Lab Manual for Tools and Technologies for Data Science

Lab-04

Data wrangling with python

Table of Contents

1. Objective 3

2. Data Wrangling 3

3. Data Cleanup: 3

4. Why Clean Data? 3
   - 4.1 Dropping unnecessary columns in data 3
   - 4.2 Manipulating Indexes of Data frame 4
   - 4.3 Dealing with missing values 5
   - 4.4 Python Data aggregation 6

5. Groups in DataFrame 6

6. Discretization and Binning 7
   - 6.1 Detecting and Filtering Outliers 8

   Outlier Identification: There can be many reasons for the presence of outliers in the data. Sometimes the outliers may be genuine, while in other cases, they could exist because of data entry errors. It is important to understand the reasons for the outliers before cleaning them. We will start the process of finding outliers by running the summary statistics on the variables. This is done using the describe() function below, which provides a statistical summary of all the quantitative variables. 8
   - 6.2 Identifying Outliers with Interquartile Range (IQR) 8

   The interquartile range (IQR) is a measure of statistical dispersion and is calculated as the difference between the 75th and 25th percentiles. It is represented by the formula IQR = Q3 − Q1. The lines of code below calculate and print the interquartile range for each of the variables in the dataset. 8
   - 6.3 With Skewness 8

7. Identifying Outliers with Visualization 9

   Box Plot 9
   - 7.1 **Outlier Treatment** 10
     - 7.1.1 **Quantile-based Flooring and Capping** 10
     - 7.1.2 **Trimming** 10
     - 7.1.3 **Replacing Outliers with Median Values** 11

8. Evaluation Task (Unseen) [Expected time = 30mins for tasks] 11
   - 8.1 Practice Task 1 11
   - 8.2 Practice Task 2 11
   - 8.3 Practice Task 3 11
   - 8.4 Practice Task 4 11
   - 8.5 Practice Task 5 11
   - 8.6 Practice Task 6 11
   - 8.7 Practice Task 7 12
   - 8.8 Practice Task 8 12

9. Evaluation criteria 12

Lab 4: Data Wangling with python

1. Objective

- Data Wrangling
- Data Cleanup and its usage
- Unnecessary Columns dealing
- Manipulating DataFrame in Python
- Dealing with Missing Values
- Discretizing and Binning
- Aggregation and Grouping in DataFrame
- Outliers Detection
- Outliers removal

2. Data Wrangling

Data Wrangling is the process of converting data from the initial format to a format that may be better for analysis.

3. Data Cleanup:

Cleaning up your data is not the most glamourous of tasks, but it's an essential part of data wrangling. Becoming a data cleaning expert requires precision and a healthy knowledge of your area of research or study. Knowing how to properly clean and assemble your data will set you miles apart from others in your field. Python is well designed for data cleanup; it helps you build functions around patterns, eliminating repetitive work.

4. Why Clean Data?

Some data may come to you properly formatted and ready to use. If this is the case, consider yourself lucky! Most data, even if it is cleaned, has some formatting inconsistencies or readability issues (e.g., acronyms or mismatched description headers). This is especially true if you are using data from more than one dataset. It's unlikely your data will properly join and be useful unless you spend time formatting and standardizing it.
Data scientists spend a large amount of their time cleaning datasets and getting them down to a form with which they can work. In fact, a lot of data scientists argue that the initial steps of obtaining and cleaning data constitute 80% of the job.

4.1 Dropping unnecessary columns in data

- Data set we are using: books.csv – A CSV file containing information about books from the British Library

This lab assumes a basic understanding of the Pandas and NumPy libraries, including Panda's workhorse Series and DataFrame objects, common methods that can be applied to these objects, and familiarity with NumPy's NaN values.

Let's import the required modules and get started!

Pandas provides a handy way of removing unwanted columns or rows from a DataFrame with the drop() function. Let's look at a simple example where we drop a number of columns from a DataFrame.

First, let's create a DataFrame out of the CSV file 'books.csv'.

```
df = pd.read_csv("path/books.txt")
print(df.columns)
```

we can see that a handful of columns provide information that would be helpful to the library but isn't very descriptive of the books themselves: Edition Statement, Corporate Author, Corporate Contributors, Former owner, Engraver, Issuance type and Shelfmarks.
We can drop these columns in the following way:

```
to_drop = ['Edition Statement','Corporate Author','Corporate Contributors',
        'Former owner',
        'Engraver',
       'Contributors',
       'Issuance type',
        'Shelfmarks']

df.drop(to_drop, inplace=True, axis=1)
or df.drop(columns=to_drop, inplace=True)
```

we defined a list that contains the names of all the columns we want to drop. Next, we call the drop() function on our object, passing in the inplace parameter as True and the axis parameter as 1. This tells Pandas that we want the changes to be made directly in our object and that it should look for the values to be dropped in the columns of the object.

When we inspect the DataFrame again, we'll see that the unwanted columns have been removed:

4.2 Manipulating Indexes of Data frame

A Pandas Index extends the functionality of NumPy arrays to allow for more versatile slicing and labeling. In many cases, it is helpful to use a uniquely valued identifying field of the data as its index.
print(df['Identifier'].is_unique)

Let's replace the existing index with this column using set_index:
df = df.set_index('Identifier')
print(df.loc[472])

Note: You may have noticed that we reassigned the variable to the object returned by the method with df = df.set_index(...). This is because, by default, the method returns a modified copy of our object and does not make the changes directly to the object. We can avoid this by setting the inplace parameter:
df.set_index('Identifier', inplace=True)

4.3 Dealing with missing values

With every dataset it is vital to evaluate the missing values. How many are there? Is it an error? Are there too many missing values? Does a missing value have a meaning relative to its context? We can sum up the total missing values using the following:

```
# Any missing values?
print(df.isnull().values.any())

print(df['Identifier'].isnull())    //one column check
print(df.isna().sum())     //total sum of all nan
```

Now that we have identified our missing values, we have a few options. We can fill them in with a certain value (zero, mean/max/median by column, string) or drop them by row.

I. Drop null value rows

```
new = df.dropna(axis = 0, how = 'any')
print(new)
```

II. Drop duplicates

//Read new csv that have duplicate data

df1 = pd.read_csv("path/pd.csv")

```
print(df1['date'].duplicated().any())
print(df1['date'].drop_duplicates())
```

III. Fill Values

Often times you'll have to figure out how you want to handle missing values. Sometimes you'll simply want to delete those rows, other times you'll replace them.

```
# Replace missing values with a number
df1['Publisher'].fillna('Test', inplace=True)
```

More likely, you might want to do a location-based imputation. Here's how you would do that.

```
df1.loc[2,'Publisher'] = 'ITU'
print(df1['Publisher'])
```

IV. Using Median

A very common way to replace missing values is using a median.

```
#Phone data
 median = df['duration'].median()
df['duration'].fillna(median, inplace=True)
```

4.4 Python Data aggregation

```
import pandas as pd
import dateutil

df = pd.read_csv("C:/Users/Sheri/Desktop/pd.csv")


# Convert date from string to date times
        df['date'] = df['date'].apply(dateutil.parser.parse, dayfirst=True)
# How many rows the dataset
        print('How many rows the dataset: ', df['item'].count() )
# What was the longest phone call / data entry?
        print('What was the longest phone call: ', df['duration'].max() )
# How many seconds of phone calls are recorded in total?
        print('How many seconds of phone calls are recorded in total: ',
        df['duration'][df['item'] == 'call'].sum() )
# Number of non-null unique network entries
        print('Number of non-null unique network entries: ', df['network'].nunique() )
# How many entries are there for each month?
        print('How many entries are there for each month: ')
        print( df['month'].value_counts() ).
```

5. Groups in DataFrame

There's further power put into your hands by mastering the Pandas "groupby()" functionality. Groupby essentially splits the data into different groups depending on a variable of your choice. For example, the expression data.groupby('month') will split our current DataFrame by month.

The groupby() function returns a GroupBy object, but essentially describes how the rows of the original data set has been split. the GroupBy object .groups variable is a dictionary whose keys are the computed unique groups and corresponding values being the axis labels belonging to each group. For example:

```
print(df.groupby(['month']).groups.keys() )
print(len(df.groupby(['month']).groups['2014-11']) )
print(len(df.groupby(['month']).groups['2014-12']) )
```

Functions like max(), min(), mean(), first(), last() can be quickly applied to the GroupBy object to obtain summary statistics for each group – an immensely useful function

```
# Get the first entry for each month
print( df.groupby('month').first() )
# Get the sum of the durations per month
print( df.groupby('month')['duration'].sum() )
# Get the number of dates / entries in each month
print( df.groupby('month')['date'].count() )
# What is the sum of durations, for calls only, to each network
print( df[df['item'] ==
'call'].groupby('network')['duration'].sum() )
```

You can also group by more than one variable, allowing more complex queries.

```
# How many calls, sms, and data entries are in each month?
print( df.groupby(['month', 'item'])['date'].count() )
# How many calls, texts, and data are sent per month, split by
network_type?
print( df.groupby(['month', 'network_type'])['date'].count() )
```

6. Discretization and Binning

Continuous data is often discretized or otherwise separated into "bins" for analysis. Suppose you have data about a group of people in a study, and you want to group them into discrete age buckets:

ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]

Let's divide these into bins of 18 to 25, 26 to 35, 35 to 60, and finally 60 and older. To do so, you have to use cut, a function in pandas:

```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
```

You can also pass your own bin names by passing a list or array to the labels option:

```
group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
cats = pd.cut(ages,bins, labels=group_names)
print(cats)
```

6.1 Detecting and Filtering Outliers

Outlier Identification: There can be many reasons for the presence of outliers in the data. Sometimes the outliers may be genuine, while in other cases, they could exist because of data entry errors. It is important to understand the reasons for the outliers before cleaning them. We will start the process of finding outliers by running the summary statistics on the variables. This is done using the describe() function below, which provides a statistical summary of all the quantitative variables.

```
print(df.describe())
```

6.2 Identifying Outliers with Interquartile Range (IQR)

The interquartile range (IQR) is a measure of statistical dispersion and is calculated as the difference between the 75th and 25th percentiles. It is represented by the formula IQR = Q3 − Q1. The lines of code below calculate and print the interquartile range for each of the variables in the dataset.

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
```

The below code will give an output with some true and false values. The data point where we have False that means these values are valid whereas True indicates presence of an outlier.

```
print(df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))
```

6.3 With Skewness

Several machine learning algorithms make the assumption that the data follow a normal (or Gaussian) distribution. This is easy to check with the skewness value, which explains the extent to which the data is normally distributed. Ideally, the skewness value should be between -1 and +1, and any major deviation from this range indicates the presence of extreme values.

```
print(df['duration'].skew())
print(df['duration'].describe())
```

The skewness value of 16.2 shows that the variable 'duration' has a right-skewed distribution, indicating the presence of extreme higher values. The maximum 'duration' value of 1058 proves this point.

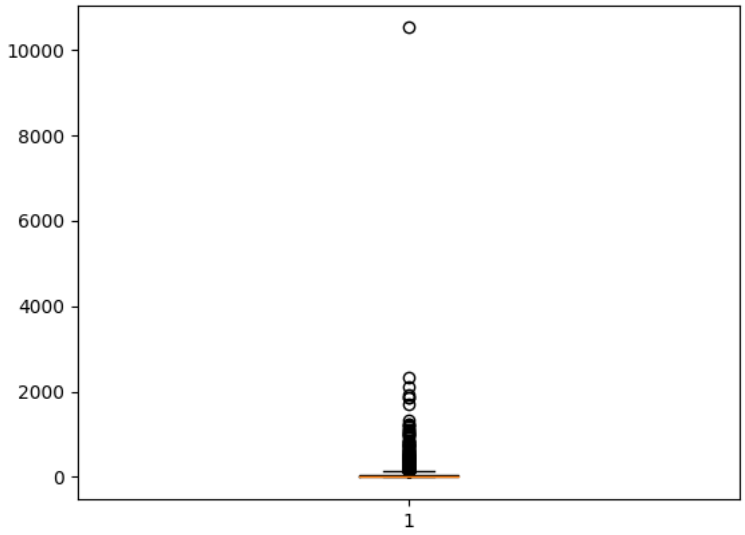
7. Identifying Outliers with Visualization

Box Plot

The box plot is a standardized way of displaying the distribution of data based on the five-number summary (minimum, first quartile (Q1), median, third quartile (Q3), and maximum). It is often used to identify data distribution and detect outliers. The line of code below plots the box plot of the numeric variable duration.

```
from matplotlib import pyplot as plt

      plt.boxplot(df["duration"])
      plt.show()
```



Histogram

A histogram is used to visualize the distribution of a numerical variable. An outlier will appear outside the overall pattern of distribution.

```
plt.hist(df['duration'])
plt.show()
```

Scatterplot

A scatterplot visualizes the relationship between two quantitative variables. The data are displayed as a collection of points, and any points that fall outside the general clustering of the two variables may indicate outliers. The lines of code below generate a scatterplot between the variable's duration and index.

```
fig, ax = plt.subplots(figsize=(12,6))
ax.scatter(df['duration'], df['item'])
ax.set_xlabel('Duration')
ax.set_ylabel('Type of Usage')
plt.show()
```

7.1 Outlier Treatment

In the previous sections, we learned about techniques for outlier detection. However, this is only half of the task. Once we have identified the outliers, we need to treat them. There are several techniques for this, and we will discuss the most widely used ones below.

7.1.1 Quantile-based Flooring and Capping

In this technique, we will do the flooring (e.g., the 10th percentile) for the lower values and capping (e.g., the 90th percentile) for the higher values. The lines of code below print the 10th and 90th percentiles of the variable 'Income', respectively. These values will be used for quantile-based flooring and capping.

```
print(df['duration'].quantile(0.10))
print(df['duration'].quantile(0.90))

df["duration"] = np.where(df["duration"] <1.0, 1.0,df['Income'])
df["duration"] = np.where(df["Income"] > 338.0,
338.0,df['Income'])
```

7.1.2 Trimming

In this method, we completely remove data points that are outliers. Consider the 'duration' variable, which had a minimum value of 0 and a maximum value of 1038.

```
index = df[(df['duration'] >= 338)|(df['duration'] <= 1)].index
df.drop(index, inplace =True)
print(df['duration'].describe())
```

7.1.3 Replacing Outliers with Median Values

In this technique, we replace the extreme values with median values. It is advised to not use mean values as they are affected by outliers

‘

```
print(df['duration'].quantile(0.50))
print(df['duration'].quantile(0.95))
df['duration'] = np.where(df['duration'] > 95%, mean,
df['duration'])
```

8. Evaluation Task (Unseen) [Expected time = 30mins for tasks]

Please find the autos.csv data in the zip folder.

8.1 Practice Task 1

Find the ? in given data and replace it with nan

8.2 Practice Task 2

Count Missing values in each column and display the results.

8.3 Practice Task 3

Calculate the mean value for the 'horsepower' column:

8.4 Practice Task 4

Replace "NaN" by mean value:

8.5 Practice Task 5

Find the car that have maximum highway mile per gallon

8.6 Practice Task 6

Find all honda car details.

8.7 Practice Task 7

Count total cars per company

8.8 Practice Task 8

Find each company's highest price car

9. Evaluation criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Table 3: Evaluation of the Labs

Sr. No.	Description	Marks
1	Problem Modeling	20
2	Procedures and Tools	10
3	Practice tasks and Testing	35
4	Evaluation Tasks (Unseen)	20
5	Comments	5
6	Good Programming Practices	10