying integrated machine learning toolsets

### 3.1.1 SQLite and Python

SQLite is a library that can be imported into Python using a Python module called **sqlite3**. This therefore makes SQLite a partially integrated in database ML solution. It is a C library that stores all data onto the user's disk, it does not require connections to a separate running server, but with the help of a nonstandard variant of SQL query language allows querying of the data [21]. SQLite is used among programmers for smaller mobile and tablet applications, the embedded system applications with mostly OLTP (Online Transaction Processing System) workloads, for daily short and fast insert, update and delete transactions [22]. It is used in Python in combination with a few other libraries and tools that ease the work with the data, such as Pandas library [23] and Scikit-learn tool for ML [24]. The usage in the scope of this thesis is explained in following chapter 4.1.

### 3.1.2 MindsDB

MindsDB is an open-source tool that allows integration of ML into different relational database systems. It automates the process of training, evaluating, selecting and optimizing models. It is intended to help users who do not have data science knowledge with building prediction models, as well to speed up the data optimization process and offer cost savings to others [25]. It has a GUI for deploying models and understanding predictions, but it can also be used with plain SQL. Currently offered integrations are ClickHouse, MariaDB, PostgreSQL, MySQL, MongoDB and Microsoft SQL Server, with this list they cover 55% of open source databases [26], but they have also started to spread their integrations to systems as Oracle, SingleStore, Apache casssandra, Kafka, snowFlake and DataStax Astra. A specialty offered by MindsDB are AI tables that are created with queries inside the database and contain predictions created in real time by the already deployed models. These offer insight into the data and the existing relations between the attributes [27]. Installation of MindsDB is possible on different operating systems.

The two integrations that were covered and compared in scope of this thesis were integration with MariaDB and with PostgreSQL.

#### MariaDB

MariaDB is a relational open-source database, a successor of MySQL. The developers are trying for it to maintain high compatibility with MySQL, ensuring a complete match with its APIs and commands, as well as ensuring usability of the libraries [28]. It has changed the standard MySQL storage engines, such as InnoDB and MyISAM, with XtraDB and Aria, which offer better caching and they speed up complex queries with the help of temporary tables that are also using mentioned engines [29]. They have bettered the solution with more storage engines and a larger connection pool [30]. MariaDB's connection to MindsDB is possible in two ways, with MariaDB client or with MindsDB Studio GUI.

#### PostgreSQL

PostgreSQL is another widely used open-source relational database. Its popularity is owed to its reliability, feature robustness and performance [31]. A few of its many offered features are definition of user's own data types, custom functions, code writing from different programming languages without recompiling the database, etc [32]. PostgreSQL brings different technology innovation to the user through one platform, such as Data Warehouse Analytics, High Performing OLTP, Cloud Services, ML and etc [33]. Its support of ACID (Atomicity, Consistency, Isolation, Durability) allowed it to be used even in the financial industry. PostgreSQL can be connected to MindsDB in the same way as MariaDB - with help of MindsDB Studio GUI or with psql client.

### 3.1.3 Oracle and Oracle Machine Learning

Oracle is a relational database management system, it supports any kind of data models and offers different editions: the Standard Edition, Enterprise Edition, Express Edition and Personal Edition. In this thesis Express edition was used, as it is free, more convenient to install and it offers mainly the same quality of content in a much more compressed solution. Oracle is mainly developed for enterprises, for their grid computing and data warehousing [34]. Its features are Scalability and performance, Availability, Backup and Recovery and Security. One of Oracle's integrated parts, important in this work, is Oracle Machine Learning, which includes a family of products used to analyze and prepare data, as well as to build, deploy and evaluate ML models [35]. The in-database algorithms which we emphasized here, manage distribution across multiple nodes and multiple threads. Tool that allows easier access to data and its manipulation is SQLDeveloper [36], and its Oracle Data Miner extension [37], enables the user to build and deploy models with the help of a graphical editor.

## 3.2 Datasets

Emphasis was put on two types of datasets, dataset with a classification problem setting and dataset with a regression problem setting. Both classification and regression are part of supervised learning which uses a model for learning to map between input values and the output target value [38]. Classification is used to predict a predefined class value. Depending if there are two or more class values, we have binary or multiclass classification. Regression is used to predict a discrete numerical value in integer form based on previous data in dataset. Besides supervised learning there is also unsupervised learning (uses models to determine or explain relationships in data by identifying hidden patterns), semi-supervised learning (combines techniques from unsupervised and supervised learning) and reinforcement learning (learning by reward/feedback by interacting with the environment) [39]. In the next subsections each dataset that was analysed in this master thesis is described.

### 3.2.1 Stroke dataset

Stroke dataset was used from Kaggle [40]. Kaggle is a web data science environment that allows publishing of various datasets, discussions about model building and organizes competitions on ML tasks. Dataset Stroke was created to help predict whether a patient will suffer a stroke based on different predictor values determined to help in accurate predictions. It contains 5110 rows and 12 columns. Its target variable is the last attribute **Stroke** which can only have one of the two values, 0 which represents a patient who had not suffered a stroke, and 1 that represents a patient who did. It is an imbalanced set of data, as seen in 3.1, with more records of patients who had not had a stroke, 4861, compared to 249 who had had a stroke.

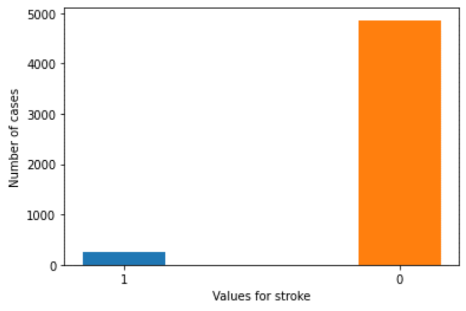


**Figure 3.1:** Balance of records in Stroke dataset

### 3.2.2 Hotel satisfaction dataset

Hotel satisfaction dataset was also found and used from Kaggle [41]. This dataset is used to predict overall satisfaction of guests staying at a hotel depending on a few criteria, like purpose of travel, ratings of hotel location, its offer of food and drinks, etc. It contains 103904 records and 17 attributes. The target variable is **Satisfaction** which is again a binary classification, with values - Satisfied, 45025 entries, and Neutral or dissatisfied, 58879 entries 3.2.

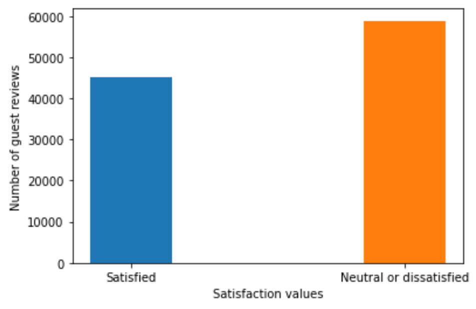


**Figure 3.2:** Balance of records in Hotel satisfaction dataset

### 3.2.3 Hotel booking datasets

Hotel Booking consists of two datasets. The data was obtained from real hotels property management system databases' servers [42]. One dataset is for a hotel located in the city and the other for a hotel located in a resort. Both datasets have the same structure, 31 attributes describing a booking entry, with their target variable being related to guests' cancellation of their reservation, **IsCanceled**. It is a binary classification with values being Canceled and Not canceled. The number of records between the two is different, City hotel contains 79330 records (with 33102 canceled reservations, 46228 not canceled reservations), while Resort hotel has 40060 rows (11122 canceled reservations, and 28938 not canceled reservation entries), seen in 3.3.

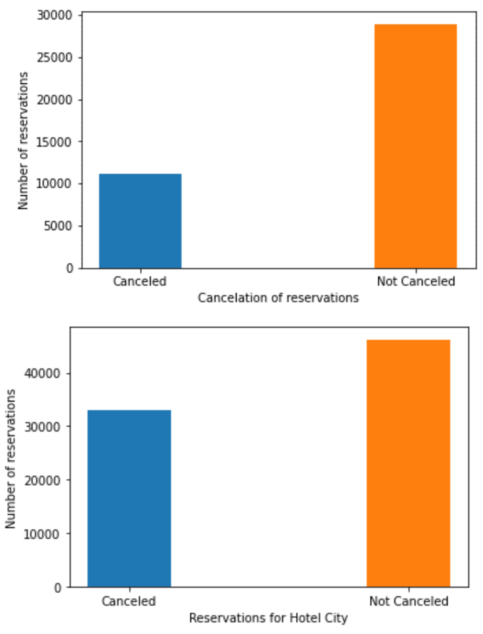


**Figure 3.3:** Balance of records in Hotel booking Resort and City datasets

### 3.2.4 In-vehicle coupon recommendation dataset

In-vehicle coupon recommendation dataset was collected via survey for Amazon Mechanical Turk [43]. It is a dataset with 12684 rows, and 26 attributes.

Its target value is a person accepting a recommended coupon, attribute **Y** which is again a binary classification with values 0 (a person not accepting a coupon) and 1 (person accepting a coupon), distribution of the two values can be seen in 3.4 (7210 participants have taken the offered coupon, while there is 5474 entries for participants who have refused to take one).

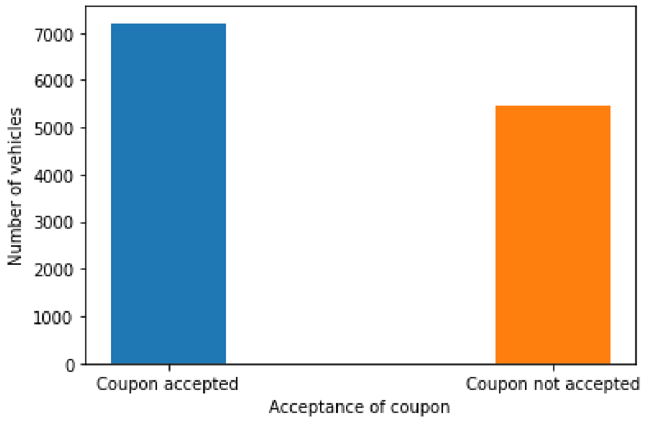


**Figure 3.4:** Balance of records in In-Vehicle coupon recommendation dataset

### 3.2.5 California housing dataset

California housing dataset is the only dataset among the ones analyzed that has a regression problem setting. It is loaded directly with `sklearn` package [44], and its target value is median house prices (**MedHouseVal**) in California districts, MedianHouseValue attribute. It has 20640 rows, and 9 attributes. Value movement can be shown with histograms that show probability distributions, so in 3.5 we see movement of the target value **MedHouseVal**.

### 3.2.6 Developer survey dataset

This is a dataset created by StackOverflow with the help of its annual survey for its users [45]. It contains 64461 rows and 61 columns of raw unclean

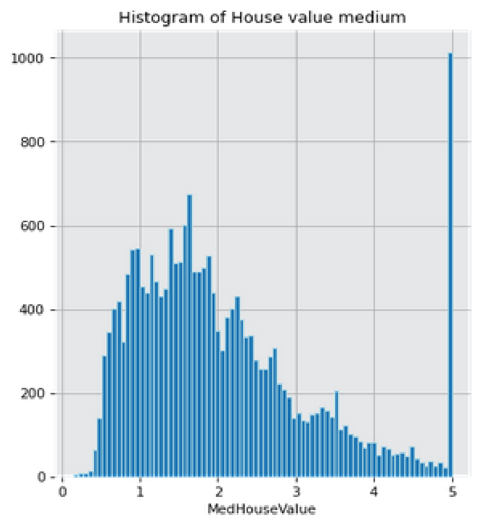


**Figure 3.5:** Histogram of MedHouseVal target value in California housing dataset

data given as answers to different questions about users' interests, desires, platforms they use at work, number of years they code, etc. Target values that were set up were column **Job Satisfaction** (determining how satisfied are the users with their job) and column **Job Seek** (questioning if the users are currently in search of a new job). Both are a multiple classification problem setting, and distribution of values can be seen in figure 3.6. As original dataset it had not been used in any of the analysis, because it had many irrelevant columns to the target values we had set, so the initial optimization done in Python had been used to create the starting point for all other integrations' analysis.

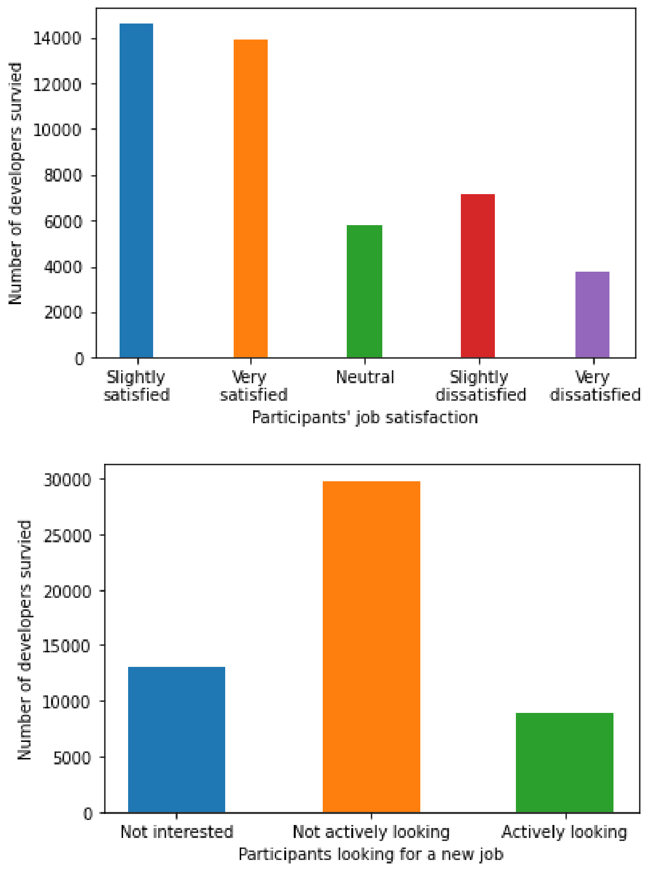


**Figure 3.6:** Distribution of values for developer survey participants about their level of satisfaction about their job, and distribution of participants who are looking for a new job

# Chapter 4

# Methodology

In the following chapter all processes involving preparing the data, installation of the needed toolsets and systems, the preformed initializing steps that were needed for further analysis are explained. The organization of sections is that explanation of each of the combinations of paired technologies is approached individually.

## 4.1 Python and SQLite

The installation of Python and SQLite was done in a Linux environment - Ubuntu 20.04 LTS, following original instructions. The set up Python version is Python 3.8.10, while the SQLite was sqlite3. All procedures were ran in a virtual environment, which was established by the help of Python's module called `venv` [46]. Venv was used so that all packages and libraries needed for this combination of tools do not interfere in any way with the system's functionalities. All of Python written code was executed with the help of JupyterNotebook [47]. JupyterNotebook is a web application that enables live code writing, visualization, notes presentation and many more, it also enables coding in 40 different programming languages. It is an open-source technology, which has a wide spread usage among data scientists, because it allows code execution of specific code parts after small corrections, while holding in memory previously ran variables, forming an iterative process between the data and data scientist [48].

After the basic Python environment set up, additional Python libraries needed for data cleaning, manipulation and presentation are:

- `pandas` - open source library intended for flexible and fast data analysis, allows imports from different data sources [23]
- `numpy` - library including various mathematical functions, fast vectorization, and for data storing it uses a lot less memory than traditional Python arrays [49]
- `sklearn` - tools used for ML functionalities, training a model and performing predictive analysis [5]
- `matplotlib` - library mostly used for data visualization and graphical plotting [50]
- `yellowbrick` - tool that extends the `sklearn` package, containing a set of diagnostic and visual analytic tools [51]
- `seaborn` - visualization library based on matplotlib, used mostly for statistical graphs [52].

Through following paragraphs, specific usage of each tool, library or module is explained. Process was as follows, the virtual environment `venv` was activated, inside which Jupyter Notebook was started. All continuing steps were done in the created Jupyter Notebook files.

The first step before any data transformation is to load data from the obtained csv file from mentioned sources into Python and create an sqlite table. The steps were essentially the same for all seven datasets we were using. We do so with the help of pandas library and its *read_csv* function which loads the csv data into pandas dataframe (data structure that offers changeable size and storing diverse tabular data, with indexed rows and columns [53]). For table creation in sqlite and filling the data, a new database connection was established (freely named "MasterAll.db")

`connection_name = sqlite.connect('MasterAll.db')`

Upon a successful connection, dataframe data is loaded into a new sqlite table.

```
dataframe.to_sql("table_name", con=connection_name,
if_exists='replace', index=False)
```

To be completely assured the correct data is used from that point onward, data is loaded directly from the sqlite table with a `select query` into a new dataframe.

`dataframe = pd.read_sql('SELECT * FROM table_name', connection_name)`

Some basic checks are performed - what type of data is stored in each column, and are there any null values, if there are, then how many. Type of data in columns is determined with following dataframe property: `dataframe.dtypes`. Sum of null values per column is calculated with: `dataframe.isnull().sum()` (this covers findings of Not a Number, NaN, values which are classified by pandas library out of missing values). The two checks are important for two reasons:

1. Python's machine learning library scikit-learn, does not use categorical data for predictions, hence the columns deemed relevant need to be transformed into numerical or binary data if they are of type object.

2. Missing values classified as NaN could lower the value of overall machine learning model's quality, so they need to be handled in some way, either the rows should be dropped, predicted with a machine learning algorithm or filled with data using statistical methods.

We will first explain how the missing values problem was tackled. Depending on the quantity of missing values, and its ratio towards overall number of rows, decision was made whether to delete the NaN values or use one of the statistical methods - mean, median or mode. Mean imputation is replacing all missing values with the mean value of the column, median imputation is replacing all with median value of the column, and ultimately mode with the most frequent value in the whole column. When the data in the column is skewed, median or mode imputations are used, as outlier data points could have a big impact on the mean value of the feature column [54]. There are three simple functions for achieving transformations: `dataframe.fillna(dataframe.mean())`, `dataframe.fillna(df.median())`, `dataframe['column_name'] = dataframe['column_name'].fillna (dataframe['column_name'].mode()[0])`. For mean values there is an additional method from `sklearn impute` module called `SimpleImputer` method:

```
from sklearn.impute import SimpleImputer
import numpy as np
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
dataframe[['column_name']] =
    imputer.fit_transform(dataframe[['column_name']])
```

For replacing categorical data with numerical there are many possible solutions. To name and explain a few [55]:

1. If the categorical data to be replaced is just words representing numbers, or a similar interconnection can be made, the technique to replace them with numerical values is simply done with pandas library `replace` function which takes as an input a dictionary of mapped words (keys) with numbers (values).

2. Label encoding is the technique which converts categorical values to numerical, the starting value is 0 up to the number of categories minus one, the encoding is performed in alphabetical ascending order, it can be done with `LabelEncoder` class from `sklearn` package and `preprocessing` module:

   ```
   from sklearn.preprocessing import LabelEncoder
   categorical_features = [list_of_columns]
   label_encoder = LabelEncoder()
   ```

```
for i in range(len(categorical_features)):
    new = label_encoder.fit_transform(
        dataframe[categorical_features[i]])
    dataframe[categorical_features[i]] = new
```

By creating an instance of the class and storing it in a variable, we go through the list of columns whose categorical values need to be replaced. With the fit_transform function, whole columns' values are replaced with new numerical ones.

3. One hot encoding encodes all categories in a numeric array of zeroes and ones by adding same amount of columns as there are possible values. It is a more accurate transformation as it confuses algorithms less with its precise category definition, but if the quantity of categories is large, it could significantly increase the table in size which would add more computation time in the end. Class `OneHotEncoder` exists in the `sklearn` package, and it can be used as:

```
from sklearn.preprocessing import OneHotEncoder
one_hot_encoder = OneHotEncoder(sparse=False,
                    handle_unknown='error',
                    drop='first')
new_dataframe = dataframe.DataFrame(
    one_hot_encoder.fit_transform(
        table_name[['column_name']]))
```

Further on, with different plotting options, we can determine which columns have an effect on the outcome of target value, and deem them relevant or irrelevant. One of those is correlation heatmap. Correlation is a statistical measure that shows how two variables are linearly connected, whether they change together equally [56]. `Seaborn's` heatmap helps us show relationships between numerical columns in a visual way. There is also `Seaborn's` violin plot that visually presents quantitative distribution of data across the categorical values, this can help make a connection with dataset's target value and dispensation of values for specific category - if the dispensation is equal for all target values then assumptions can be made it is a column with little effect on the outcome.

After data cleaning predictive ML models were built over the datasets. Several were taken into consideration as to find one that gives best results for each dataset. Used ML models for classification problem setting are:

- Logistic Regression (LR) - based on a statistical function, the logistic function. A method most successful when used to predict binary classification problems, it describes relationships between binary variable and nominal or interval independent variables [57], e.g. `from sklearn.linear_model import LogisticRegression`.
- Gaussian Naive Bayes (GNB) - based on the Bayes algorithm that follows Gaussian normal distribution and promotes continuous data. As a ML algorithm it works on a hypothesis that continuous values connected with each classification are distributed following Gaussian normal distribution [58], e.g. `from sklearn.naive_bayes import GaussianNB`.
- Linear Support Vector Classification (SVC) - creates its prediction by determining a hyperplane in n-dimensional space that has the the maximum distance between data of both classes. A well used algorithm because of its low computational power for significant accuracies [59] e.g. `from sklearn.svc import LinearSVC`.
- K-nearest Neighbors Classifier (KNN) - based on approach feature similarity, it does not make an assumption based on mathematical theorems to find data connections, but takes k amount of nearest neighbors to the point needed to be predicted and majority of defined classes is taken as predicted value, e.g. `from sklearn.neighbors import KNeighborsClassifier`.

- Decision Tree Classifier (DTC) - bases its predictions on certain rules, it uses dataset features to create yes and no tree nodes and questions, which ultimately help to segregate data points of each class, e.g. `from sklearn.tree import DecisionTreeClassifier`.

- Random Forest Classifier (RFC) - based on decision trees, but it is organized of many smaller decision trees that together work as a troupe. Each tree represents one predicted classification value, and the largest number of trees are the overall predicted value, the low correlation is what contributes to higher accuracies [60], e.g. `from sklearn.ensemble import RandomForestClassifier`.

- Gradient Boosting Classifier (GBC) - it is also part of the ensemble group using multiple lower predictors and boosting their accuracies with gradients, perfecting the predictor by learning on the precursor [61], e.g. `from sklearn.ensemble import GradientBoostingClassifier`.

While algorithms used for regression problem setting are:

- Linear Regression (LiR) - based on a linear equation, tried to be fitted to observed data while determining the existence of relationship between the data points, e.g. `from sklearn.linear_model import LinearRegression`.

- K-nearest Neighbors Regressor (KNNR) - same functionality as K-nearest Neighbors Classifier, e.g. `from sklearn.neighbors import KNeighborsRegressor`.

- Support Vector Regression (SVR) - uses the same principles as Linear Support Vector Classification with smaller irrelevant differences, e.g. `from sklearn.svm import SVR`.

- Bayesian Ridge Regression (BR) - linear regression formulated with probability distributors, its goal is to find rear distribution for model parameters [62], e.g. `from sklearn.linear_model import BayesianRidge`.

- Elastic Net (EN) - based on linear regression that uses regularized penalties, from the lasso and ridge techniques, by adding them to loss function during training [63], e.g. `from sklearn.linear_model import ElasticNet`.
- Gradient Boosting Regressor (GBR) - same principles as Gradient Boosting Classifier, e.g. `from sklearn.ensemble import GradientBoostingRegressor`.
- Decision Tree Regressor (DTR) - same principles as Decision Tree Classifier, e.g. `from sklearn.tree import DecisionTreeRegressor`.

As seven different datasets were used, the taken approach in cleaning and optimizing each of them is listed below. After optimization, we used the above mentioned ML algorithms.

### 4.1.1 Stroke dataset

It was mentioned that the Stroke dataset is an imbalanced one, but as it also has a smaller number of records, any null values that were discovered could not have been removed, as it would lower the already bad score. Especially since the 249 records who had the target value 1 (representing the patients who had suffered a stroke), 40 of those had a null value. The only column with missing values was BMI, a numeric float value. Using `sns.distplot(dataset_name.column_name)` plotting technique for determining the data distribution, we see that it is symmetrical 3.3.

Per this kind of distribution, mean statistical method was used with the help of before mentioned `SimpleImputer` method. With the missing values exchanged with mean numbers, what was left to be done was determine which columns contain categorical data of type object, and if those need to be exchanged with numerical values. All values from columns: Gender,

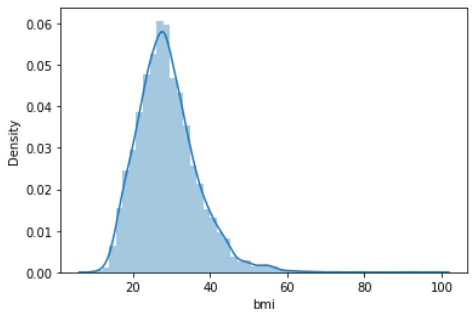


**Figure 4.1:** Distribution of values in BMI column

EverMarried, WorkType, ResidenceType and SmokingStatus, were converted to numerical ones, using the `LabelEncoder` class from `sklearn` package. Since the quantity of data was quite low we did not consider dropping any columns out.

### 4.1.2 Hotel satisfaction dataset

The Hotel Satisfaction dataset did not have any null values which made the process of data preparing much easier. Out of the seventeen columns only five had categorical values: Gender, PurposeOfTravel, TypeOfTravel, TypeOfBooking and Satisfaction. These were again transformed into numerical with the help of `LabelEncoder`.

### 4.1.3 Hotel booking datasets

Hotel booking dataset was made out of two as mentioned before, one referring to a city hotel and other to a resort one, but they both had same columns. The process of data preparation was the equal for the two, except Resort dataset being imbalanced, while the City dataset has the ratio of cancellation values overall the same. Both datasets contained multiple null values in three columns Agent, Company and Country. Missing values in Country column spread through all of the records, so this column was dropped completely, since it would make little impact on the overall prediction success. Agent and Country were also dropped because seaborn's violinplots and heatmaps showed correlations towards the target value of little influence. Additional dropped columns were: ArrivalDateDayOfMonth, ArrivalDateMonth, ArrivalDateWeekNumber, ArrivalDateYear and ReservationStatusDate, along with column Babies for Resort dataset, and Children for the City dataset.

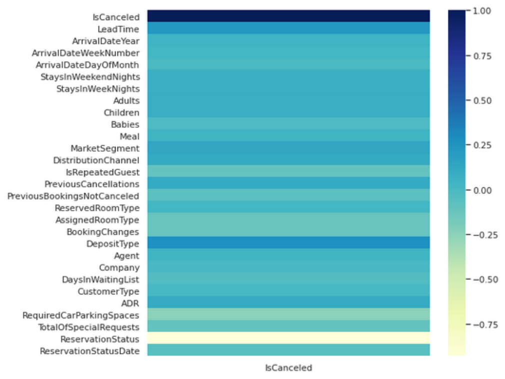


**Figure 4.2:** Heatmap representing correlations amongst the different columns towards the target value in dataset Resort

Columns Meal, ReservedRoomType, AssignedRoomType, DepositType, CustomerType, ReservationStatus, DistributionChannel and MarketSegment, were again transformed into numerical data as they contained categorical data with `LabelEncoder`.

### 4.1.4 In-vehicle coupon recommendation dataset

While analysing data for the In-vehicle coupon recommendation dataset it was noticed there are coupons offered for five different establishments - a Bar, a Coffee house, a Carry out and Take away restaurant, a Restaurant with prices up to 20 dollars and a Restaurant with prices between 20 and 50 dollars, seen in 4.3. Hence, besides the original full dataset, an additional separation of data into four parts was made: dataset "Bar", dataset "Coffee House", dataset "Carry out and Take away", dataset "Restaurants" (two

Restaurant values ("Restaurant<20" and "Restaurant(20-50)") merged into one dataset).

As there is also an attribute where participants had to make known their visit frequency to each of the offered establishments, this division was done to determine if the visit frequency relates to accepting the coupon for each establishment.

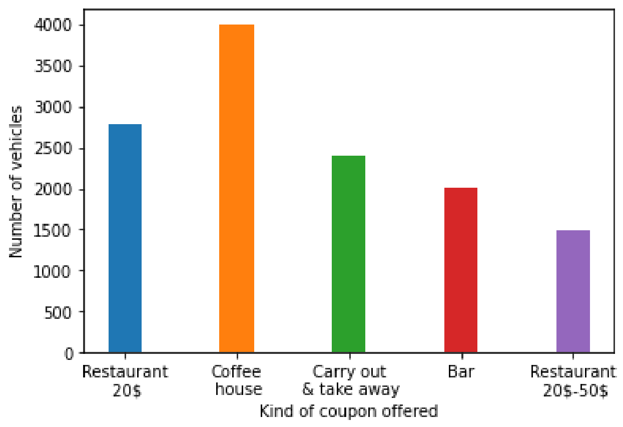


**Figure 4.3:** Distribution of different coupons that were offered in In-Vehicle coupon recommendation dataset

The data preparation for all datasets, prior to the division, was the same. The check for missing values returned that the column Car had value null for most of the entries (12576 out of 12684), columns determining visit frequencies also had some values missing for all the establishments. The column Car was dropped, as that many null values could not influence the prediction score. The other null values for visit frequencies were removed with pandas `dropna()` function. As some of the columns shared missing values, this resulted in dataset being left with 12079 records, and the balance of the target variable remaining the same (with 6877 participants that have taken the offered coupon, and 5202 that had not). Out of 25 remaining attributes, 17 of them were of type object. For nine of those we used pandas `replace` function which was previously mentioned, these were:

- Time, with following dictionary: {`'2PM':14, '10AM':10, '6PM':18, '7AM':7, '10PM':22`}
- Expiration, with dictionary: {`'1d':24, '2h':2`}
- Age, with dictionary: {`'below21':0, '21':1, '26':2, '31':3, '36':4, '41':5, '46':6, '50plus':7`}
- Income, with dictionary: {`'$37500 - $49999':3, '$62500 - $74999':5, '$12500 - $24999':1, '$75000 - $87499':6,'$50000 - $62499':4, '$25000 - $37499':2, '$100000 or More':8, '$87500 - $99999':7, 'Less than $12500':0`}
- Bar, CoffeeHouse, CarryAway, RestaurantLessThan20 and Restaurant20to50, all with the same dictionary: {`'never':0, 'less1':1, '1~3':2, '4~8':3, 'gt8':4`}

Remaining columns with categorical values: Destination, Passanger, Coupon, Gender, MaritalStatus, Education and Weather, were transformed using the `LabelEncoder`. Ultimately additional two columns, deemed of little importance, were dropped - ToCoupon_GEQ5min and Occupation. When the data was prepared and committed, we could divide it into predefined datasets with SQL queries, e.g. `coupons_bar = pd.read_sql('Select * from coupons where coupon = 0', connection_name)`, with 0 being the value assigned by the label encoder for establishment Bar.

### 4.1.5 California housing dataset

California housing is the only regression dataset, and its data was fetched directly from `scikit` library.

```
from sklearn import datasets
import numpy as np
import pandas as pd
ch = datasets.fetch_california_housing()
```

```
california_housing = pd.DataFrame(
    data= np.c_[ch['data'], ch['target']],
    columns= ch['feature_names'] + ['MedHouseVal'])
```

Since we fetched the data, we needed to save it from the dataframe to the open database SQL connection.

```
california_housing.to_sql(
    'california_housing',
    con=connection_name,
    if_exists='replace',
    index=False)
```

There was no missing values in this datasets, and all of the data was already in numerical form.

### 4.1.6 Developer survey dataset

Developer survey was the most unclean dataset, as it contains real world data collected in a survey. To reduce the quantity of data, we had to first remove some columns. The survey was observed through its pdf format obtained from the web page where all the data was available [45]. Since we have determined the target values to be the two columns related to participants seeking a new job (JobSeek) and their satisfaction with the current one (JobSat), questions in the survey related to Stackoverflow or the survey in general were marked as irrelevant (NEWOffTopic, NEWOtherComms, SOAccount, NEWSOSites, SOComm, SOPartFreq, SOVisitFreq, SurveyEase, SurveyLength), but also some columns about personal information, actions, interests or previous experience with certain tools and languages (Country, Ethnicity, Sexuality, Trans, NEWStuck, WelcomeChange, NEWPurchaseResearch, NEWPurpleLink, DatabaseDesireNextYear, DatabaseWorkedWith, LanguageDesireNextYear, LanguageWorkedWith, MiscTechDesireNextYear, MiscTechWorkedWith, NEWCollabToolsDesireNextYear, NEWCollabToolsWorked-With, PlatformDesireNextYear, PlatformWorkedWith, WebframeDesireNex-tYear, WebframeWorkedWith). All of these were dropped along with additional explanatory columns (Respondent, CompFreq, CompTotal, Currency-Desc, CurrencySymbol).

The missing value check returned that all of the columns have some null values, but most important for immediate resolving were those contained in target values columns, JobSat and JobSeek. With dropping the null values from the two we have resolved some in other columns, but many more remained. Since some of the columns with missing values were of numeric type (ConvertedComp, Age, Age1stCode, WorkWeekHrs, YearsCode, YearsCode-Pro), two separate approaches were taken. First was dropping the null values, and second was replacing this missing values with mean value (by the help of `SimpleImputer`). Idea was to determine if mean value made an impact in predictions, as it allowed for more records to remain. Exempted was the ConvertedComp column because it contains salaries of participants, and research done by job search magazines in Belgium found that paychecks play a great role in decision of changing jobs and in overall motivation to work and job satisfaction [64]. Hence, population of the column with mean value, or with another value, would possibly affect the prediction score badly, therefore null values for ConvertedComp were dropped in both approaches. Another four columns were dropped: NEWJobHunt, NEWJobHuntResearch, JobFactors and DevType. All of them were questions with multiple answers, with Dev-Type for example having 23 answering options, which led to a lot of different combinations given in the answers. Since there were so many, separating answers in multiple columns would greatly increase the size of the dataset, better option was then to completely remove them.

Three attributes had some of the values exchanged with pandas `replace` function: YearsCode (dictionary: {`'Less than 1 year': 0, 'More than 50 years': 51`}), YearsCodePro (dictionary: {`'Less than 1 year': 0, 'More than 50 years': 51`}) and Age1stCode (dictionary: {`'Older than`

`85': 86, 'Younger than 5 years': 4}`). Other values in the three columns were of numerical form, after the replacement data type was changed to numeric accordingly.

```
    dataset_name['column_name'] = pd.to_numeric(
                                        dataset_name['column_name'])
```

Other 17 columns that had categorical values (MainBranch, Hobbyist, EdLevel, Employment, Gender, JobSat, JobSeek, NEWDevOps, NEWDevOpsImpt, NEWEdImpt, NEWLearn, NEWOnboardGood, NEWOvertime, OpSys, OrgSize, PurchaseWhat, UndergradMajor) were changed with the help of `LabelEncoder` into numerical ones. This also resolved what was left of the null values. In the end, the dataset where we dropped all null values was left with 29865 records, while the one where we have used simple imputer had 33730 records.

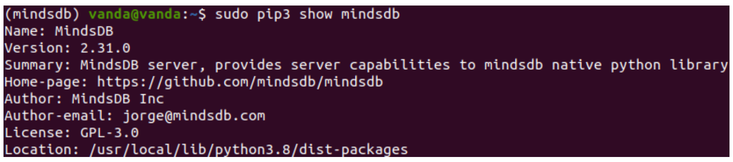
## 4.2 MindsDB and MariaDB

MindsDB set up was done, as in 4.1, in an Ubuntu 20.04 LTS operating system, following the instructions on its official pages [65]. It was installed through the terminal in a Python's virtual environment using pip commands. For MindsDB to work correctly, Python version must be 3.6.x, 3.7.x, or 3.8.x versions. MindsDB version that was set up was 2.31.0.

```
    Python -m venv mindsdb
    source mindsdb/bin/activate
    pip install mindsdb
```

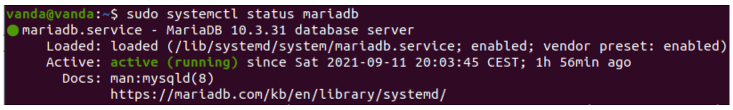
MariaDB was installed in similar fashion, again through the terminal with command `sudo apt install mariadb-server`.

Before connecting the two, we had to set up MariaDB properly. We ran the `sudo mysql_secure_installation` which runs the script to set up our root password, removes anonymous user and test database, as well as



```
(mindsdb) vanda@vanda:~$ sudo pip3 show mindsdb
Name: MindsDB
Version: 2.31.0
Summary: MindsDB server, provides server capabilities to mindsdb native python library
Home-page: https://github.com/mindsdb/mindsdb
Author: MindsDB Inc
Author-email: jorge@mindsdb.com
License: GPL-3.0
Location: /usr/local/lib/python3.8/dist-packages
```

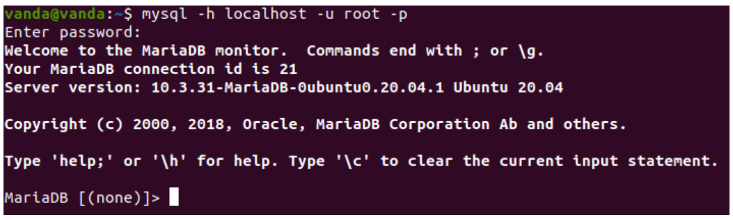
**Figure 4.4:** Terminal pip command inside the virtual environment named mindsdb checking whether MindsDB application was successfully installed



```
vanda@vanda:~$ sudo systemctl status mariadb
● mariadb.service - MariaDB 10.3.31 database server
     Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2021-09-11 20:03:45 CEST; 1h 56min ago
       Docs: man:mysqld(8)
             https://mariadb.com/kb/en/library/systemd/
```

**Figure 4.5:** Terminal command checking the status of MariaDB server - its successful set up and running status

restricts root user access only to the local machine. All of these are asked as questions through the terminal, and at the end it reloads the privileges table for all changes to take effect. When this was finished to connect to MariaDB and open the shell from the command line we run `mysql -h localhost -u root -p`, which prompts us to enter root user password 4.6.



```
vanda@vanda:~$ mysql -h localhost -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 21
Server version: 10.3.31-MariaDB-0ubuntu0.20.04.1 Ubuntu 20.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> █
```

**Figure 4.6:** Terminal command to connect to MariaDB

Afterwards a database was created with a command `CREATE DATABASE master_maria_db;`. In this database we store all tables representing our datasets. All tables were created the same way with command `CREATE OR REPLACE TABLE table_name`, with listing all columns and column types appropriate for MariaDB server, referent example given below for first dataset Stroke:

```
CREATE OR REPLACE TABLE stroke
```

```
(
    id                  INT NOT NULL,
    gender              VARCHAR(10) DEFAULT NULL,
    age                 INT(11) DEFAULT NULL,
    hypertension        INT(11) DEFAULT NULL,
    heart_disease       INT(11) DEFAULT NULL,
    ever_married        VARCHAR(5) DEFAULT NULL,
    work_type           VARCHAR(25) DEFAULT NULL,
    Residence_type      VARCHAR(15) DEFAULT NULL,
    avg_glucose_level   FLOAT DEFAULT NULL,
    bmi                 FLOAT DEFAULT NULL,
    smoking_status      VARCHAR(35) DEFAULT NULL,
    stroke              INT(11) DEFAULT NULL
)
engine=innodb
DEFAULT charset=latin1;
```

The table was then populated with command below. It defines how the fields are separated and lines terminated in the dataset's csv file, we also add to ignore the first line as it contains columns' names:

```
LOAD DATA LOCAL INFILE 'location_of_dataset_csv_file\stroke.csv'
INTO table master_maria_db.stroke
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

The process for other datasets was:

- **Hotel satisfaction** had no special features, created the same way as Stroke table.

- For **Hotel booking** naturally we created two tables with same columns, but populated with different data, one with csv file for the city hotel and the other with csv file for the resort hotel.
- **In-vehicle coupon recommendation** dataset was created in the same way as referent example, the division mentioned in 4.1.4 was done later on when creating the AI MindsDB tables.
- **California housing** table was created in same way as referent table, while the csv file was obtained with the help of Python's command `dataset_name.to_csv("location_on_disk.csv", index=False)` through the preparations in 4.1.5.
- For **Developer survey** four tables were created with different columns following the preparation in 4.1.6. This was done to lower the computation time because of the sheer amount of columns that were included in the original dataset. First one contained all columns from the first approach for missing values (after columns with numerical data containing null values were dropped), second one contained all columns from the second approach for missing values (numerical columns populated with mean value instead of null), third and fourth ones are the image of the first two mentioned datasets after columns representing questions with multiple answers were dropped. As MariaDB's import function does not manage to differentiate between coma signs in content of a column and where it is considered as delimiter between columns, we replaced coma sign inside content with space sign with the help of dataframes in Python. These replacement were afterwards used in PostgreSQL MindsDB integration and in Oracle as well.

Last prerequisite was installing the connect engine `sudo apt-get install mariadb-plugin-connect`, which enables MariaDB to access remote or external local data [66]. Now we finally connected MindsDB with MariaDB:

```
Python3 -m mindsdb --api=http,mysql --config=config.json
```

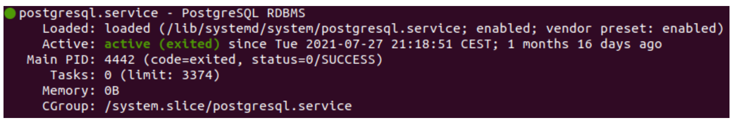
Original configuration file obtained from MindsDB website [67], had to be amended to fit the environment we had set up, specifically the json integration object, by adding json elements database, and changing the password to the one that was set up earlier as a root MariaDB password.

```
"integrations": {
    "default_mariadb": {
        "database": "master_maria_db",
        "publish": true,
        "enabled": true,
        "host": "localhost",
        "password": "password",
        "port": 3306,
        "type": "mariadb",
        "user": "root"}}
```

With this connection a schema is automatically created inside the MariaDB server named **mindsdb** containing two tables: **commands** and **predictors** (where all machine learning models are stored). Another way of making a connection between the two is using the MindsDB studio GUI, which is started through the command line with `Python3 -m mindsdb`. When we access the GUI we need to follow the next steps to reach data in our MariaDB database: click the button From database, click Add database, on the opened pop up pick MariaDB from the drop down menu on Supported Database, write down our database name, hostname, port, our MariaDB user and password, and lastly we click CONNECT. Then to load our data and create models of all our tables we need to add new datasource, by clicking on New Datasource button, then in the pop up Datasource from DB integration we add datasource name, database name, and the SELECT query from the MariaDB table (e.g. `SELECT * FROM table_name`), and we click Create.

## 4.3 MindsDB and PostgreSQL

MindsDB set up was precisely the same as done in 4.2, also in an Ubuntu 20.04 LTS operating system. PostgreSQL installation steps were followed from the original documentation [68] with command `sudo apt-get -y install postgresql`. Common check preformed to confirm the successful installation and running of the newly created server is `sudo systemctl status postgresql`, while we also confirm that Postgresql server is ready to accept connections from clients `sudo pg_isready`.



```
● postgresql.service - PostgreSQL RDBMS
     Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
     Active: active (exited) since Tue 2021-07-27 21:18:51 CEST; 1 months 16 days ago
   Main PID: 4442 (code=exited, status=0/SUCCESS)
      Tasks: 0 (limit: 3374)
     Memory: 0B
     CGroup: /system.slice/postgresql.service
```

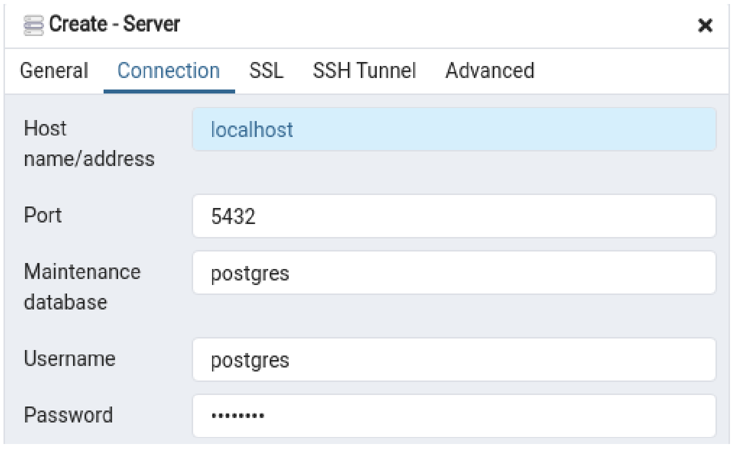
**Figure 4.7:** Terminal command to check PostgreSQL server status

To preform further actions inside PostgreSQL we access PostgreSQL database shell `psql` program from the command line with `sudo -u postgres psql` (annotation -u indicates we start the `psql` program with user postgres). A new database was created named postgres (`CREATE DATABASE postgres;`) where all tables of datasets were to be placed in. While we kept original system user postgres for all further actions and connections, we had only altered it by adding a new password. For easier interaction with PostgreSQL database, pgAdmin4 was installed. PgAdmin4 is an open source management tool, with a web user interface to preform any administrative actions with PostgreSQL databases and local and remote servers [69]. Its installation process was followed from its dedicated download website [70], since its not available in Ubuntu repositories, we had to set up the pgAdmin4 APT repository first by adding the public key for it and creating the repository's configuration file, and then we could preform the installation.

```
#sudo curl https://www.pgadmin.org/static/packages_pgadmin_org.pub
| sudo apt-key add
#sudo sh -c 'echo "deb https://ftp.postgresql.org/pub/pgadmin/
```

```
pgadmin4/apt/$(lsb_release -cs) pgadmin4 main" >
/etc/apt/sources.list.d/pgadmin4.list && apt update'
#sudo apt install pgadmin4
```

Upon first start of pgAdmin, master password was set up to secure all our credentials and other passwords. Then a new server was added to connect to our PostgreSQL database server seen in 4.8.


Create - Server

General Connection SSL SSH Tunnel Advanced

| | |
|---|---|
| Host name/address | localhost |
| Port | 5432 |
| Maintenance database | postgres |
| Username | postgres |
| Password | •••••••• |

**Figure 4.8:** Create server pop up window in pgAdmin

Before connecting our PostgreSQL server to MindsDB, another prerequisite was fulfilled - the installation of MySQL foreign data wrapper. Foreign data wrapper is used to access tables on a remote database (data source) and communicate with it while hiding the details of this connection and obtaining data from it. Installation is done from the command line (`sudo apt-get install postgresql-9.5-mysql-fdw`), but before we can use the wrapper few additional steps had to be done inside our database shell `psql` program:

1. creation of extension to our foreign wrapper:
   `CREATE EXTENSION mysql_fdw;`

2. creation of a foreign server to remote database we want to connect to:
   `CREATE SERVER server_mindsdb FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'localhost', port '47335');`

3. creation of a user mapping which associates postgres role with the foreign server, it summarizes connection information needed by the foreign-data wrapper to access an external data resource:
   ```
   CREATE USER MAPPING FOR postgres SERVER server_mindsdb options
   (user 'postgres', password 'password');
   ```

Following the installation, MindsDB and PostgreSQL were connected in the same way as MindsDB and MariaDB in previous section 4.2. We edited the configuration file to accommodate necessities for PostgreSQL. Upon successful connection the previously mentioned **mindsdb** schema was created automatically.

```
integrations": {
   "default_postgres": {
        "database": "postgres",
        "publish": true,
        "host": "localhost",
        "password": "password",
        "port": 5432,
        "type": "postgres",
        "user": "postgres"}}
```

Tables for datasets were all created similarly through pgAdmin GUI, with a referent example of the first dataset Stroke creation given below. To populate the created tables we use the `copy` function while defining the type of delimiter, what is the format of null values inside the dataset, and that the given file contains a header. Not all datasets had the same format of null values, hence this was changed accordingly (Hotel booking datasets had 'NA', California housing had none so this was omitted, but it was also omitted for In-vehicle coupon recommendation and Developer survey dataset as they had a few different representations throughout the columns). The rest of the tables' data was obtained in the same way as mentioned in 4.2.

```
create table public.stroke(
```

```
    id integer,
    gender varchar(10),
    age numeric,
    hypertension numeric,
    heart_disease integer,
    ever_married varchar(3),
    work_type varchar(25),
    Residence_type varchar(25),
    avg_glucose_level numeric,
    bmi numeric,
    smoking_status varchar(25),
    stroke integer
);
COPY stroke
FROM 'location_of_dataset_csv_file/stroke.csv'
WITH(FORMAT CSV, DELIMITER ',', NULL  'N/A', HEADER);
```

## 4.4 Oracle Machine Learning

Oracle is not currently supported in a debian ubuntu operating system, hence we had to change the installation destination to a CentOS Linux 8 derived from Red Hat enterprise Linux. As was mentioned the Oracle express edition was used, specifically the Oracle database XE 18c. Since we are not using Oracle Linux, a pre-intsallation package was obtained, which does not come with original RPM (Red Hat Package Manager) installation file. In this situation we had a few more prerequisites to install, those were gcc-c++ (the GNU compiler), ksh (language that reads commands from the terminal or file and executes them), sysstat (collection of utilities designed to care and monitor for the server), xorg-x11-utils (collection of client utilities to query the X server), libnsl (library that provides an interface for different network services) - all of these were installed through command line (`#yum install`

`-y name_of_prerequisite`). While the last two were procured through [71] website, and installed locally:

- `#yum localinstall compat-libcap1-1.10-7.el7.i686.rpm` - library for getting and setting POSIX.1e capabilites.
- `#yum localinstall compat-libstdc++-33-3.2.3-72.el7.i686.rpm` - c++ compatibility standard library.

Then the pre-install rpm was executed.

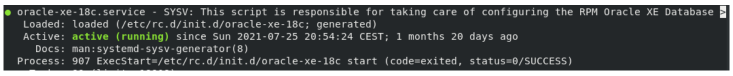
```
#curl -o oracle-database-preinstall-18c-1.0-1.el7.x86_64.rpm
https://yum.oracle.com/repo/OracleLinux/OL7/latest/x86_64/
getPackage/oracle-database-preinstall-18c-1.0-1.el7.x86_64.rpm

#yum -y localinstall
    oracle-database-preinstall-18c-1.0-1.el7.x86_64.rpm
```

And the database software installed, previously downloaded of the official sites for Linux system [72]:

```
# yum -y localinstall oracle-database-xe-18c-1.0-1.x86_64.rpm
```

We ran the service configuration script to set up the main passwords, and the Oracle database was set up as seen in 4.9.



```
● oracle-xe-18c.service - SYSV: This script is responsible for taking care of configuring the RPM Oracle XE Database >
   Loaded: loaded (/etc/rc.d/init.d/oracle-xe-18c; generated)
   Active: active (running) since Sun 2021-07-25 20:54:24 CEST; 1 months 20 days ago
     Docs: man:systemd-sysv-generator(8)
  Process: 907 ExecStart=/etc/rc.d/init.d/oracle-xe-18c start (code=exited, status=0/SUCCESS)
```

**Figure 4.9:** Command `#systemctl status oracle-xe-18c` gives us the current status of the Oracle server.

Oracle Data Miner is a GUI to Oracle Data Mining, a feature of Oracle Database, For all further operations and usage of the Machine learning Oracle integration we installed SQL Developer, as an application for simplification of to be done actions. Installation instructions on the official site [36] were followed. To actually use the ML through the SQL Developer Oracle data miner extension was activated which with is interactive workflow enables creating, evaluating, modifying, sharing, and deploying different methodologies, the features of Oracle data mining. Even though it is included in the Oracle express edition, a side installation is required. After creating a sys admin connection through the developer, a new data mining user was created and granted certain privileges to be able to preform data mining actions.

```
CREATE USER name_of_user IDENTIFIED BY password
   DEFAULT TABLESPACE USERS
   TEMPORARY TABLESPACE TEMP
   QUOTA UNLIMITED ON USERS;
GRANT CREATE MINING MODEL TO name_of_user;
GRANT SELECT ANY MINING MODEL TO name_of_user;
GRANT CREATE SESSION TO name_of_user;
GRANT CREATE TABLE TO name_of_user;
GRANT CREATE VIEW TO name_of_user;
```

Afterwards, a connection for that user was created, and the Data miner repository installed using an automated process [73]:

1. in Tools tab we click Data miner and make it visible
2. create a new connection on the newly appeared Data miner tab
3. double clicking the connection from previous step Repository installment process is begun (User was set as default tablespace, and Temp as the temporary one).

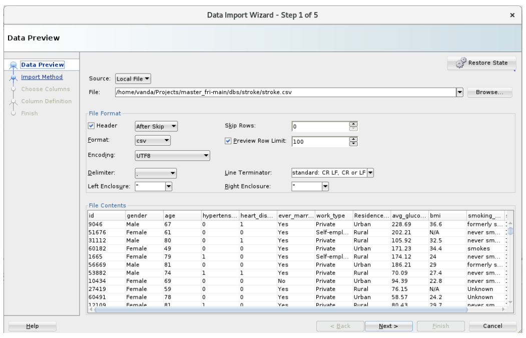
To preform data mining, the dataset file sources had to be imported into the data connection with data miner user, after creating all the tables (referent creation given below for the first dataset Stroke). The rest of the datasets were obtained in the same way as mentioned in 4.2. In building models inside Data miner extension all tables need a Case ID column, since some did not have it originally an extra column ID with a primary key keywords was added in create table query. On this column we added a column sequence so that with importing data the value is auto-incremented.

```
create table stroke(
    id number(10),
    gender varchar(10),
    age number(10),
    hypertension number(10),
    heart_disease number(10),
    ever_married varchar(3),
    work_type varchar(25),
    residence_type varchar(25),
    avg_glucose_level float(10),
    bmi varchar(10),
    smoking_status varchar(25),
    stroke number(10)
);
```

We imported data with the help of SQL Developer through the import wizard where conditions are defined (about the delimiter, number of rows, etc.) in 4.10. The import method is too defined through the wizard, as well as matching the columns (this also immediately gives notice if the types are mismatching).

The ML algorithms used for classification datasets in Oracle are adjusted by Oracle from original algorithms, so tuning models or data preparation is mostly redundant as Oracle minimized these actions.

- General linear model (GLM) - includes multiple regression models, that enables target variable to have an error distribution different to normal, and builds relationship between the target variable and features even if their relationship is not linear. Oracle has automatic data preparation set here in place. [74].
- Support vector machine (SVM) - as explained in 4.1, while Oracle's fine



Source: Local File
File: /home/vanda/Projects/master_fri-main/dbs/stroke/stroke.csv

File Format
[✓] Header After Skip
Skip Rows: 0
Format: csv
[✓] Preview Row Limit: 100
Encoding: UTF8
Delimiter: ,
Line Terminator: standard: CR LF, CR or LF
Left Enclosure: "
Right Enclosure: "

File Contents

| id | gender | age | hypertens... | heart_dis... | ever_marr... | work_type | Residence... | avg_gluco... | bmi | smoking_... |
|---|---|---|---|---|---|---|---|---|---|---|
| 9046 | Male | 67 | 0 | 1 | Yes | Private | Urban | 228.69 | 36.6 | formerly s... |
| 51676 | Female | 61 | 0 | 0 | Yes | Self-empl... | Rural | 202.21 | N/A | never sm... |
| 31112 | Male | 80 | 0 | 1 | Yes | Private | Rural | 105.92 | 32.5 | never sm... |
| 60182 | Female | 49 | 0 | 0 | Yes | Private | Urban | 171.23 | 34.4 | smokes |
| 1665 | Female | 79 | 1 | 0 | Yes | Self-empl... | Rural | 174.12 | 24 | never sm... |
| 56669 | Male | 81 | 0 | 0 | Yes | Private | Urban | 186.21 | 29 | formerly s... |
| 53882 | Male | 74 | 1 | 1 | Yes | Private | Rural | 70.09 | 27.4 | never sm... |
| 10434 | Female | 69 | 0 | 0 | No | Private | Urban | 94.39 | 22.8 | never sm... |
| 27419 | Female | 59 | 0 | 0 | Yes | Private | Rural | 76.15 | N/A | Unknown |
| 60491 | Female | 78 | 0 | 0 | Yes | Private | Urban | 58.57 | 24.2 | Unknown |
| 12109 | Female | 81 | 1 | 0 | Yes | Private | Rural | 80.43 | 29.7 | never sm... |

**Figure 4.10:** Data import wizard in SQL Developer for defining the import data into a table process.

tuning is seen in algorithm's ability to reduce training size as needed by using stratified sampling, or its ability to determine appropriate tuning settings based on our data [74].

- Decision tree (DT) - basics as explained in 4.1. In Oracle, data preparation is done internally.

- Naive Bayes (NB) - using Bayes formula, gives probability of a current label based on probability of the previous label, essentially counts occasions where this two labels occurred together and divides it by the label occurring alone. The Automatic data preparation uses supervised binding in Oracle, meaning it creates optimal bin boundaries using decision trees. [74]

# Chapter 5

# Results

The testing and running of all combinations were done in a Linux virtual machine, specific pointed out in corresponding sections in 4, set up through Oracle VirtualBox tool on a Windows 10 with AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx, 2100 Mhz, 4 cores and 8 logical processors with 8GB RAM. Each virtual machine was configured with four virtually assigned AMD Ryzen 4x processors and base memory of 3000MB. Certain procedures and documentation available at [75].

This chapter explains all methods of evaluation that were applied to machine learning algorithms results' to compare and conclude the differences, and results that were received.

## 5.1 Evaluation methods

To understand the success of different database system's ML capabilities we have used a few various evaluation metrics. These differentiate depending if analyzed dataset has a regression or a classification problem setting. Universal metric for all was the training time measurement. In Python it was executed by importing `time` module, with whose function `time()` we assigned current time to a variable before ML algorithm execution and after it was finished, subtracted starting variable from then current time. MindsDB execution of algorithms comes with measuring duration seen on its web GUI during training. While Oracle ML has timestamps in details of trained model, the built and tested timestamp.

### 5.1.1 Classification problem evaluation metrics

As classification problems are essentially used to predict specific class labels, the correctness of this prediction is determined. Accuracy is the main evaluation metric in these settings. It is calculated by dividing correct predictions and overall number of input records. But more often than not the F1 score contains more information than the actual accuracy. Values needed for F1 score calculation are model's precision (number of correct positive results divided by overall positives predicted by the model) and recall (number of correct positives divided by all records that should have been predicted as positive by the model). The two metrics can be calculated with the help of a confusion matrix, which shows complete performance of the model. F1 score is then a harmonic mean between the two values. Value 1 represents perfect assignment in class's predictions [76]. In Python a `classification_report` imported from module `sklearn.metrics` measures the quality of predictions, containing overall accuracy, F1 score of classification labels and division of records between classes. Another useful metric is Area under the curve (AUC) - Receiver operator characteristic (ROC), ROC is the probability curve, while AUC measures the ability of separating different classes, it is a summary of ROC curve values. Value 1 here represents a model that perfectly distinguishes between classes [77]. In Python we imported `roc_curve` and `auc` from same module as previously stated, `sklearn.metrics`, while in Oracle Data miner it was included by default in the model analysis as a metric. It can only be used for binary classification problems.

### 5.1.2 Regression problem evaluation metrics

Regression problem's goal is prediction of a continuous quantity by approximating output value from input parameters. Hence, the output is a real value (integer or float values) and accuracy cannot be calculated for regression problem setting, as the quantity of possible values is enormous. Therefore, we measure the success with calculating an error in its predictions. Most often used error metrics are the following ones:

- Mean Squared Error (MSE) - calculated as a mean value of squared differences between predicted and expected values, perfect value is 0.0 meaning all values were predicted correctly.
- Root Mean Squared Error (RMSE) - calculated by performing a square root over mean squared error to maintain the same units as original data, perfect value is therefore here too 0.0.
- Mean Absolute Error (MAE) - calculated as average of absolute error values between expected and predicted value, unlike MSE and RMSE it does not give higher or lower weight to bigger or smaller errors, again the perfect score is 0.0.

All three error metrics are in Python imported through the `sklearn.metrics` module, respectively `mean_squared_error`, RMSE is same library as MSE with additional input parameter of `squared=False` and `mean_absolute_error`.

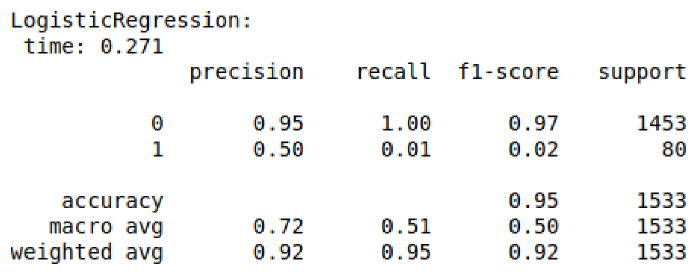
## 5.2 Training the models

For Python / SQLite tool combination multiple algorithms were tested (listed in 4.1), but for relevance we only visualized in figures best results, hence only one algorithm with highest accuracy and best training time was mentioned in 5.3. To train the algorithms, data had to be separated into two arrays: `X` holding the feature attributes' values and `y` holding the target value column. From `sklearn` package and `model_selection` module, `train_test_split` class is imported. This class takes multiple parameters, the ones we have inputted are the two arrays (`X` and `y`), `test_size` and `random_state`. The variables it returns are values from `X` to train and to test on (`x_train`, `x_test`), and values from `y` to train and to test on (`y_train`, `y_test`). In general, ratio for testing and training used was 30 to 70 (in rare cases it made a difference when the ratio was set 25 to 75, or 20 to 80 but it was minimal), hence the variable test size was 0.3, while the random state was put at a random value of 7. The basic run of each algorithm can be seen below (Logistic regression algorithm as a referent example). Each algorithm's run was surrounded with `time` function to measure training time. The other possible parameters are detailed in datasets' subsesctions.

```
start_time_lr = time.time() #assigns current time to variable
lr = LogisticRegression().fit(x_train, y_train) #fits / builds the model from the training set
predictions_lr = lr.predict(x_test) #predicts labels or values for given samples
elapsed_time_lr = time.time() - start_time_lr #calculates time passed from building the model to predicting values
```

Accuracies for built module are visualized with `classification_report` class from `sklearn` package and `metrics` module. We give it two parameters, the `y_test` values and received predicted labels / values `predictions_algorithm`. The values it shows to the user can be seen in 5.1.

For both MindsDB integrations (MariaDB and PostgreSQL) models for each dataset had to be inserted into the automatically created **predictors** table in **mindsdb** schema. The insertion had the same `insert` statement with values adjusted accordingly for each dataset. Upon inserting values in the predictors table the model is automatically trained. The predictors table can be queried to check whether the model's training was completed and to see model's accuracy, even though in our analysis for these checks we have mostly used Studio GUI because we wanted the concrete duration of training



```
LogisticRegression:
 time: 0.271
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1453
           1       0.50      0.01      0.02        80

    accuracy                           0.95      1533
   macro avg       0.72      0.51      0.50      1533
weighted avg       0.92      0.95      0.92      1533
```

**Figure 5.1:** An example of classification report values for one algorithm (e.g. logistic regression).

time. After completed training, the model's table is published in the same mindsdb schema, the table is considered to be MindsDB's AI table. This essentially means, that by generating specific queries with `where` clauses we can query the trained model and receive predictions on data not contained in original records. The division ratio was 20 to 80, training and testing data.

```
INSERT INTO
    mindsdb.predictors(name, predict, select_data_query)
VALUES ('dataset_model_name',
        'dataset_target_value',
        'SELECT * FROM dataset_name);
```

In Oracle Data miner SQL Developer's extension was used. We created a workflow where we trained models over all datasets individually. Data sources were added with Data source node option in the Components tab under Data category. Under the Model category, different models are offered that can be used for training, the two we concentrated on are the Classification and Regression model. In Classification model four algorithms are built by default, as mentioned in 4.4, we here repeated as done for Python algorithms and visualized results only for the algorithm with highest accuracy. While the Regression model had two algorithms the Generalized linear model (GLM) and Support vector machine (SVM). When connecting the models with data sources in the workflow we had to define the Target value and Case ID, we kept the test and training split ratio on 30 to 70.

## 5.3 Analysis of results

Now the results for each dataset will be analyzed individually for all systems and accompanying tools.

### 5.3.1 Stroke dataset results

The algorithm with best accuracy for the Stroke dataset in Python is the GNB, with its default parameter values. Since Stroke is an imbalanced dataset, we paid attention not only to the overall F1 score accuracy, but also to each label's F1 score accuracy. GNB had also the fastest training time and highest ROC curve value. MariaDB's integration with MindsDB produced quite low result, in particular with predicting the less represented label where the overall accuracy was 0%. Though, the PostgreSQL's integration had a higher score, even though in prediction of less represented label it still performed bad. MariaDB had slightly faster training time than PostgreSQL, but quite slow compared to other two tools. While in Oracle's Data miner DT gave best overall accuracy. It also had best results for prediction of both labels equally. ROC was highest with the SVM model (value 0.862).

| Tools | Overall accuracy | Training time (s) | ROC |
|---|---|---|---|
| Python/SQLite | 0.86 | 0.007 | 0.64 |
| MindsDB/MariaDB | 0.517 | 393 | - |
| MindsDB/PostgreSQL | 0.727 | 491 | - |
| Oracle/Oracle ML | 0.737 | 45 | 0.83 |

**Table 5.1:** Visualisation of values for Stroke dataset.

### 5.3.2 Hotel satisfaction dataset results

Results for Hotel Satisfaction dataset are the following, in Python despite that the highest accuracy was achieved with RFC and GBC (GBC reached same result with setting input parameters max_depth=5 and learning_rate=0.4, but had twice as long training time than RFC), and that the fastest training was with GNB (value 0,049, but overall accuracy was lower), as best algorithm DTC was picked because it had the best ratio between high accuracy and low training time. ROC curve was of course highest for RFC and GBC (value 0,95). MariaDB and PostgreSQL had similar accuracies, but PostgreSQL had quite faster training time than MariaDB for this dataset. While in Oracle data miner, DT algorithm had best overall accuracy. Area under curve was highest for the same algorithm.

| Tools | Overall accuracy | Training time (s) | ROC |
|---|---|---|---|
| Python/SQLite | 0.93 | 0.290 | 0.93 |
| MindsDB/MariaDB | 0.951 | 752 | - |
| MindsDB/PostgreSQL | 0.942 | 62 | - |
| Oracle/Oracle ML | 0.91 | 50 | 0.966 |

**Table 5.2:** Visualisation of values for Hotel satisfaction dataset.

### 5.3.3 Hotel booking dataset results

Hotel booking datasets were the cleanest, by results it was proven that they had also had most dependable data for predicting the target values. For City dataset in Python three algorithms had accuracy equal to 1 (DTC, RFC and GBC), out of which fastest was DTC. Even though GNB had best training time (value 0,047) but its accuracy had not reached one (value 0,99). In the Resort dataset we had similar situation, DTC, RCF and GBC had highest possible accuracy, fastest out of the three was again DTC, and fastest overall was GNB (value 0.028) but its accuracy again by 0.01 lower than the three. ROC curve was therefore equal to 1 for both datasets. MariaDB had accuracies close or equal to one for both datasets, and PostgreSQL had the same. Training time was here too quite longer than other solutions. In

Oracle absolutely all algorithms produced 100% overall accuracy, and 1 as ROC, for both City and Resort dataset.

| Tools | Overall accuracy | Training time (s) | ROC |
|---|---|---|---|
| **City dataset** | | | |
| Python/SQLite | 1 | 0.126 | 1 |
| MindsDB/MariaDB | 1 | 714 | - |
| MindsDB/PostgreSQL | 1 | 887 | - |
| Oracle/Oracle ML | 1 | 40 | 1 |
| **Resort dataset** | | | |
| Python/SQLite | 1 | 0.038 | 1 |
| MindsDB/MariaDB | 0.995 | 945 | - |
| MindsDB/PostgreSQL | 1 | 530 | - |
| Oracle/Oracle ML | 1 | 51 | 1 |

**Table 5.3:** Visualisation of values for Hotel booking datasets.

### 5.3.4 In-vehicle coupon recommendation dataset results

For the In-vehicle coupon recommendation dataset, as we had mentioned, we divided the data based on the establishment for which a coupon had been offered. Results are seen in 5.4.

In Python with select queries we created, for each establishment, a separate panda's dataframe 4.1.4 and then used those for the `train_test_split` function. In algorithms ran over all data, highest accuracy was with GBC, by setting specific parameters: `loss = 'exponential', max_depth = 10, learning_rate = 0.75, n_estimators = 120`. Fastest was NB (value 0,012) but its accuracy was by 14% lower. For "Bar" highest accuracy was again with GBC with same set parameters, despite max_depth set to 5 and n_estimators to 150. Fastest was again NB (value 0,006) but with lower accuracy. Following same steps, highest accuracy was with GBC and fastest time with NB for all other establishments, i.e. for "Coffee House" (parameters set `max_depth = 10, learning_rate = 0.75`), "Carry away & take out" (parameters set `max_depth = 5, learning_rate = 0.4`, and as it was an imbalanced set the algorithm with highest accuracy for the less represented label was chosen) and "Restaurants" (parameters set `max_depth = 15, learning_rate = 0.75`).

For MindsDB's integrations we trained it for four different establishments by adjusting the `select` query for proper columns and appropriate values in the insertion in predictors. Highest accuracy in MariaDB was in prediction of values for the "Bar" establishment, while the worst was for the "Carry away & take out". PostgreSQL had similar results as seen in **??**.

In Data miner four different select queries were run over the original dataset selecting data accordingly for each establishment, over which a model was then built. Algorithm that reached highest accuracy with training of models over all data was DT, while ROC was highest with SVM (value 0.737). For "Bar" this was the SVM model, while ROC was best with GLM (value 0.797). For the other three, GLM had highest accuracies, and SVM highest areas under the curve.

### 5.3.5 California housing dataset results

California housing is the only regression dataset hence here we are comparing MSE, RMSE and MAE metrics. Since MariaDB and PostgreSQL with MindsDB integration do not have appropriate metrics set up they cannot be accurately compared with the other two. Nevertheless, we did input in the table its received accuracies 5.5. In Python all three metrics were the lowest with GBR. While fastest was BR algorithm (value 0.024), but with much worse metrics' results. In Oracle MAE was best with SVM, while RMSE with GLM. Results are seen in 5.5.

### 5.3.6 Developer survey dataset results

For developer survey the division into multiple datasets was done with the two approaches towards missing values. Results are seen in 5.6.

In Python we had only two datasets. In both datasets we tested accuracies for two columns JobSat and JobSeek. The first approach gave for JobSat column quite worse results, below 50% with GBC algorithm, while the second approach had accuracy 1 for GBC, DTC and RFC algorithms, of which the fastest was DTC. While results for the JobSeek columns were similar for both approaches (differentiating by 0.01) with RFC and GBC algorithm, of which RFC was faster in both cases. The only algorithm unable to perform was the KNN, the jupyter notebook tended to freeze and kernel automatically restarted without completing the task.

As for both MindsDB integrations, we wanted to test accuracies on datasets where the columns with longer strings were not yet dropped, but the GUI tended to freeze and training lasted for a couple of hours without reaching results, before it was cancelled. Therefore, we again only had predictions for the same two datasets as for Python. JobSat column for both approaches had below 50% accuracy, while the JobSeek column in the contrary had accuracy of 1. The results were similar for MariaDB and PostgreSQL, though PostgreSQL had faster training time.

Oracle was the only tool where we could test behaviour and prediction accuracy with columns containing longer strings. In the end here we created models over four data sources, two for each approach, one with dropped longer strings columns and other where these had not been removed. Here too, as per MindsDB integrations, all JobSeek predictions had accuracy of 1, in most cases all three algorithms achieved same result. While the JobSat predictions remained below the 50% accuracy line.

| Tools | Overall accuracy | Training time (s) | ROC |
|---|---|---|---|
| **All data** | | | |
| Python/SQLite | 0.75 | 10.529 | 0.74 |
| MindsDB/MariaDB | 0.74 | 1020 | - |
| MindsDB/PostgreSQL | 0.749 | 802 | - |
| Oracle/Oracle ML | 0.678 | 156 | 0.726 |
| **Bar** | | | |
| Python/SQLite | 0.77 | 0.702 | 0.76 |
| MindsDB/MariaDB | 0.769 | 75 | - |
| MindsDB/PostgreSQL | 0.747 | 101 | - |
| Oracle/Oracle ML | 0.718 | 38 | 0.788 |
| **Coffee house** | | | |
| Python/SQLite | 0.74 | 3.884 | 0.74 |
| MindsDB/MariaDB | 0.731 | 206 | - |
| MindsDB/PostgreSQL | 0.742 | 167 | - |
| Oracle/Oracle ML | 0.712 | 48 | 0.773 |
| **Carry out & take away** | | | |
| Python/SQLite | 0.73 | 0.477 | 0.60 |
| MindsDB/MariaDB | 0.59 | 69 | - |
| MindsDB/PostgreSQL | 0.617 | 151 | - |
| Oracle/Oracle ML | 0.622 | 36 | 0.664 |
| **Restaurants** | | | |
| Python/SQLite | 0.70 | 5.243 | 0.69 |
| MindsDB/MariaDB | 0.738 | 127 | - |
| MindsDB/PostgreSQL | 0.733 | 125 | - |
| Oracle/Oracle ML | 0.692 | 39 | 0.752 |

**Table 5.4:** Visualisation of values for In-vehicle coupon recommendation dataset.

| Tools | MSE | RMSE | MAE | Training time (s) | Accuracy |
|---|---|---|---|---|---|
| Python/SQLite | 0.269 | 0.519 | 0.365 | 5.404 | - |
| MindsDB/MariaDB | - | - | - | 928 | 0.851 |
| MindsDB/PostgreSQL | - | - | - | 212 | 0.86 |
| Oracle/Oracle ML | - | 0.519 | 0.762 | 30 | - |

**Table 5.5:** Visualisation of values for California housing dataset.

| Tools | Overall accuracy | Training time (s) |
|---|---|---|
| 1st approach longer string columns not dropped (JobSat/JobSeek) | | |
| Python/SQLite | - | - |
| MindsDB/MariaDB | - | - |
| MindsDB/PostgreSQL | - | - |
| Oracle/Oracle ML | 0.391 / 1 | 346 |
| 1st approach longer string columns dropped (JobSat/JobSeek) | | |
| Python/SQLite | 0.44 / 0.63 | 30.962 / 7.258 |
| MindsDB/MariaDB | 0.304 / 1 | 2076 / 5006 |
| MindsDB/PostgreSQL | 0.332 / 1 | 1575 / 1380 |
| Oracle/Oracle ML | 0.404 / 1 | 31 |
| 2nd approach longer string columns not dropped (JobSat/JobSeek) | | |
| Python/SQLite | - | - |
| MindsDB/MariaDB | - | - |
| MindsDB/PostgreSQL | - | - |
| Oracle/Oracle ML | 0.389 / 1 | 565 |
| 2nd approach longer string columns dropped (JobSat/JobSeek) | | |
| Python/SQLite | 1 / 0.64 | 0.069 / 6.467 |
| MindsDB/MariaDB | 0.304 / 1 | 5400 / 5402 |
| MindsDB/PostgreSQL | 0.335 / 1 | 1814 / 1593 |
| Oracle/Oracle ML | 0.377 / 1 | 37 |

**Table 5.6:** Visualisation of values for Developer survey dataset.

# Chapter 6

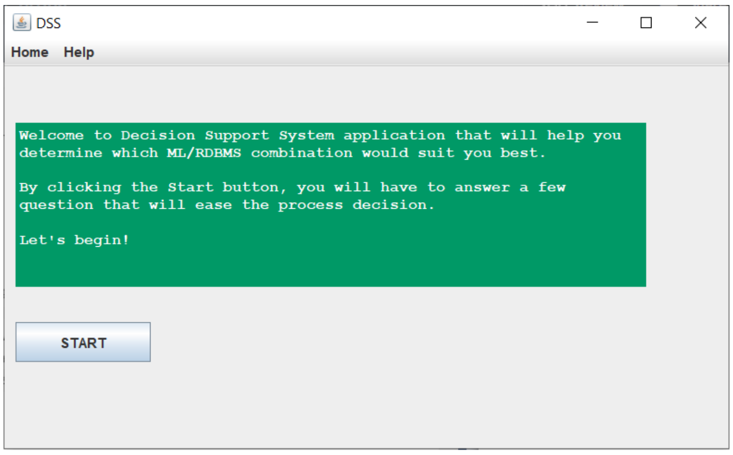
# Decision Support System

DSS is an interactive system designed to support different decision, define courses of actions or make a judgment, it can be used in all levels of an organization. The idea behind using a DSS in this work was to help users who are uncertain which solution for ML project involving predictive algorithms to use. The final creation is a simple java GUI application created in Eclipse IDE 2021-06. The application's logic is based on seven questions that are answered by the user, and taking into consideration different conditions with a simple if else logic behind, it produces most appropriate ML/RDBMS combination of tools for the user out of the four analyzed.

## 6.1 DSS flow

Created DSS has the following flow, users are presented with a main dialogue, where by clicking the Start button questionnaire made out of seven questions is opened.

An addition to each window is a menu bar with two submenues. One is **Home** that offers the user to retake the questionnaire from scratch. Another is **Help** that has two additional options, one is a **Welcome** screen where the short introduction to what is the application for is again presented, and the other is **About** that leads to github page of the creator where additional



Welcome to Decision Support System application that will help you determine which ML/RDBMS combination would suit you best.

By clicking the Start button, you will have to answer a few question that will ease the process decision.

Let's begin!

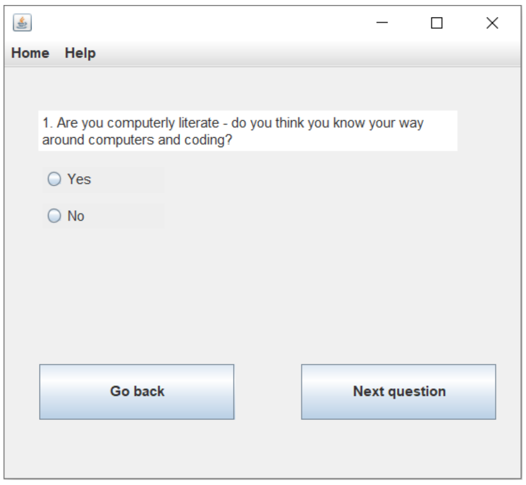
**Figure 6.1:** Decision support system main screen.

information about the whole thesis is shown, along with source code of the application itself.

Each question has a dialogue made with approximately the same java elements, as seen in 6.2. The two buttons are used to move through the questions. Button **Go back** returns us to the previous screen / question. It has implemented a java Confirm dialog with a JOptionPane yes and no option with the question Are you sure you want to go back (Yes returns user to previous screen, No keeps user on the current question). The **Next question** button moves the user onto the next question. It also preforms a check if an answer was picked among the offered ones (if any of the radio buttons is selected, `radio_button.isSelected()`), and if one was, it does its role on moving the user forward. If none was picked it disables the user to move (with a warning message dialog - You have to pick one of the given answers!).

The logic behind the asked questions is predisposed on conclusions made with the ML/RDBMS tools combination analysis, given questions and offered answers are:

1. `Question:` What operating system are you using on your computer?
   `Answers:` Microsoft Windows; MacOS; Debian Ubuntu; Fedora Red Hat; Other Linux system


1. Are you computerly literate - do you think you know your way around computers and coding?

Yes

No

**Figure 6.2:** Example of the First question dialogue in created DSS application.

2. `Question:` Have you ever coded in Python?
   `Answers:` Yes; No
3. `Question:` How much free space for different needed installations do you have on your computer?
   `Answers:` Less than 1GB; Between 1GB and 10GB; More than 10GB; I don't know
4. `Question:` What kind of predictions will you be needing?
   `Answers:` Predictions of class labels; Predictions of a continuous quantity
5. `Question:` What kind of data does your dataset contain?
   `Answers:` Mostly numerical; Combination of numerical and one word strings; Mostly one word records; Mostly longer strings
6. `Question:`How fast do you need your predictions?
   `Answers:` As soon as possible; I have some time
7. `Question:` Are you willing to invest some time in manual data optimization?

`Answers:` Yes; No

## 6.2 DSS logic behind decision making

Created decision support system has implemented a cross work of many if else conditions in the tree form to come to a decision. Each question's answer leads down a specific path to a given recommendation. Known features of each system that were considered, besides the accuracy and training time, were the disk space consumption, operating system support, user's previous experience with Python, as seen from the presented questions 6.1. Disk space consumption - hardware requirements:

- MindsDB takes around 3GB with its entire requirements [78] including PyTorch
- PostgreSQL requires around 512MB
- MariaDB needs around 400MB
- Python and SQLite (SQLite is included in the standard Python library since version 2.5) take up around 30MB
- Oracle needs around 10GB

Operating system support:

- MindsDB - Linux, Windows, macOS
- PostgreSQL - Linux, macOS, Windows, BSD, Solaris (only built from source)
- MariaDB - Linux, Windows
- Python - Linux, Windows, FreeBSD, macOS
- Oracle - Oracle Linux, Red Hat Enterprise Linux, SUSE Linux Enterprise, Oracle Solaris, Windows

Why each question was put together is defined in the next list.

1. Not all of the database solutions are compatible with all operating systems, on some they have no implementation solution, as the operating system compatibility explained above. We had only taken the most popular and used operating system, possible improvement could be to take into consideration also the marginal cases of users with less mainstream system usage.

2. To use SQLite solution along with Python, a person needs to understand their dataset in detail, be able to use Python to analyze, clean and prepare their data, along with preform different kinds of transformation and use ML algorithms by themselves. It is useful to know if they have ever used Python, if they have not it might be easier to use a different option.

3. All system's installation take up different amounts of memory space. Oracle installation, for example, takes up a lot, hence it is pertinent to be informed of available space on user's computers.

4. Two solutions, MindsDB ones, are more suitable for successful predictions of only classification problem settings, while for the regression ones, MindsDB is missing appropriate evaluation metrics.

5. Some solutions, again MindsDB ones, handle larger quantities of string data worse, hence it is pertinent to know data types contained in their dataset. Even though Python is here considered to handle data with longer strings good, as the user can do their own optimization accordingly.

6. Not all solution take same amount of time to set up the system and data or to give predictions, hence if the answer is that the results are needed as soon as possible, the quicker combination will be returned for the user.

7. This is an extra question shown if the user has answered yes to ever coding in Python, because even if they are acquainted with Python programming language they might not want to manually do optimization processes themselves. It is put at the end, because other questions might lead the user to another solution, or this one will help reach a better decision.

An Answer array is implemented as a global variable, answers are added or removed from this array depending on the action performed by the user. The **Go back** button from the question window preforms a check if the answer from the previous screen was added in the answers array, if so it removes it so that duplicates do not appear, as well as so the user is given an option to change their answer, while the **Next button** adds the current answer into the answer array. Each question's answer is given numbers from 0 to number of offered answers per that question minus 1. The if else logic goes through this array to the end, and moves down the tree depending on the answer.

# Chapter 7

# Conclusion

The goal of comparing different toolsets and detecting some minor variations was reached. To be seen from results is that all combinations had equal outcome in some cases. But some conclusions can be extracted from it all. Like that MariaDB has the worse approach towards imbalanced sets, as the less represented label was not predicted in both cases. But this should be tested on more examples to conclude it concretely, as we only had two examples with imbalanced datasets of small quantities. It was also deemed slowest in our performed analysis, while fastest was Python. Even though the speed with Python follows up that its algorithms are ran strictly on integer data, while in all other combinations, datasets were included mostly "as is", with larger strings. Hence, training times received here can only be viewed subjectively as they heavily depend on type of data contained in the dataset, but also depend on system's abilities where the training was performed. So even though we included in the DSS our own timing conclusions, we cannot be completely certain MindsDB would not have performed faster analysis on an another machine.

Our experiment on the In-vehicle coupon recommendation dataset if visiting frequency of the establishment has any effect on accepting coupon has proven void, as only for "Bar" establishment the overall accuracy was higher than predictions over "All data", and only Oracle had "All data" accuracy lower than other establishment's accuracies, except for "Carry out and Take away". As well as for the experiment in Developer survey dataset, the differences were minimal between the two taken approaches, the dropped missing values columns did not affect the final result significantly besides in Python.

Main conclusion is MindsDB's inability to process heavier, i.e. larger datasets. It showed worst performance here. Further research should be done with more datasets gradually increasing the column string values lengths, to test whether our assumption was correct, as we have had only one example of a heavier dataset. As well as to test where is the border for making a successful prediction. Despite its shortcomings, it offers great insight into our data before training. It graphically determines the quality of our data - the importance of each column in the final predictions, percentage of null values in the columns, type of data contained in the column, how many distinct values are contained and the probability of potential bias (it will mark that a certain column contains too many records of the same value despite there being potential few distinct ones). So it can greatly help in other parts of analysis even before offering predictions.

Overall, we can say Oracle had produced the best results, as it was fast and had mainly similar accuracies as others. Its main drawback is it requires huge amount of disk space when compared to others, but also that the free express edition limits the user to only 12GB of data, what can be troublesome if we have large datasets. Some points, as mentioned would require further analysis on other datasets.

# Appendix A

# List of columns in datasets

**Stroke:** Id, Gender, Age, Hypertension, HeartDisease, Ever-Married, Work-Type, ResidenceType, AvgGlucoseLevel, Bmi, SmokingSta-tus, Stroke.

**Hotel satisfaction:** Id, Gender, Age, PurposeOfTravel, TypeOfTravel, TypeOfBooking, HotelWifiService, Departure/arrivalConvenience, EaseOfOnlineBooking, HotelLocation, FoodAndDrink, StayComfort, CommonRoomEntertainment, Check-in/Check-outServices, OtherService, Cleanliness, Satisfaction.

**Hotel booking:** IsCanceled, LeadTime, ArrivalDateYear, ArrivalDateMonth, ArrivalDateWeekNumber, ArrivalDateDayOfMonth, StaysInWeekendNights, StaysInWeekNights, Adults, Children, Babies, Meal, Country, MarketSegment, DistributionChannel, IsRepeatedGuest, PreviousCancellations, PreviousBookingsNotCanceled, ReservedRoomType, AssignedRoomType, BookingChanges, DepositType, Agent, Company, DaysInWaitingList, CustomerType, Adr, RequiredCarParkingSpaces, TotalOfSpecialRequests, ReservationStatus, ReservationStatusDate.

**In-vehicle coupon recommendation:** Destination, Passanger, Weather, Temperature, Time, Coupon, Expiration, Gender, Age, MaritalStatus, HasChildren, Education, Occupation, Income, Car, Bar, CoffeeHouse, CarryAway, RestaurantLessThan20, Restaurant20To50, ToCouponGEQ5min, ToCouponGEQ15min, ToCouponGEQ25min, DirectionSame, DirectionOpp, Y.

**California housing:** MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude, MedHouseVal.

**Developer survey:** Respondent, MainBranch, Hobbyist, Age, Age1stCode, CompFreq, CompTotal, ConvertedComp, Country, CurrencyDesc, CurrencySymbol, DatabaseDesireNextYear, DatabaseWorkedWith, DevType, EdLevel, Employment, Ethnicity, Gender, JobFactors, JobSat, JobSeek, LanguageDesireNextYear, LanguageWorkedWith, MiscTechDesireNextYear, MiscTechWorkedWith, NEWCollabToolsDesireNextYear, NEWCollabToolsWorkedWith, NEWDevOps, NEWDevOpsImpt, NEWEdImpt, NEWJobHunt, NEWJobHuntResearch, NEWLearn, NEWOffTopic, NEWOnboardGood, NEWOtherComms, NEWOvertime, NEWPurchaseResearch, NEWPurpleLink, NEWSOSites, NEWStuck, OpSys, OrgSize, PlatformDesireNextYear, PlatformWorkedWith, PurchaseWhat, Sexuality, SOAccount, SOComm, SOPartFreq, SOVisitFreq, SurveyEase, SurveyLength, Trans, UndergradMajor, WebframeDesireNextYear, WebframeWorkedWith, WelcomeChange, WorkWeekHrs, YearsCode, YearsCodePro.

# Bibliography

[1] licence-cc.pdf.
URL https://ucilnica.fri.uni-lj.si/course/view.php?id=274

[2] C. Ré, D. Agrawal, M. Balazinska, M. Cafarella, M. Jordan, T. Kraska, R. Ramakrishnan, Machine Learning and Databases: The Sound of Things to Come or a Cacophony of Hype?, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, 2015, p. 283–284. doi:10.1145/2723372.2742911.

[3] R. V. Zicari, On in-database machine learning. Interview with Waqas Dhillon., http://www.odbms.org/blog/2018/11/on-in-database-machine-learning-interview-with-waqas-dhillon/, accessed: 2020-12-03 (2018).

[4] A. Agrawal, R. Chatterjee, C. Curino, A. Floratou, N. Godwal, M. Interlandi, A. Jindal, K. Karanasos, S. Krishnan, B. Kroth, J. Leeka, K. Park, H. Patel, O. Poppe, F. Psallidas, R. Ramakrishnan, A. Roy, K. Saur, R. Sen, M. Weimer, T. Wright, Y. Zhu, Cloudy with High Chance of DBMS: a 10-year Prediction for Enterprise-Grade ML, arXiv preprint arXiv/1909.00084 (2019).

[5] Scikit-learn website, https://scikit-learn.org/ (2020).

[6] MindsDB website, https://mindsdb.com/ (2020).

[7] Oracle Machine Learning - Technical brief, `https://www.oracle.com/a/tech/docs/technical-resources/oml-technical-brief.pdf`, accessed: 2020-11-28.

[8] A. Jalal Zadeh Fard, A. Le, G. Larionov, W. Dhillon, C. Bear, Vertica-ML: Distributed Machine Learning in Vertica Database, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD '20, 2020, pp. 755–768. `doi:10.1145/3318464.3386137`.

[9] Decision Support Systems – Introduction, Categorization and Development, `https://www.managementstudyguide.com/decision-support-systems.htm`, accessed: 2020-12-03.

[10] Vertica website, `https://www.vertica.com/`, accessed: 2020-12-03.

[11] M. Schüle, F. Simonis, T. Heyenbrock, A. Kemper, S. Günnemann, T. Neumann, In-Database Machine Learning: Gradient Descent and Tensor Algebra for Main Memory Database Systems, in: Proceeding of the 18th symposium of Database Systems for Business, Technology and Web, BTW 2019, 2019, pp. 247–266. `doi:10.18420/btw2019-16`.

[12] L. Passing, M. Then, N. Hubig, H. Lang, M. Schreier, S. Günnemann, A. Kemper, T. Neumann, SQL- and Operator-centric Data Analytics in Relational Main-Memory Databases, in: Proceedings of the 20th International Conference on Extending Database Technology, EDBT '17, 2017, pp. 84–95.

[13] MindsDB - Comparisons with other libraries, `https://docs.mindsdb.com/comparisons/ComparsionWithOtherLibraries/`, accessed: 2020-11-28.

[14] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, A. Kumar, The MADlib

Analytics Library or MAD Skills, the SQL, in: Proc. of the 38th International Conference on Very Large Data Bases, Vol. 5 of VLDB '12, 2012, p. 1700–1711. `doi:10.14778/2367502.2367510`.

[15] Apache Mahout website, `http://mahout.apache.org/` (2020).

[16] A. Pavlo, E. Paulson, A. Rasin, D. Abadi, D. DeWitt, S. Madden, M. Stonebraker, A Comparison of Approaches to Large-Scale Data Analysis, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, SIGMOD '09, 2009, pp. 165–178. `doi:10.1145/1559845.1559865`.

[17] K. Karanasos, M. Interlandi, D. Xin, F. Psallidas, R. Sen, K. Park, I. Popivanov, S. Nakandal, S. Krishnan, M. Weimer, Y. Yu, R. Ramakrishnan, C. Curino, Extending Relational Query Processing with ML Inference, in: 10th Annual Conference on Innovative Data Systems Research, CIDR 2020, 2020.

[18] ONNX Runtime website, `https://www.onnxruntime.ai/` (2020).

[19] SimSQL Overview, `https://cmj4.web.rice.edu/SimSQL/SimSQL.html`, accessed: 2020-12-03.

[20] D. Jankov, S. Luo, B. Yuan, Z. Cai, J. Zou, C. Jermaine, Z. Gao, Declarative Recursive Computation on an RDBMS, or, Why You Should Use a Database For Distributed Machine Learning, in: Proceedings of the 45th International Conference on Very Large Data Bases, Vol. 12 of VLDB '19, 2019, pp. 822–835. `doi:10.14778/3317315.3317323`.

[21] Sqlite library, `https://docs.python.org/3/library/sqlite3.html`, accessed: 2021-18-08.

[22] S. Gulati, The 5 minute introduction to duckdb: The sqlite for analytics, `https://shekhargulati.com/2019/12/15/the-5-minute-introduction-to-duckdb-the-sqlite-for-analytics/`, accessed: 2021-18-08 (2019).

[23] Pandas library, https://pandas.pydata.org/, accessed: 2021-18-08.

[24] Scikit-learn, https://scikit-learn.org/stable/, accessed: 2021-18-08.

[25] MindsDB product, https://mindsdb.com/product/ (2020).

[26] MindsDB, Machine learning capabilities come to the majority of open source databases with mindsdb ai-tables, https://mindsdb.com/newsroom/machine-learning-capabilities-come-to-the-majority-of-open-source-databases-wit accessed: 2021-18-08 (October 2020).

[27] MindsDB AI tables, https://mindsdb.com/aitables/ (2020).

[28] What is MariaDB?, https://hub.docker.com/_/mariadb (2020).

[29] MariaDB, Why mariadb? advantages over mysql, https://mariadb.com/resources/blog/why-should-you-migrate-from-mysql-to-mariadb/, accessed: 2021-18-08 (October 2014).

[30] R. Peterson, MariaDB vs MySQL: What is the Difference Between MariaDB and MySQL, https://www.guru99.com/mariadb-vs-mysql.html, accessed: 2021-01-10 (October 2021).

[31] PostgreSQL website, https://www.postgresql.org/ (2020).

[32] PostgreSQL About, https://www.postgresql.org/about/ (2020).

[33] Advantages of postgresql, https://www.cybertec-postgresql.com/en/postgresql-overview/advantages-of-postgresql/, accessed: 2021-18-08.

[34] What is oracle?, https://www.educba.com/what-is-oracle/, accessed: 2021-18-08.

[35] Build Machine Learning Solutions with Oracle's Services and Tools, `https://www.oracle.com/a/ocom/docs/build-machine-learning-solutions-cloud-essentials.pdf`, accessed: 2021-08-28.

[36] Oracle SQLDeveloper, `https://www.oracle.com/database/technologies/appdev/sqldeveloper-landing.html`, accessed: 2021-08-28.

[37] Oracle Data Miner, `https://www.oracle.com/database/technologies/datawarehouse-bigdata/dataminer.html`, accessed: 2021-08-28.

[38] J. Brownlee, 14 different types of learning in machine learning, `https://machinelearningmastery.com/types-of-learning-in-machine-learning/`, accessed: 2021-18-08 (2019).

[39] O. Tsymbal, 5 essential machine learning techniques for business applications, `https://mobidev.biz/blog/5-essential-machine-learning-techniques`, accessed: 2021-18-08 (2020).

[40] Stroke prediction dataset, `https://www.kaggle.com/fedesoriano/stroke-prediction-dataset`, accessed: 2021-18-08.

[41] Hotel satisfaction dataset, `https://www.kaggle.com/ishansingh88/europe-hotel-satisfaction-score`, accessed: 2021-18-08.

[42] N. Antonio, A. de Almeida, L. Nunes, Hotel booking demand datasets, Data in Brief 22 (2019) 41–49. `doi:https://doi.org/10.1016/j.dib.2018.11.126`.
URL `https://www.sciencedirect.com/science/article/pii/S2352340918315191`

[43] In-vehicle coupon recommendation dataset, https://archive.ics.uci.edu/ml/datasets/in-vehicle+coupon+recommendation#, accessed: 2021-18-08.

[44] California housing dataset, https://scikit-learn.org/stable/datasets/real_world.html#california-housing-dataset, accessed: 2021-18-08.

[45] StackOverflow Developer Survey, https://insights.stackoverflow.com/survey, accessed: 2021-18-08.

[46] Venv module, https://docs.python.org/3/library/venv.html, accessed: 2021-08-28.

[47] The Jupyter Notebook, https://jupyter.org/, accessed: 2021-08-28.

[48] O. O. D. Science, Why you should be using jupyter notebooks, https://medium.com/@ODSC/why-you-should-be-using-jupyter-notebooks-ea2e568c59f2, accessed: 2021-18-08 (July 2020).

[49] Library numpy, https://numpy.org/, accessed: 2021-08-28.

[50] Library matplotlib, https://matplotlib.org/, accessed: 2021-08-28.

[51] Tool yellowbrick, https://www.scikit-yb.org/en/latest/, accessed: 2021-08-28.

[52] Library seaborn, https://seaborn.pydata.org/#:~:text=Seaborn%20is%20a%20Python%20data,introductory%20notes%20or%20the%20paper., accessed: 2021-08-28.

[53] Pandas structure dataframe, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html, accessed: 2021-08-28.

[54] A. Kumar, Python – replace missing values with mean, median & mode, https://vitalflux.com/

pandas-impute-missing-values-mean-median-mode/, accessed: 2021-18-08 (July 2021).

[55] C. Moffitt, Guide to encoding categorical values in python, https://pbpython.com/categorical-encoding.html, accessed: 2021-18-08 (February 2017).

[56] What is correlation, https://www.jmp.com/en_ch/statistics-knowledge-portal/what-is-correlation.html, accessed: 2021-08-28.

[57] What is Logistic Regression?, https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/what-is-logistic-regression/#:~:text=Logistic%20regression%20is%20the%20appropriate,variable%20is%20dichotomous%20(binary).&text=Logistic%20regression%20is%20used%20to,or%20ratio%2Dlevel%20independent%20variables., accessed: 2021-08-28.

[58] Gaussian Naive Bayes, https://iq.opengenus.org/gaussian-naive-bayes/, accessed: 2021-08-28.

[59] R. Gandhi, Support vector machine — introduction to machine learning algorithms, https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a44
accessed: 2021-18-08 (June 2018).

[60] T. Yiu, Understanding random forest, https://towardsdatascience.com/understanding-random-forest-58381e0602d2, accessed: 2021-18-08 (June 2019).

[61] V. Aliyev, Gradient boosting classification explained through python, https://towardsdatascience.com/gradient-boosting-classification-explained-through-python-60cc980eeb3d, accessed: 2021-18-08 (September 2020).

[62] W. Koehrsen, Introduction to bayesian linear regression - an explanation of the bayesian approach to linear modeling, `https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7`, accessed: 2021-18-08 (April 2018).

[63] What is Elastic Net?, `https://corporatefinanceinstitute.com/resources/knowledge/other/elastic-net/`, accessed: 2021-08-28.

[64] C. Vermandere, Impact of salary on job satisfaction, `https://www.eurofound.europa.eu/publications/article/2013/impact-of-salary-on-job-satisfaction`, accessed: 2021-28-08 (May 2013).

[65] MindsDB installation on Linux, `https://docs.mindsdb.com/deployment/linux/`, accessed: 2021-10-17.

[66] Connect storage engine, `https://mariadb.com/kb/en/connect/`, accessed: 2021-08-28.

[67] Configuration file for connecting MindsDB and MariaDB, `https://docs.mindsdb.com/datasources/configuration/#mariadb-configuration`, accessed: 2021-08-28.

[68] Linux downloads (Ubuntu), `https://www.postgresql.org/download/linux/ubuntu/`, accessed: 2021-08-28.

[69] PgAdmin4 website, `https://www.pgadmin.org/`, accessed: 2021-08-28.

[70] PgAdmin4 download website, `https://www.pgadmin.org/download/pgadmin-4-apt/`, accessed: 2021-08-28.

[71] RPM repository, `https://rpmfind.net/linux/rpm2html/`, accessed: 2021-09-13.

[72] Location of RPM for Oracle database, `https://www.oracle.com/database/technologies/xe-downloads.html`, accessed: 2021-09-13.

[73] Setting Up Oracle Data Miner 4.1 tutorial, `https://www.oracle.com/webfolder/technetwork/tutorials/obe/db/12c/r1/dm/dm_41/ODM12c-41_SetUp.html#section1`, accessed: 2021-09-13.

[74] Oracle data mining concepts, `https://docs.oracle.com/database/121/DMCON/toc.htm`, accessed: 2021-10-02.

[75] Thesis documentation and some code examples, `https://github.com/vanda1345/master_fri` (2021).

[76] A. Mishra, Metrics to Evaluate your Machine Learning Algorithm, `https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234`, accessed: 2021-28-08 (Feb 2018).

[77] S. Narkhede, Understanding AUC - ROC Curve, `https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5`, accessed: 2021-28-08 (June 2018).

[78] Requirements text file for MindsDB, `https://github.com/mindsdb/mindsdb/blob/stable/requirements.txt`, accessed: 2021-09-13.