Assignment 2. Data cleaning, schema matching, and data matching

COMPSCI 767

Juan David Roa Valencia

Data Loading and Preliminary Analysis

```
import pandas as pd
import matplotlib.pyplot as plt

def load_and_analyze_data(path):
    df = pd.read_csv(path)

    missing_values_percentage = df.isnull().mean() * 100
    print(f"Percentage of missing values in {path.split("/")[-1]}:")
    print(missing_values_percentage)

    data_types = df.dtypes
    print(f"\nData types of the columns in {path.split("/")[-1]}:")
    print(data_types)

    return df

df_tableA = load_and_analyze_data("tableA/tableA.csv")
df_tableB = load_and_analyze_data("tableB/tableB.csv")
```

Output:

```
Percentage of missing values in tableA.csv:
ID          0.0
title       0.0
authors     0.0
abstract    0.0
url         0.0
date        0.0
dtype: float64

Data types of the columns in tableA.csv:
ID           int64
title       object
authors     object
abstract    object
url         object
date        object
```

```
dtype: object
Percentage of missing values in tableB.csv:
ID           0.0
title        0.0
authors      2.1
abstract    10.1
url          0.0
date         0.0
dtype: float64

Data types of the columns in tableB.csv:
ID           int64
title       object
authors     object
abstract    object
url         object
date        object
dtype: object
```

{: .box-note} **Note:** The csv files were loaded into `pandas dataframes`.

```
missing_values_percentage = df.isnull().mean() * 100
print(f"Percentage of missing values in {path.split("/")[-1]}:")
print(missing_values_percentage)

data_types = df.dtypes
print(f"\nData types of the columns in {path.split("/")[-1]}:")
print(data_types)
```

{: .box-note} **Note:** For columns of type `"object"` (strings), the average, minimum, and maximum lengths were calculated.

```
for column in df.select_dtypes(include="object").columns:
    average_length = df[column].str.len().mean()
    minimal_length = df[column].str.len().min()
    maximal_length = df[column].str.len().max()

    print(f"\nFor column "{column}" in {df_name}:")
    print(f"Average length: {average_length:.3f}")
    print(f"Minimal length: {minimal_length}")
    print(f"Maximal length: {maximal_length}")
```

Output:

```
For column "title" in tableA:
Average length: 78.259
Minimal length: 14
```

```
Maximal length: 194

For column "authors" in tableA:
Average length: 63.550
Minimal length: 7
Maximal length: 463

For column "abstract" in tableA:
Average length: 1346.466
Minimal length: 412
Maximal length: 2232

For column "url" in tableA:
Average length: 49.000
Minimal length: 49
Maximal length: 49

For column "date" in tableA:
Average length: 10.831
Minimal length: 8
Maximal length: 14

For column "title" in tableB:
Average length: 85.043
Minimal length: 9
Maximal length: 249

For column "authors" in tableB:
Average length: 60.471
Minimal length: 8.0
Maximal length: 288.0

For column "abstract" in tableB:
Average length: 201.875
Minimal length: 11.0
Maximal length: 215.0

For column "url" in tableB:
Average length: 96.858
Minimal length: 83
Maximal length: 120

For column "date" in tableB:
Average length: 13.682
Minimal length: 9
Maximal length: 17
```

Data Visualization and Outlier Detection

{: .box-note} **Note:** Histograms were created for the `"title"` and `"abstract"` columns of both dataframes to understand the distribution of lengths and identify potential outliers.

```
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.hist(df["title"].str.len(), bins=20, color="skyblue",
edgecolor="black")
plt.title(f"Histogram of Title Lengths in {df_name}")
plt.xlabel("Length")
plt.ylabel("Frequency")

plt.subplot(1, 2, 2)
plt.hist(df["abstract"].str.len(), bins=20, color="skyblue",
edgecolor="black")
plt.title(f"Histogram of Abstract Lengths in {df_name}")
plt.xlabel("Length")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```

![Histograms]({{ "/assets/img/Figure_1.png" | relative_url }})

![Histograms]({{ "/assets/img/Figure_2.png" | relative_url }})

Outlier Identification

{: .box-note} **Note:** Outliers were identified in the `"ID"` column using the Interquartile Range (`IQR`) method.

```
Q1 = df["ID"].quantile(0.25)
Q3 = df["ID"].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = df[(df["ID"] < lower_bound) | (df["ID"] > upper_bound)]
print(f"Outliers in {df_name}")
print(outliers)
```

Output:

```
Outliers in tableA
Empty DataFrame
Columns: [ID, title, authors, abstract, url, date]
Index: []
Outliers in tableB
Empty DataFrame
```

```
Columns: [ID, title, authors, abstract, url, date]
Index: []
```

Comparing Length Distributions of "Title" Between Datasets

{: .box-note} **Note:** Box plots were created to compare the distributions of the lengths of the `"title"` column in both dataframes. These visualizations help to understand if there are significant differences between the datasets regarding title lengths.

```
def compare_distributions(df1, df2, column, df1_name, df2_name):
    df1_lengths = df1[column].str.len()
    df2_lengths = df2[column].str.len()

    data_to_plot = [df1_lengths.dropna(), df2_lengths.dropna()]

    fig = plt.figure(figsize=(10, 5))

    ax = fig.add_axes([0, 0, 1, 1])

    bp = ax.boxplot(data_to_plot, patch_artist=True, notch=True, vert=0)

    colors = ["#0000FF", "#00FF00"]

    for patch, color in zip(bp["boxes"], colors):
        patch.set_facecolor(color)

    for whisker in bp["whiskers"]:
        whisker.set(color="#8B008B", linewidth=1.5)

    for cap in bp["caps"]:
        cap.set(color ="#8B008B", linewidth = 2)

    for median in bp["medians"]:
        median.set(color="red", linewidth=3)

    ax.set_yticklabels([df1_name, df2_name])

    plt.title(f"Comparison of {column} Length Distribution between
{df1_name} and {df2_name}")

    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

    plt.show()

compare_distributions(df_tableA, df_tableB, "title", "tableA", "tableB")
```

Data Comparison Report

This report compares two distinct datasets: `"tableA"` and `"tableB"`. The target is to analyze and compare the distribution of the lengths of `"title"` attribute in both datasets.

Methodology

To compare the two datasets, the main focus will be on the `"title"` attribute, which is common to both datasets. Specifically, we looked at the length of the title as a characteristic feature for comparison.

A `boxplot` was used to visualize the distribution of the title lengths in each dataset. A boxplot, or box-and-whisker plot, displays a summary of the set of data values having properties like minimum, first quartile, median, third quartile, and maximum. In the boxplot, a box is created from the first quartile to the third quartile. A vertical line is also there, which goes through the box at the median. Here whiskers are drawn from the edges of the box to show the range of the data.

The plot was generated using the Python library `Matplotlib`, and the data pre-processing was done using `pandas`.

Results

The boxplot produced (as shown below) presents a comparative view of the title length distribution in both `"tableA"` and `"tableB"`.

The upper boxplot (green) represents the `"title"` length distribution in `"tableA"`, while the lower boxplot (blue) represents the `"title"` length distribution in "tableB".

- **"tableA" (Green)**: The green plot shows a large interquartile range indicating high variability in the middle half of the title lengths in `"tableA"`. There is only one outlier on the extreme right which suggests there"s only one title that is significantly longer than all the others.
- **"tableB" (Blue)**: The blue plot, on the other hand, has a short left whisker, indicating less variability in the shorter title lengths. However, many outliers on the right show that several titles with lengths are much larger than most of the title lengths in `"tableB"`.

Conclusion

The analysis and comparison of the `"title"` attribute length in both `"tableA"` and "tableB" provide valuable insights into the datasets. Despite some differences, such as the number of outliers and the variability of title lengths, both datasets show a wide range of titles, indicating diverse topics and contexts.

Further comparisons and more in-depth analyses can be performed to continue exploring the datasets based on different attributes and parameters.

Further Analysis Recommendations

Future analysis could further explore the content of the `"title"`, `"abstract"`, and `"authors"` fields. For instance, text or word frequency analysis could offer insights into the prevalent topics within the dataset. The `"authors"` field could also be analyzed to identify the most prolific contributors or institutions.

Temporal trends could also be investigated, provided the "date" field is consistently formatted. Such an analysis might reveal how title lengths or prominent topics have changed.

The "url" field offers another potential area of exploration. Parsing these URLs may uncover additional information about the data sources or hosting journals, offering further context and insights.

It is valuable to align any further analysis steps with these datasets' specific research questions and goals.

Measuring Textual Columns

To evaluate the textual data's quality, a basic analysis of the length of values in the 'title', 'authors'``, 'abstract', and 'url`` columns will be performed. This is computed by calculating the average, minimum, and maximum lengths of the values in these columns for both tables.

```
def measure_textual_columns(df, df_name, columns):
    for column in columns:
        avg_length = df[column].str.len().mean()
        min_length = df[column].str.len().min()
        max_length = df[column].str.len().max()

        print(f"For column {column} in {df_name}, average length:
{avg_length:.3f}, minimum length: {min_length}, maximum length:
{max_length}")

textual_columns = ['title', 'authors', 'abstract', 'url']
measure_textual_columns(df_tableA, 'tableA', textual_columns)
measure_textual_columns(df_tableB, 'tableB', 'tableB', textual_columns)
```

Analysis:

Table A:

```
'title' column: average length of 78.259 characters, with a minimum length
of 14 characters and a maximum length of 194 characters.
'authors' column: average length of 63.55 characters, with a minimum length
of 7 characters and a maximum length of 463 characters.
'abstract' column: average length of 1346.466 characters, with a minimum
length of 412 characters and a maximum length of 2232 characters.
'url' column: all the values have a fixed length of 49 characters.
```

Table B:

```
``'title'`` column: average length of 85.043 characters, with a minimum
length of 9 characters and a maximum length of 249 characters.
``'authors'`` column: average length of 60.471 characters, with a minimum
length of 8 characters and a maximum length of 288 characters.
``'abstract'`` column: average length of 201.875 characters, with a minimum
```

```
length of 11 characters and a maximum length of 215 characters.
``'url'`` column: average length of 96.858 characters, with a minimum length
of 83 characters and a maximum length of 120 characters.
```

- For both tables, the `'title'` and `'authors'` columns have more variability in the length of their values as compared to the `'url'` column, which has a fixed length in `tableA` and less variability in `tableB`. This suggests that the information in these columns might be more diverse and, therefore, require more sophisticated analysis and cleaning techniques.
- The `'abstract'` column in table A has significantly more content (based on the average length) than `tableB`. This`indicates that abstracts in`tableA`might provide more detailed information about the articles than those in`tableB`.

These observations help to identify the nature and quality of the textual data in the tables and can guide the further steps for data cleaning and preparation for analysis.

```
def plot_histogram(df, column, df_name):
    df[column].str.len().hist(bins=30, color='skyblue', edgecolor='black')
    plt.title(f"Histogram of {column} length in {df_name}")
    plt.xlabel('Length of Text')
    plt.ylabel('Frequency')
    plt.grid(False)
    plt.show()

plot_histogram(df_tableA, 'title', 'tableA')
plot_histogram(df_tableB, 'title', 'tableB')

plot_histogram(df_tableA, 'abstract', 'tableA')
plot_histogram(df_tableB, 'abstract', 'tableB')
```

![TextualHistograms]({{ "/assets/img/Figure_4.png" | relative_url }}) ![TextualHistograms]({{ "/assets/img/Figure_5.png" | relative_url }}) ![TextualHistograms]({{ "/assets/img/Figure_6.png" | relative_url }}) ![TextualHistograms]({{ "/assets/img/Figure_7.png" | relative_url }})

```
description_tableA_title = df_tableA['title'].str.len().describe()
print("Description of title lengths in tableA:")
print(description_tableA_title)

description_tableA_abstract = df_tableA['abstract'].str.len().describe()
print("\nDescription of abstract lengths in tableA:")
print(description_tableA_abstract)

description_tableB_title = df_tableB['title'].str.len().describe()
```

```
print("\nDescription of title lengths in tableB:")
print(description_tableB_title)

description_tableB_abstract = df_tableB['abstract'].str.len().describe()
print("\nDescription of abstract lengths in tableB:")
print(description_tableB_abstract)
```

Descriptive Statistics for Text Lengths

TableA

- **Title Lengths**
  - Count: 1000
  - Mean: 78.259
  - Standard Deviation: 24.299679
  - Minimum: 14
  - 25%: 63
  - 50%: 76
  - 75%: 92
  - Maximum: 194
- **Abstract Lengths**
  - Count: 1000
  - Mean: 1346.466
  - Standard Deviation: 328.546703
  - Minimum: 412
  - 25%: 1120.75
  - 50%: 1359
  - 75%: 1579.25
  - Maximum: 2232

TableB

- **Title Lengths**
  - Count: 1000
  - Mean: 85.043
  - Standard Deviation: 37.446353
  - Minimum: 9
  - 25%: 60
  - 50%: 82
  - 75%: 110
  - Maximum: 249
- **Abstract Lengths**
  - Count: 899

- Mean: 201.875417
- Standard Deviation: 33.587900
- Minimum: 11
- 25%: 212
- 50%: 212
- 75%: 212
- Maximum: 215

Identifying Unique and Duplicate Entries

```
unique_titles_A = df_tableA['title'].nunique()
unique_authors_A = df_tableA['authors'].nunique()
unique_abstracts_A = df_tableA['abstract'].nunique()

print(f"Unique titles in TableA: {unique_titles_A}")
print(f"Unique authors in TableA: {unique_authors_A}")
print(f"Unique abstracts in TableA: {unique_abstracts_A}")

duplicate_titles_A = df_tableA.duplicated(['title']).sum()
duplicate_authors_A = df_tableA.duplicated(['authors']).sum()
duplicate_abstracts_A = df_tableA.duplicated(['abstract']).sum()

print(f"Duplicate titles in TableA: {duplicate_titles_A}")
print(f"Duplicate authors in TableA: {duplicate_authors_A}")
print(f"Duplicate abstracts in TableA: {duplicate_abstracts_A}")

unique_titles_B = df_tableB['title'].nunique()
unique_authors_B = df_tableB['authors'].nunique()
unique_abstracts_B = df_tableB['abstract'].nunique()

print(f"Unique titles in TableB: {unique_titles_B}")
print(f"Unique authors in TableB: {unique_authors_B}")
print(f"Unique abstracts in TableB: {unique_abstracts_B}")

duplicate_titles_B = df_tableB.duplicated(['title']).sum()
duplicate_authors_B = df_tableB.duplicated(['authors']).sum()
duplicate_abstracts_B = df_tableB.duplicated(['abstract']).sum()

print(f"Duplicate titles in TableB: {duplicate_titles_B}")
print(f"Duplicate authors in TableB: {duplicate_authors_B}")
print(f"Duplicate abstracts in TableB: {duplicate_abstracts_B}")
```

Title length in TableA:

- The average length of a title is approximately 78 characters.
- The standard deviation (a measure of spread) is around 24, suggesting that the lengths of the titles vary within a relatively moderate range.
- The shortest title has 14 characters, while the longest has 194 characters.
- 50% of the titles have a length of 76 characters or less (median).

- Most titles (75%) have lengths under 92 characters.

Abstract length in TableA:

- The average length of an abstract is approximately 1346 characters.
- The standard deviation is around 328, suggesting that the lengths of the abstracts vary widely.
- The shortest abstract has 412 characters, while the longest abstract has 2232 characters.
- 50% of the abstracts have a length of 1359 characters or less (median).
- Most abstracts (75%) have lengths under 1579 characters.

This might suggest differences in how the data was collected or entered for each table. For example, there might be a strict character limit for abstracts in the source that TableB was collected from. These kinds of insights can be helpful when considering how to preprocess the data for further analysis or modeling.

For TableA:

All `'title'` and `'abstract'` entries are unique, with no duplicates. There are 6 duplicate 'authors' entries, indicating that the same author or group of authors has published more than one paper.

For TableB:

There are 86 duplicate `'title'` entries, 83 duplicate `'authors'` entries, and 150 duplicate `'abstract'` entries. This might indicate that several papers are highly similar, or even the same, in this dataset.

This can be particularly important when performing a data matching between the two tables. Cleaning the data by removing duplicates, especially in TableB should be considered. However, we must be careful because duplicates might be due to different reasons: It could be an error, but the same author or group of authors publishing in different years or journals, but the work is essentially the same.

Checking for Missing Values

```
missing_values_A = df_tableA.isnull().sum()
print("Missing values in TableA:")
print(missing_values_A)

missing_values_B = df_tableB.isnull().sum()
print("\nMissing values in TableB:")
print(missing_values_B)
```

TableA does not contain any missing values across its fields. This is a good indicator of the completeness of the data.

On the other hand, **TableB** has missing data in the `'authors'` and `'abstract'` columns. Specifically, there are:

- 21 missing values in the `'authors'` column
- 101 missing values in the `'abstract'` column

The analysis process needs to include data to ensure the results are accurate. Handling missing values properly is crucial. Various strategies for handling missing data include:

- **Deleting Rows:** This method involves removing the rows with missing values. This is typically used when the number of missing values is small. However, if the missing data is not random, this can introduce bias.
- **Filling with a specific value:** Sometimes, filling the missing values with a specific value might make sense. For instance, if we are sure that the absence of a value indicates a zero, then filling with 0 might be appropriate.
- **Filling with a central tendency measure:** For numerical columns, we might fill in missing values with a measure of central tendency, such as the mean or median. The mode (or most common value) is often used for categorical columns.
- **Data imputation:** This involves filling missing values based on other dataset information. A simple method is to fill with the average of similar items. More sophisticated methods involve using machine learning algorithms to predict missing values.

Choosing the right strategy depends on the nature of the data and the specific context.

Handling missing values is one of the critical steps in data preprocessing. Depending on the choice of handling missing data, the next steps in data cleaning involve dealing with duplicates, handling outliers, and normalizing or standardizing data.

Identifying Language of Text Data

Using the langdetect library, we can identify the predominant language of the text fields in our datasets. This library can detect over 55 languages and assigns a language code to each text it is given. It's important to note that this is a probabilistic method and may only sometimes be perfectly accurate, especially with shorter texts.

```
from langdetect import detect

def detect_language(text):
    try:
        return detect(text)
    except:
        return None

df_tableA['title_lang'] = df_tableA['title'].apply(detect_language)
df_tableA['abstract_lang'] = df_tableA['abstract'].apply(detect_language)

df_tableB['title_lang'] = df_tableB['title'].apply(detect_language)
df_tableB['abstract_lang'] = df_tableB['abstract'].apply(detect_language)

title_lang_count_tableA = df_tableA['title_lang'].value_counts()
abstract_lang_count_tableA = df_tableA['abstract_lang'].value_counts()

title_lang_count_tableB = df_tableB['title_lang'].value_counts()
abstract_lang_count_tableB = df_tableB['abstract_lang'].value_counts()
```

```
print("Language counts for 'title' in TableA:")
print(title_lang_count_tableA)
print("\nLanguage counts for 'abstract' in TableA:")
print(abstract_lang_count_tableA)

print("\nLanguage counts for 'title' in TableB:")
print(title_lang_count_tableB)
print("\nLanguage counts for 'abstract' in TableB:")
print(abstract_lang_count_tableB)
```

Results of the analysis:

TableA

Title: Predominantly in English (en) with 982 entries. There are a small number of titles in Italian (it), French (fr), Dutch (nl), Romanian (ro), Portuguese (pt), Catalan (ca), and Norwegian (no). **Abstract:** Almost exclusively in English (en) with 999 entries. There is a single abstract in French (fr).

TableB

Title: Predominantly in English (en) with 942 entries. There are also titles in Romanian (ro), Catalan (ca), Italian (it), French (fr), German (de), Danish (da), Tagalog (tl), and Dutch (nl). **Abstract:** Exclusively in English (en) with 899 entries.

From this, we can see that the text data in both tables is predominantly in English, particularly for the abstract field. This means there is no need to perform a language translation. However, a small number of entries in other languages may need to be considered in the data cleaning process. The presence of multiple languages in `'title'` might be because of the international nature of scientific publications. Language detection can also be improved, especially for shorter texts like titles.

Conclusions

Based on the exploratory data analysis, we better understand the tables `TableA` and `TableB`. Here are the key takeaways:

1. **Text Lengths**: `Title` and `abstract` lengths vary across and within the tables. `TableA` has significantly longer abstracts than `TableB`. Understanding these differences can help formulate better data cleaning and preprocessing strategies and may report our choices when constructing a data matching algorithm.

2. **Unique and Duplicate Entries**: We have identified the count of unique and duplicate entries in both tables. Notably, `TableB` has many duplicate titles, authors, and abstracts. This might indicate that several papers are highly similar or even the same in this dataset. This information will be crucial when performing data matching.

3. **Missing Values**: `TableB` contains missing values in `authors` and `abstracts`, while `TableA` has no missing data. Handling these missing values will be essential to the data preprocessing step.

4. **Language Identification**: The text data in both tables is predominantly in English, particularly in the `abstract` field. However, a small number of entries in other languages in the `title` field may need to be considered in the data cleaning process.

The above conclusions provide an understanding of the data at hand. The following steps involve cleaning and preprocessing the data considering the insights obtained. This includes dealing with missing values, removing duplicates, handling texts in other languages, and normalizing the lengths of text fields. After these steps, we can proceed with the data matching, using suitable methods for textual data, such as Jaccard similarity, Levenshtein distance, or more sophisticated methods like TF-IDF vectorization followed by cosine similarity.

Libraries and tools used on this project

- Python 3.10
- pandas
- numpy
- matplotlib
- langdetect
- datetime

The following code was used to normalize the format of the `'date'` field

```
from datetime import datetime

def parse_date(date_string):
    try:
        return datetime.strptime(date_string, '%d %B %Y')
    except ValueError:
        try:
            return datetime.strptime(date_string, '%B %Y')
        except ValueError:
            return pd.NaT

df_tableA['date'] = df_tableA['date'].apply(parse_date)
df_tableB['date'] = df_tableB['date'].apply(parse_date)
```

Blog engine and tools

- I implemented a free software (MIT license) Jekyll template for blogging which I adapted for my class projects.
- I will migrate the assignment one content to this new implementation.

- Jekyll is a static website generator built in the Ruby language. I have used it for several personal and professional projects in the past.
- This type of website is great for working on this kind of project because it has many tools and customization possibilities for content and code snippets.
- Everything is written in Markdown syntax (source code of this website can be accessed at https://github.com/roadev/datamining).