Assignment 3 – Raw Data Processing, ETL and Data Mining

Matthew Lindstrom, Spencer Reid, Justin Savenko
https://github.com/mlind423/Database-2_Assignment_3

Extraction Section:

MariaDB

MariaDB has a couple of different ways of extracting data from the database into CSV. The most popular way, and the easiest to find within the MariaDB documentation, is "SELECT INTO OUTFILE." You would execute this method similarly to how a query is executed through the command line of MariaDB. Through this method, you would have to appropriately select the columns from which you would be interested in gathering data, followed by "INTO OUTFILE" with the desired path for the file. This method can terminate fields with the required comma and terminate lines with the necessary field to match the CSV format. The following is an example of the command format to execute to output a table in the CSV format of the Employee Information Table.

```
SELECT Employee_ID, DOB, DOJ, Department_ID

INTO OUTFILE 'Employee_Information.csv'

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

FROM Employee_Information;
```

The second popular way to use the command line structure is through the client itself. This involves executing the desired queries to fetch the data and then using output redirection to a file within the command.

```
mysql -u your_username -p -e "SELECT Employee_ID, DOB, DOJ, Department_ID
FROM Employee_Information" MRU > Employee_Information.csv
```

-p will prompt the user to enter a password for the database, and -e will execute the following command. The second line redirects file output from the screen/command line to a file.

The final method is very similar to the second method but can be done through the mysqldump command that MariaDB offers. Due to the way, the data is dumped, this requires piping the output from the mysqldump into a SED to parse it into the required CSV format but would result in a CSV output. The following can be executed to achieve this result

```
Mysqldump -u your_username -p MRU Employee_Information

| sed 's/\t/","/g;s/\n/\\n/g' > Employee_Information.csv
```

Note: assumptions made in the commands is that the table is Employee_Information and that there may be additional fields in the table not outputted and that database name is MRU

MongoDB

Like MariaDB, MongoDB has several different ways of extracting data from the database into CSV. The most popular method or recommended method from the MongoDB documentation is the "mongoexport" command line tool that already comes with MongoDB. This tool allows you to export the data from your MongoDB to JSON, CSV, or any other format supported. The basic command line structure to achieve this would look like the following

```
mongoexport --host MRU --db MRU --collection Employee_Information --type=csv --
out Employee_Information.csv --fields Employee_ID,DOB,DOJ,Department_ID
```

Suppose you are looking for a more graphical way of extracting the data from MongoDB. In that case, MongoDB does provide a GUI called MongoDB Compass that offers various tools for working with MongoDB databases. One of these tools allows you to export query results to CSV format. To use this method, first, you need to open MongoDB Compass and connect to your MongoDB server itself, navigate to the appropriate collection from which you wish to export data and execute any required queries to filter the data as needed. Once done, an "Export" button is labelled, and select "Export Collection." These may be represented by small downward arrows depending on the version and selected CSV as the export option.

For those who are more comfortable with Python, you can also export data from MongoDB using the Pandas library. This method, which we've been using throughout the semester, allows you to export to a CSV file. While it may be considered the most involved method, it's a familiar tool for many data analysts. Here's the Python code to achieve this:

```
import pandas as pd

from pymongo import MongoClient

client = MongoClient('mongodb://MRU:27017/')

db = client['MRU']

collection = db['Employee_Information']

cursor = collection.find({}, {'Employee_ID': 1, 'DOB': 1, 'DOJ': 1, 'Department_ID': 1,
'_id': 0})

df = pd.DataFrame(list(cursor))

df.to_csv('Employee_Information.csv', index=False)
```

Assignment 3 – Raw Data Processing, ETL and Data Mining

Matthew Lindstrom, Spencer Reid, Justin Savenko

https://github.com/mlind423/Database-2_Assignment_3

Transformation:

The transformation was relatively straightforward for the most part. For most fields, you can simply check that they are all filled in and check for any out-of-range values. This will directly solve most issues, especially for our purposes in regression.

Trouble arises in searching for uniqueness among tables, especially the larger ones, such as the performance table. We used a concatenation of any relevant PK elements and fed that into a dictionary, as searching an array is far too slow to be used in this manner; it also allowed me to keep track of the indexes (at least in the CSV file) for future reference concerning the DB admin cleaning their actual database. FK checks were made but found nothing of concern, so the DBMS is doing its job of retaining consistency there.

Regarding Completeness, we pinged and deleted those lines with missing values. However, the data may be missing a few performances. With our dataset, our sample should still be reflective, but this blind area is worth noting.

For Validity, we checked implausible or impossible values for Effort and Results and deleted them for our analysis. It was presumed that Dates were stored in MM/DD/YYYY format, but there is a chance that most of the database is wrong. Regardless, this only brought up a few employee birthdates as incorrect.

In terms of consistency, no FK deletion errors were found. Good Job.

For uniqueness, there were surprisingly no duplicate assignments for any of the students. However, there are several duplicate entries for the other tables, at least with primary keys, which was what was checked on. Each duplicate pings to the previously found entry of that data using Python Dictionaries. There was no triplicate information in this DB, but in a different database, one would have to check for transient duplicate references when viewing the reports.

Each of the Report CSV files summarizes this, and the index refers to the position of each item in the CSV extracts given to us.

Data was cleaned up manually using the references found from the data cleansing program, and fed into our regression as a new file (Updated_Performance.csv).

Regression Predictive Model:

The provided extracted data underwent a preliminary examination, including visualizations to discern the distribution of marks and assess the relationship between effort hours and performance. Notably, the distribution of marks appeared non-normal, suggesting potential complexities in further analysis. However, analyzing the data through a scatter plot with effort hours versus marks revealed a visible trend, though with apparent anomalies such as effort hours predominantly recorded as whole numbers with no partial hours.

Specific student performance data is isolated for a further focused analysis, facilitating a closer examination of the correlation between effort hours and marks. This more targeted approach revealed a positive correlation between the two variables, highlighting the potential significance of effort investment in academic performance. Data preparation further involved partitioning the dataset into the training and testing subset to help ensure an unbiased evaluation of the predictive model's efficacy.

We used the linear regression algorithm from the sci-kit-learn library for the modelling. This choice was made due to its simplicity and interpretability, factors that should instill confidence in its use. Through a series of computations, including intercept and coefficient determination, the model is trained on the training dataset to predict student marks based on their reported effort hours. The models' predictive capabilities were validated using the testing datasets, with metrics such as mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE) providing a quantifiable assessment of predictive accuracy.

After training the linear regression model on the training dataset, we evaluated its performance using the testing dataset. The MAE, which measures the average absolute difference between predicted and actual values, was 2.39 units. Similarly, the MSE, which quantifies the average squared difference between predicted and actual values, was calculated to be 9.36 units. Additionally, the RMSE, representing the average magnitude of the errors in the same units as the response variable, was determined to be 3.06 units. These metrics collectively provide insights into the accuracy and performance of our regression model, with lower values indicating better predictive performance. Considering the grading scale of 0 to 100, these metrics are relatively accurate, with deviations of only a few points from the actual performance that will be presented from predictions.

According to our predictive regression model, the predicted score for a student ID of SID20131151 with ten effort hours will be 82.091761. A student with a student ID of SID20149500 predicted a score with ten effort hours would be 81.97564057. Students with a student ID of SID20182516 and ten effort hours will be 82.22862079. Overall, these results line up with what we saw with our analysis of the scatter plot we made at the beginning. Given the accuracy of our predictive regression model and taking the RMSE value, all these predicted scores can be plus or minus 3.06 points from the actual score if said students put in 10 effort hours.