Data Cleaning

Objectives

In this reading assignment you'll be able to:

- Exemplify Clean common data problems in structured data: Tabular data
- Exemplify common issues with unstructured data: Text and Images
- Explain the need for data versioning

Prerequisites required

- ML Level One

Data Cleaning Process

The first step to data cleaning is to identify impurity in your dataset. Impurities creates cleaning problems in the dataset. Some Common Cleaning Problems are:

- Irrelevant feature
- Parsing and Type Conversions
- Structural Errors (check for typos and mislabeled classes)

Irrelevant feature

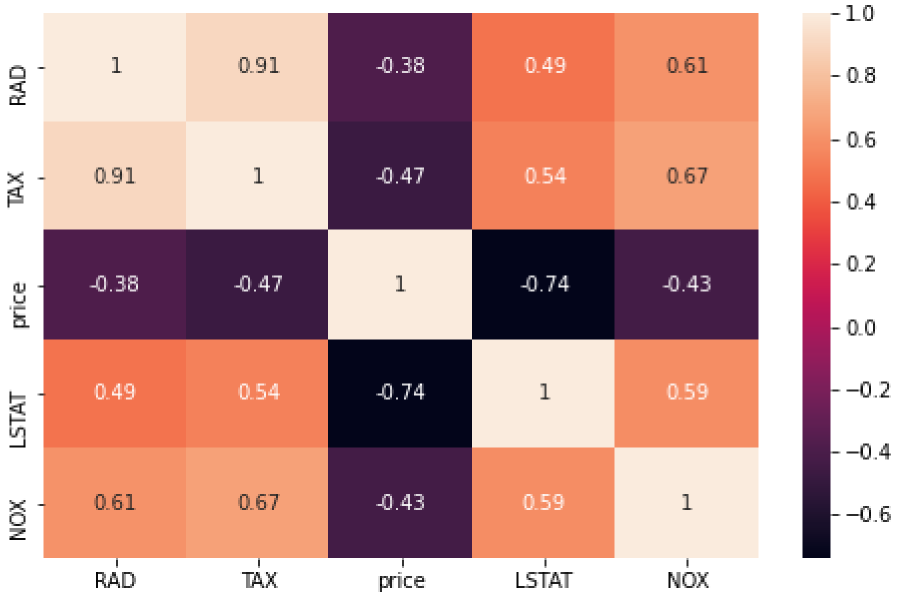


Figure 1: Heatmap

Let's say you want to fit a regression model, and you plotted a correlation heatmap of the features of the dataset. From the given heatmap, you know `TAX and RAD` are independent variables. Therefore either `TAX` or `RAD` becomes irrelevant because there is no point in using correlated independent features in the Linear Regression problem.

There is another problem you will face so called parsing and type conversion. Your dataset may contains datetime features. Using datetime column you can extract other features which is called parsing. Type conversion problem is just datatype conversion problem. Let's discuss parsing and type conversion on the new exmaple.

Parsing and Type Conversions

Suppose a King's garage buys used Ford's cars and sells it to other customers. King's garage wants to predict how much a car will cost for them when they buy it form owners. Column details:

- purchase_date: Date of purchase of new car
- sold_date: Date of car sold in King's garage
- model: Car's model name
- car_class: Class of cars
- Price in dollars: Cost price of King's garage

In [ ]:

```
import pandas as pd
car = {
    "purchase_date":['2016-04-16','2015-11-21','2015-07-06','2018-12-02'],
    "sold_date": ['2018-05-22','2017-09-18','2016-01-01','2019-08-04'],
    "model":['e-series', 'mustang', 'mustang' , 'E-SERIES'],
    "car_class":['third','first',1,'first'],
    "Price in dollars":[3495.0,4217.0,5000.0,4225.0]

}
df = pd.DataFrame(car)
df
```

Out[1]:

	purchase_date	sold_date	model	car_class	Price in dollars
0	2016-04-16	2018-05-22	e-series	third	3495.0
1	2015-11-21	2017-09-18	mustang	first	4217.0
2	2015-07-06	2016-01-01	mustang	1	5000.0
3	2018-12-02	2019-08-04	E-SERIES	first	4225.0

In [ ]:

```
df.dtypes
```

Out[3]:

```
purchase_date        object
sold_date            object
model                object
car_class            object
Price in dollars    float64
dtype: object
```

Type Conversion

In the given data set `Price in dollars` Column has all integer value. However, datatype of this column is float type. Therefore, you need to convert it into integer using pandas astype() (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.astype.html) method. Animation Here (https://drive.google.com/uc?id=1XeL-x5fsnVwXLIziT1TEyQ3a5izEQvCM)

In [ ]:

```
df['Price in dollars'] = df['Price in dollars'].astype('int')
df
```

Out[4]:

	purchase_date	sold_date	model	car_class	Price in dollars
0	2016-04-16	2018-05-22	e-series	third	3495
1	2015-11-21	2017-09-18	mustang	first	4217
2	2015-07-06	2016-01-01	mustang	1	5000
3	2018-12-02	2019-08-04	E-SERIES	first	4225

Parsing

In this section, you will see examples of DateTime parsing. Using DateTime data, you will calculate days of car used.

- Parsing is mostly used in Web scraping to clean HTML DOM. Animation Here (https://drive.google.com/uc?id=1aN2P41W47zwnVBzZFWHgV8pfQ-IuQXGw)

The column `purchase_date` and `sold_date` has object datatype; however, it should be in DateTime format. Let's type convert object type dates to DateTime format/datatype.

In [ ]:

```
df['purchase_date'] = pd.to_datetime(df['purchase_date'], format ="%Y-%m-%d")
df['sold_date'] = pd.to_datetime(df['sold_date'], format ="%Y-%m-%d")
df.dtypes
```

Out[5]:

```
purchase_date       datetime64[ns]
sold_date           datetime64[ns]
model                       object
car_class                   object
Price in dollars             int64
dtype: object
```

Let's calculate how many days the owner used his car and went to sell it to King's garage.

In [ ]:

```
df['used days'] = (df['sold_date'] - df['purchase_date']).dt.days
df
```

Out[7]:

	purchase_date	sold_date	model	car_class	Price in dollars	used days
0	2016-04-16	2018-05-22	e-series	third	3495	766
1	2015-11-21	2017-09-18	mustang	first	4217	667
2	2015-07-06	2016-01-01	mustang	1	5000	179
3	2018-12-02	2019-08-04	E-SERIES	first	4225	245

Structural Errors

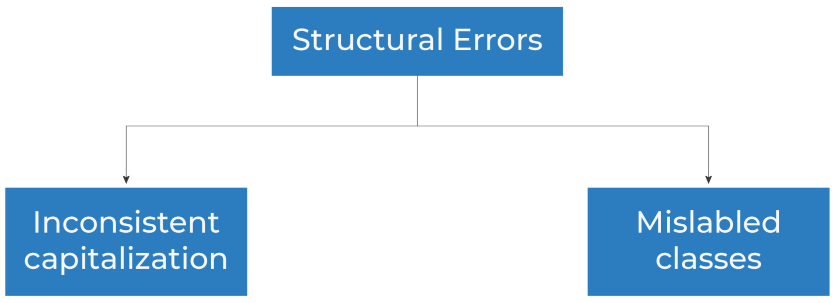


Figure 2: Structural Errors in Data

Inconsistent Capitalization

- In the column model, the e-series is in uppercase and lowercase both. It would be best if you made it consistent because while encoding this data, the encoder will take the lowercase and uppercase word as a different entity. Consistent means either lowercase or in uppercase. In our case, I will make it lowercase.

Animation Here (https://drive.google.com/uc?id=1iJqYPParauXfy763Ft_DPfHvZsQCDqUg)

Let's solve inconsistent capitalization problem.

In [ ]:

```
df.model = [x.lower() for x in df['model']]
df
```

Out[8]:

	purchase_date	sold_date	model	car_class	Price in dollars	used days
0	2016-04-16	2018-05-22	e-series	third	3495	766
1	2015-11-21	2017-09-18	mustang	first	4217	667
2	2015-07-06	2016-01-01	mustang	1	5000	179
3	2018-12-02	2019-08-04	e-series	first	4225	245

Mislabeled Classes

- The second problem of structural error is the mislabeled class. In the column `car_class,` you will see the use of number 1 and first. One and first is logically same. You can label this correctly by replacing 1 with first. Animation here (https://drive.google.com/uc?id=1Qp_KyCXLih3ExOoPqBST-w6_QukvKwGo)

Let's Solve Mislabeled class problem.

In [ ]:

```
df.car_class.replace(1, 'first', inplace = True)
df
```

Out[9]:

	purchase_date	sold_date	model	car_class	Price in dollars	used days
0	2016-04-16	2018-05-22	e-series	third	3495	766
1	2015-11-21	2017-09-18	mustang	first	4217	667
2	2015-07-06	2016-01-01	mustang	first	5000	179
3	2018-12-02	2019-08-04	e-series	first	4225	245

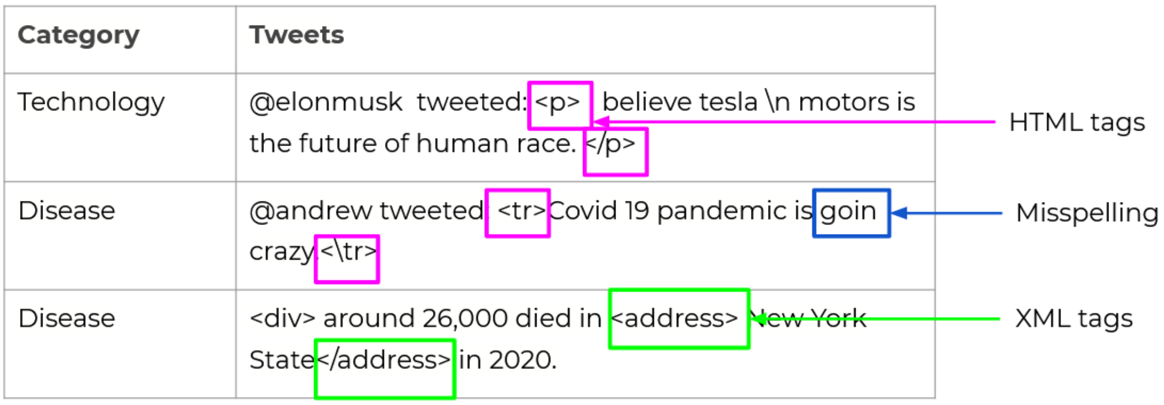
Now let's discss about Cleaning problem in unstructured data.

Data cleaning problems in Unstructured data: Text and Images

Cleaning problems in Text data

Text data are mostly generated on social media or the web. Therefore can occur with HTML tags or with XML tags. Some problems that text data might have are:

- Misspelling
- Random white spaces in-between words
- HTML tags,
- XML tags and
- special characters



Category	Tweets
Technology	@elonmusk tweeted: <p> believe tesla \n motors is the future of human race. </p>
Disease	@andrew tweeted <tr>Covid 19 pandemic is goin crazy <\tr>
Disease	<div> around 26,000 died in <address> New York State</address> in 2020.

HTML tags
Misspelling
XML tags

All these problems are inspected and solved using the NLTK library. There are some other libraries like TextBlob, Gensim, etc., to inspect and solve these problems.

Cleaning problems in Images

Now let’s discuss data quality issues in the images. Some well-known quality issues of images are:

- Blur
- Missing pixels
- Overexposure
- Underexposure
- Noise etc.

There are other quality issues like distortion, partial obstruction, etc. An Image with these problems is said to be a low-quality image. Low-quality images are an unavoidable reality for many real-world computer vision applications. They can be life-threatening at an extreme level, limiting autonomous vehicles and traffic controllers to navigate their surroundings safely. Therefore it is essential to measure the quality of images. Image Quality Assessment is a challenging task yet crucial. You can inspect image quality using:

Hosaka Plot:

- Hosaka Plot is used to evaluate the quality of a compressed image.

Histogram:

- Since histograms contain cumulated information of the image, the grey value histogram measure gives a global impression of the image quality by considering statistically relevant effects.

Data Versioning

The basic idea of data versioning is that, as you work, you create static copies of your work so that you can refer it back.

Table 1: Sentiment analysis data(raw)

Sentiment	Message
Happy	\\r May all the love you gave to us come back to you a hundredfold on this special day!
Happy	\\n Summer will end soon enough Hurrey <:)
Sad	#Bad_day \n Some days are just bad, that's all.

For example, You have raw data of sentiment analysis. After getting that data, you preprocess it and clean it. The preprocessing step may include the removal of a special character using Regex. It is a static copy of your data. Let's say it is version one(v1).

Table 2: Sentiment analysis data(v1)

Sentiment	Message
Happy	may all the love you gave to us come back to you a hundredfold on this special day
Happy	summer will end soon enough hurrey
Sad	bad day some days are just bad that's all

The `Sentiment analysis data(v1)` is particularly important when working in a team because it aids in reproducibility; there is no point to preprocess data again. However, sharing of processed data which changes often makes no sense. Therefore, it is preferable to share raw data and the scripts and notebooks that transform raw into the processed data. The idea that you should version your data is somewhat controversial.

Suppose you built the sentiment analysis model using the pre-processed table 1 dataset. After some months, you collected additional data. You can either use newly collected data to create a model or use combined data to create a model. Data changes over time; therefore, data versioning is used. Data versioning is used to keep track of the collected dataset at a different time. It is somewhat similar to git, which is used to track changes in any set of files during software development.

Some open source data version control system are: DVC, MLflow, etc.

Note the exciting point. Here raw data represents ground reality; however, preprocessed data don't.

When should you version your data?

- When making metadata changes, like adding or deleting columns or changing the datatype.
- When you're training experimental machine learning models. You can apply a different version of your data and test its performance on the model.

When should you consider not versioning your data?

- If your data is large enough that storing a versioned copy would be expensive.

- If you are version controlling your code in git hub. Git hub doesn't support large files. You can use GitLFS (https://git-lfs.github.com/) in this case, but in general, it is not recommended.

Key Take-Aways

Let’s conclude this chapter with some key takeaways:

- A feature of a dataset becomes irrelevant depending upon the domain problem. In a regression problem, if two independent feature is correlated, one become irrelevant.
- Parsing is used to generate a new feature in DateTime data.
- If data contains Inconsistent capitalization, then it must be made consistent.
- Misspelling, Extra spaces, HTML tags, XML tags are the typical quality issues on the text data.
- Blur, distortion, overexposure, underexposure, noise, missing pixels, etc., are common quality issues with the image.
- Data versioning is generally used to keep track of different versions of data.

References

- Cai, L. and Zhu, Y., 2015. The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. Data Science Journal, 14, p.2. DOI: http://doi.org/10.5334/dsj-2015-002 (http://doi.org/10.5334/dsj-2015-002)
- McKnight, W. (2013). Information management: strategies for gaining a competitive advantage with data. Newnes.
- Olson, J. E. (2003). Data quality: the accuracy dimension. Elsevier.

- Sadiq, S. (Ed.). (2013). Handbook of data quality: Research and practice. Springer Science & Business Media.