4.2-Data Cleaning - Outliers and Text Handling

KeytoDataScience.com

Data cleaning, or cleansing, is the process of correcting and deleting inaccurate records from a database or table. It mainly consists of identifying and replacing incomplete, inaccurate, irrelevant, or otherwise problematic ('dirty') data and records.

Table of Contents

1 What is Outlier and How to detect Outliers?

📌 **Q. What is Outlier??**

Outlier is the data that is out of range of data that we expect

__Noise Value: Unwanted and wrong data

Example: Let's take Average annual income of 50–100 people in an normal neighbourhood. If we include Bill Gates in that list, then automatically average salary of everyone in this group will surpass million. So, Bill Gates would be an outlier in this case.

An outlier is not a false value or void in meaning. It is definite and accurate but when it is linked with the other data in your model, it is just not in the same range.

📌 **Q. How to detect Outliers?**

1. Using box plot

2. Using Z score
3. IQR (Inter Quartile Range)

In [1]:
```
import pandas as pd
import numpy as np
```

In [8]:
```
outlier_data= [10,22,22,33,22,41,74,83,25,30,60,20,62,74,23,890,899,
               32,68, 97,43,12,25,92,23,12,59,24,33,85,30,1001,45,22,52,1005]
```

In [3]:
```
df = pd.DataFrame(outlier_data, columns=['Values'])
```

In [4]:
```
df.head()
```

Out[4]:

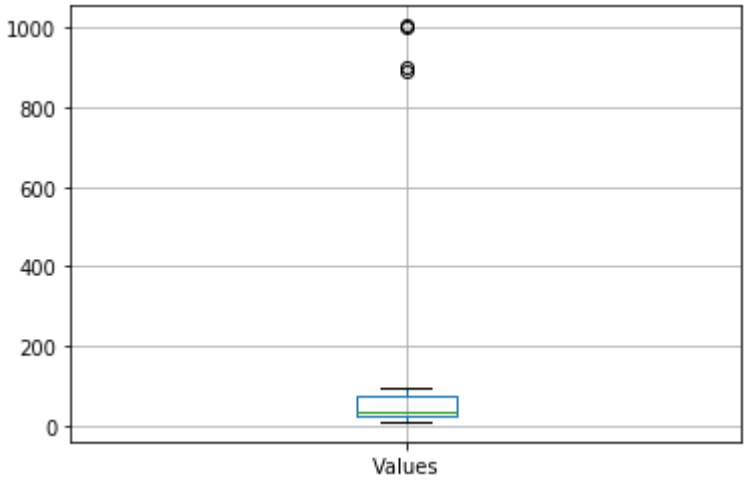
	Values
0	10
1	22
2	22
3	33
4	22

1.1 Box Plot

In [28]:
```
#Box Plot uses IQR

df.boxplot(column=['Values'])
```

Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x243fd07c388>
```



1.2 Z score

The value of the z-score tells, how many standard deviations a datapoint is away from the mean.

If a z-score is equal to 0, it is on the mean.

Formula for Z score = (Observation — Mean)/Standard Deviation

$$z = (X - \mu) / \sigma$$

The z-score is positive if the value lies above the mean, and negative if it lies below the mean.

In [15]:
```
print('Mean:',df.mean().values)
print('Standard Deviation:',df.std().values)
print('Median:',df.median().values)
```

```
Mean: [143.05555556]
Standard Deviation: [290.46504038]
Median: [37.]
```

Let's create a function to detect outliers using Z Test

In [6]:
```
outliers=[]
def detect_outlier(data):

    threshold=2
    mean_1 = np.mean(data)
    std_1 = np.std(data)


    for y in data:
        z_score= (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            outliers.append(y)
    return outliers
```

In [7]:
```
outlier_datapoints = detect_outlier(outlier_data)
outlier_datapoints
```

Out[7]:
```
[890, 899, 1001, 1005]
```

In above cells we saw:

- Mean: 143.05
- Standard Deviation: 290.46
- Median: 37

We can confirm by checking the output of `detect_outlier` function that, these 4 data points are actually outliers.

1.3 InterQuartile Range

IQR is 25% - 75% values in the dataset, or data between 25th to 75th percentile of dataset

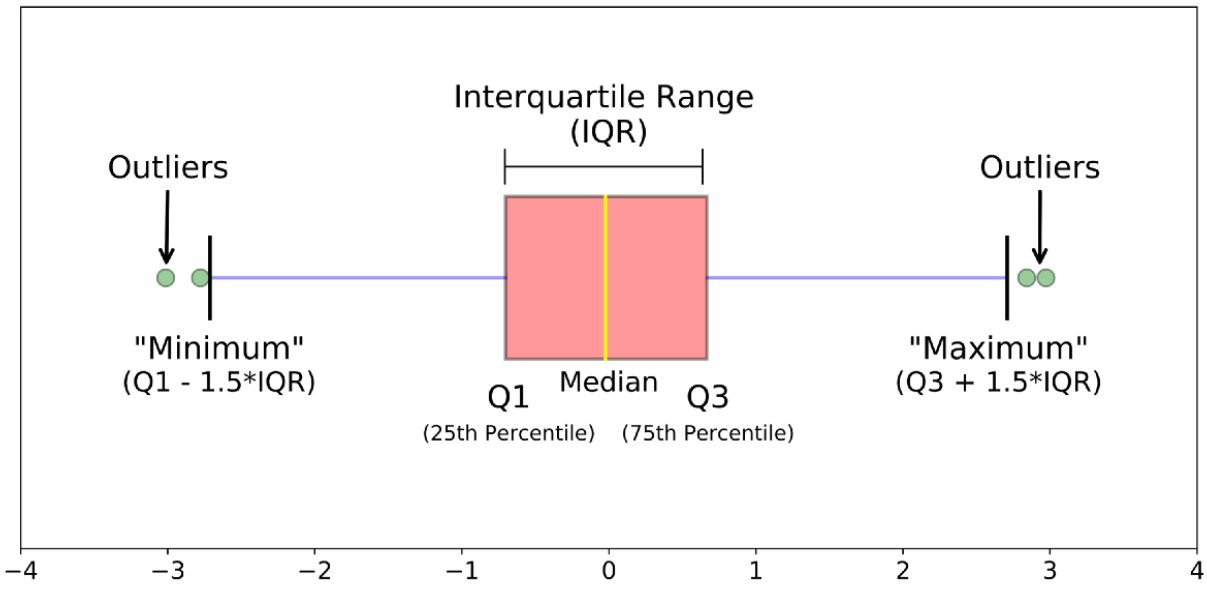
IQR tells how spread the middle values are. It can be used to tell when a value is too far from the middle.

An outlier is a point which falls more than 1.5 times the interquartile range above the third quartile or below the first quartile.

```
- Quartile: Divide dataset into 4 equal parts, it is called quartiles.
2nd and 3rd quartile is IQR
```

Steps

- Arrange the data in increasing order.
- Calculate the first quartile(q1) and third quartile(q2)
- Find the interquartile range (q3 - q1)
- Find lower bound q1 - (1.5 * iqr)
- Find upper bound q3 + (1.5 * iqr)
- Anything that lies lower and upper bound is an outlier



Read more regarding IQR and quatile here

In [32]:
```
sorted(outlier_data)
```

Out[32]:
```
[10,
 12,
 12,
 20,
 22,
 22,
 22,
 22,
 23,
 23,
 24,
 25,
 25,
 30,
```

```
30,
32,
33,
33,
41,
43,
45,
52,
59,
60,
62,
68,
74,
74,
83,
85,
92,
97,
890,
899,
1001,
1005]
```

In [33]:
```
q1, q3= np.percentile(outlier_data ,[25,75])
```

In [34]:
```
iqr = q3 - q1
iqr
```

Out[34]:
```
51.0
```

In [35]:
```
lower_bound_val = q1 - (1.5 *  iqr)
upper_bound_val = q3 + (1.5 * iqr)
```

Lower bound q1 - (1.5 * iqr)

In [ ]:
```
lower_bound_val
```

Upper bound q3 + (1.5 * iqr)

In [37]:
```
upper_bound_val
```

Out[37]:
```
150.5
```

In [38]:
```
outlier_iqr_list=[]
for d in outlier_data:
    if d < lower_bound_val or d > upper_bound_val:
        outlier_iqr_list.append(d)

outlier_iqr_list
```

Out[38]:
```
[890, 899, 1001, 1005]
```

We get the same set of outliers as in Z-Score method. However, this may not always be true.

Anything that lies outside lower and upper bound is an outlier

In [39]:
```
#Removing the Outlier
df = df[(df.Values<150.5) & (df.Values>-53.5)]
```

In [40]:
```
df.head()
```

Out[40]:

	Values
0	10
1	22
2	22
3	33
4	22

↥ back to top

2 Text data Cleaning

In machine learning we ultimately always work with numbers or specifically vectors.

A vector is simply an array of numbers, such as (1, 2, 3)—or a nested array that contains other arrays of numbers, such as (1, 2, (1, 2, 3)).

The main points:

- All non-numerical data types (such as images, text, and categories) must eventually be represented as numbers
- In machine learning, the numerical representation will be in the form of an array of numbers—that is, a vector

Therefore, Text data needs to be cleaned and encoded to numerical values before giving them to machine learning models, this process of cleaning and encoding is called as Text Preprocessing

---

💡💡💡💡💡 **Steps to clean Text Data** 💡💡💡💡💡

---

1. Capitalization/De-capitalization (Optional)
2. Removal of punctuation
3. Tokenization
4. Removal of Stopwords
5. Stemming/Lemmatizing

In [23]:
```

```

```
#Fist letls install nltk packages

!pip install nltk

#or
# pip install nltk
```

```
Collecting nltk
  Downloading nltk-3.6.7-py3-none-any.whl (1.5 MB)
Requirement already satisfied: joblib in c:\users\prateek\appdata\local\programs\python
\python39\lib\site-packages (from nltk) (1.0.1)
Collecting click
  Downloading click-8.0.3-py3-none-any.whl (97 kB)
Requirement already satisfied: tqdm in c:\users\prateek\appdata\local\programs\python\py
thon39\lib\site-packages (from nltk) (4.62.3)
Collecting regex>=2021.8.3
  Downloading regex-2021.11.10-cp39-cp39-win_amd64.whl (273 kB)
Requirement already satisfied: colorama in c:\users\prateek\appdata\local\programs\pytho
n\python39\lib\site-packages (from click->nltk) (0.4.4)
Installing collected packages: regex, click, nltk
Successfully installed click-8.0.3 nltk-3.6.7 regex-2021.11.10
```

In [24]:
```
import nltk
```

In [25]:
```
data = "it has become imperative for an organization to have a structure in place to mi
dealing with text data has never been more important."
```

2.1 Removal of punctuation:

Punctuation marks don't add much information to the data, hence we need to remove it.

In [26]:
```
#importing python library
import string
print( string.punctuation)

#checking each character whether it contains any punctuation
nopunc = []
for char in data:
    if char not in string.punctuation:
        nopunc.append(char)

#joining all the individual characters into words
nopunc = ''.join(nopunc)
nopunc
```

Out[26]:
```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
'it has become imperative for an organization to have a structure in place to mine actio
nable insights from the text being generated from social media analytics to risk managem
ent and cybercrime protection dealing with text data has never been more important'
```

2.2 Tokenization:

The process of breaking down a text paragraph into smaller chunks such as words or sentence is called `Tokenization` .

In [27]:
```
#Simple tokenization example using split function on Whitespaces
nopunc.split()
```

Out[27]:
```
['it',
 'has',
 'become',
 'imperative',
 'for',
 'an',
 'organization',
 'to',
 'have',
 'a',
 'structure',
 'in',
 'place',
 'to',
 'mine',
 'actionable',
 'insights',
 'from',
 'the',
 'text',
 'being',
 'generated',
 'from',
 'social',
 'media',
 'analytics',
 'to',
 'risk',
 'management',
 'and',
 'cybercrime',
 'protection',
 'dealing',
 'with',
 'text',
 'data',
 'has',
 'never',
 'been',
 'more',
 'important']
```

In [28]:
```
#using nltk library to tokenize on whitespaces and punctuation
from nltk.tokenize import word_tokenize
tokens = word_tokenize(nopunc)
tokens
```

Out[28]:
```
['it',
 'has',
 'become',
 'imperative',
 'for',
 'an',
 'organization',
 'to',
 'have',
```

```
'a',
'structure',
'in',
'place',
'to',
'mine',
'actionable',
'insights',
'from',
'the',
'text',
'being',
'generated',
'from',
'social',
'media',
'analytics',
'to',
'risk',
'management',
'and',
'cybercrime',
'protection',
'dealing',
'with',
'text',
'data',
'has',
'never',
'been',
'more',
'important']
```

2.3 Removal of Stopwords:

Stopwords considered as noise in the text.

Stopwords are the common words such as 'a', 'an', 'the', 'this', etc which again don't add much meaning to the data and should be removed.

This improves the computational time and also free up the database space.

In [30]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Prateek\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

Out[30]:
```
True
```

In [31]:
```
from nltk.corpus import stopwords

stop_words=set(stopwords.words("english"))
print(stop_words)
```

```
{"you're", 'during', 'she', 'those', 'not', "hadn't", 'been', 'both', 'what', 'me', 'ar
e', 'is', 'm', 'can', 'up', 'while', 'll', 'further', 'they', 'if', 'won', 'doing', 'aga
```

```
in', "haven't", 'between', 'which', 'my', 'them', 'now', 'hadn', 'all', 'our', 'isn', 'w
ere', 'yours', 'why', "wasn't", 'i', 'just', 'd', "should've", "mustn't", 'or', 'where',
"you'll", 'under', 'once', 'his', 'their', 'him', "needn't", 's', 'mustn', 'that', 're',
'at', 'do', 'her', 'will', 'about', "doesn't", 'by', "you've", 'any', 'same', 'we', 'tha
n', 'itself', 'has', 'very', "wouldn't", 'of', 'own', 'didn', 'a', 'ma', 'with', 'into',
'through', 'off', 'mightn', 'having', 'above', 'hers', 'until', "mightn't", "that'll",
'before', 'more', 'herself', 'needn', 've', "hasn't", 'hasn', 'it', 'these', "aren't",
"couldn't", 'in', 'o', 'aren', 'too', 'ourselves', 'for', "she's", 'he', 'few', "you'd",
'be', 'couldn', 'you', 'don', 'themselves', 'each', 'then', 'who', 'being', 'most', "sha
n't", 'no', 'some', 'shan', "weren't", 'did', 'there', 'as', 'yourselves', 'was', 'y',
'when', 'other', 'wasn', 'over', 'shouldn', "isn't", 'wouldn', 'does', 'had', 'but', "sh
ouldn't", 'how', 'such', 'himself', 'its', 'so', 'against', "it's", 'to', "won't", 't',
'from', 'this', 'because', 'here', 'doesn', 'have', 'ours', 'myself', 'and', 'theirs',
"don't", 'only', 'ain', 'am', 'yourself', 'whom', 'out', 'weren', 'down', 'an', 'below',
'on', 'nor', "didn't", 'the', 'your', 'after', 'should', 'haven'}
```

In [32]:

```
filtered_data=[]
for w in tokens:
    if w not in stop_words:
        filtered_data.append(w)
print("Tokenized Sentence:",tokens)
print("------------------------------------------------------------------------------------------")
print("After removing Stop words:",filtered_data)
```

```
Tokenized Sentence: ['it', 'has', 'become', 'imperative', 'for', 'an', 'organization',
'to', 'have', 'a', 'structure', 'in', 'place', 'to', 'mine', 'actionable', 'insights',
'from', 'the', 'text', 'being', 'generated', 'from', 'social', 'media', 'analytics', 't
o', 'risk', 'management', 'and', 'cybercrime', 'protection', 'dealing', 'with', 'text',
'data', 'has', 'never', 'been', 'more', 'important']
------------------------------------------------------------------------------------------
-------------
After removing Stop words: ['become', 'imperative', 'organization', 'structure', 'plac
e', 'mine', 'actionable', 'insights', 'text', 'generated', 'social', 'media', 'analytic
s', 'risk', 'management', 'cybercrime', 'protection', 'dealing', 'text', 'data', 'neve
r', 'important']
```

2.4 Stemming vs Lemmatization

`Stemming` and `Lemmatization` both generate the root form of the words.

The difference is that:

- stem might not be an actual word whereas,
- lemma is an actual language word.

`Stemming` follows an algorithm with steps to perform on the words which makes it faster.

Whereas, in `lemmatization`, you used WordNet corpus and a corpus for stop words as well to produce lemma which makes it slower than stemming. You also had to define a parts-of-speech to obtain the correct lemma.

- If you `lemmatize` the word 'Caring', it would return 'Care'.
- If you `stem`, it would return 'Car' and this is erroneous.
- Words such as walking, running, swimming will return walk, run, swim for both `stemming` and `lemmatize`

2.4.1 Stemming:

It is the process of converting a word into its root or base form. Also, the prefixes or suffixes get removed from a word that results in a word that may or may not be meaningful.

📌 **For Example:**

- `running ---> run`
- `generated ---> gener`

In [33]:
```
from nltk.stem.porter import PorterStemmer

porter = PorterStemmer()
stemmed_words=[]
for w in filtered_data:
    stemmed_words.append(porter.stem(w))

print("After removing Stop words:",filtered_data)
print("-----------------------------------------------------------------------------------")
print("After Stemming:",stemmed_words)
```

```
After removing Stop words: ['become', 'imperative', 'organization', 'structure', 'plac
e', 'mine', 'actionable', 'insights', 'text', 'generated', 'social', 'media', 'analytic
s', 'risk', 'management', 'cybercrime', 'protection', 'dealing', 'text', 'data', 'neve
r', 'important']
-----------------------------------------------------------------------------------
After Stemming: ['becom', 'imper', 'organ', 'structur', 'place', 'mine', 'action', 'insi
ght', 'text', 'gener', 'social', 'media', 'analyt', 'risk', 'manag', 'cybercrim', 'prote
ct', 'deal', 'text', 'data', 'never', 'import']
```

2.4.2 Lemmatization:

It is a process that is similar to stemming that is converting a word into its root or base form but unlike stemming it always returns a proper word that can be found in a dictionary.

📌 **For Example:**

- `generated ---> generate`

In [ ]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lem_data = []
for word in filtered_data:
    lem_data.append(lemmatizer.lemmatize(word, 'v'))

lem_data
```

2.5 POS Tagging

The primary target of Part-of-Speech(POS) tagging is to identify the grammatical group of a given word. Whether it is a NOUN, PRONOUN, ADJECTIVE, VERB, ADVERBS, etc. based on the context.

POS Tagging looks for relationships within the sentence and assigns a corresponding tag to the word.

If no POS Tag is provided to WordNetLemmatizer, It defaults to nouns.

In [34]:
```
nltk.pos_tag(tokens)
```

Out[34]:
```
[('it', 'PRP'),
 ('has', 'VBZ'),
 ('become', 'VBN'),
 ('imperative', 'JJ'),
 ('for', 'IN'),
 ('an', 'DT'),
 ('organization', 'NN'),
 ('to', 'TO'),
 ('have', 'VB'),
 ('a', 'DT'),
 ('structure', 'NN'),
 ('in', 'IN'),
 ('place', 'NN'),
 ('to', 'TO'),
 ('mine', 'VB'),
 ('actionable', 'JJ'),
 ('insights', 'NNS'),
 ('from', 'IN'),
 ('the', 'DT'),
 ('text', 'NN'),
 ('being', 'VBG'),
 ('generated', 'VBN'),
 ('from', 'IN'),
 ('social', 'JJ'),
 ('media', 'NNS'),
 ('analytics', 'NNS'),
 ('to', 'TO'),
 ('risk', 'VB'),
 ('management', 'NN'),
 ('and', 'CC'),
 ('cybercrime', 'NN'),
 ('protection', 'NN'),
 ('dealing', 'VBG'),
 ('with', 'IN'),
 ('text', 'NN'),
 ('data', 'NNS'),
 ('has', 'VBZ'),
 ('never', 'RB'),
 ('been', 'VBN'),
 ('more', 'RBR'),
 ('important', 'JJ')]
```

In [36]:
```
# download tagsets
nltk.download('tagsets')
```

```
[nltk_data] Downloading package tagsets to
[nltk_data]     C:\Users\Prateek\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping help\tagsets.zip.
```

Out[36]:
```
True
```

In [37]:
```
nltk.help.upenn_tagset("VBZ")
```

```
VBZ: verb, present tense, 3rd person singular
    bases reconstructs marks mixes displeases seals carps weaves snatches
    slumps stretches authorizes smolders pictures emerges stockpiles
    seduces fizzes uses bolsters slaps speaks pleads ...
```

2.6 Vectorization (Optional)

After we have normalized the text, we can take the next step of actually encoding it in a numerical form. This is done by text vectorization—that is, by turning a piece of text into a vector.

Common approaches include:

- Term Frequency-Inverse Document Frequency (TF-IDF) vectorization
- Word embedding, as done with Word2vec or Global Vectors (GloVe)

Documents → **Normalized Text** → **Vectorized Text**

The quick fox.
The lazy dog.
The rabid hare.

→ [quick, fox]
[lazy, dog]
[rabid, hare]

→

quick	fox	lazy	dog	rabid	hare
0.32	0.23	0.0	0.0	0.0	0.0
0.0	0.0	0.12	0.23	0.0	0.0
0.0	0.0	0.0	0.0	0.56	0.12

↥ back to top

Great Job!
