Data Handling Using Pandas: Cleaning and Processing

towardsdatascience.com/data-handling-using-pandas-cleaning-and-processing-3aa657dc9418

April 6, 2019

Mastering Pandas to Deal with 'Dirty Data'

Keep calm, learn Pandas! (Source: Pixabay)

While practicing for some old Kaggle projects, I've realized that preparing data files before applying machine learning algorithms took a whole lot of time. This post is to review some of the beginner to advanced level data handling techniques with Pandas, written as a follow-up of a previous post. Let's get started without any delay !

For this post, I have used IMDB movie-dataset to cover the most relevant data-cleaning and processing techniques. We can start of with knowing more about the data-set as below

```
movies_df = pd.read_csv("movie_metadata.csv")
print "data-frame shape: ", movies_df.shape

>>> data-frame shape:  (5043, 28)
```

So the data-set has 5043 rows, 28 columns and, we can check the column names with

```
>>> column names:
['color' 'director_name' 'num_critic_for_reviews' 'duration'
 'director_facebook_likes' 'actor_3_facebook_likes' 'actor_2_name'
 'actor_1_facebook_likes' 'gross' 'genres' 'actor_1_name' 'movie_title'
 'num_voted_users' 'cast_total_facebook_likes' 'actor_3_name'
 'facenumber_in_poster' 'plot_keywords' 'movie_imdb_link'
 'num_user_for_reviews' 'language' 'country' 'content_rating' 'budget'
 'title_year' 'actor_2_facebook_likes' 'imdb_score' 'aspect_ratio'
 'movie_facebook_likes']
```

Before we can apply some ML algorithms to predict, let's say 'imdb_score', we need to investigate the data-set bit more, as it is not so well processed like Boston House Data-Set. First, I will discuss on how to handle missing data.

Handling Missing Data: DataFrame.isna( ), DataFrame.fillna( )

We can use `pandas.DataFrame.isna()` to detect missing values for an array like object. This returns a Boolean same-sized object where NA values, such as None or `numpy.NaN` , gets mapped to True and everything else is mapped to False. This does exactly the same with `pandas.DataFrame.isnull()` .

The above commands return the following output

```
null values:
      color  director_name  num_critic_for_reviews  duration  director_facebook_likes  ...  title_year  actor_2_facebook_likes  imdb_score  aspect_ratio  movie_facebook_likes
0     False          False                   False     False                    False  ...       False                   False       False         False                 False
1     False          False                   False     False                    False  ...       False                   False       False         False                 False
2     False          False                   False     False                    False  ...       False                   False       False         False                 False
3     False          False                   False     False                    False  ...       False                   False       False         False                 False
4      True          False                    True      True                    False  ...        True                   False       False          True                 False
5     False          False                   False     False                    False  ...       False                   False       False         False                 False
6     False          False                   False     False                    False  ...       False                   False       False         False                 False
7     False          False                   False     False                    False  ...       False                   False       False         False                 False
8     False          False                   False     False                    False  ...       False                   False       False         False                 False
9     False          False                   False     False                    False  ...       False                   False       False         False                 False
10    False          False                   False     False                    False  ...       False                   False       False         False                 False
11    False          False                   False     False                    False  ...       False                   False       False         False                 False
12    False          False                   False     False                    False  ...       False                   False       False         False                 False
13    False          False                   False     False                    False  ...       False                   False       False         False                 False
14    False          False                   False     False                    False  ...       False                   False       False         False                 False
15    False          False                   False     False                    False  ...       False                   False       False         False                 False
16    False          False                   False     False                    False  ...       False                   False       False         False                 False
17    False          False                   False     False                    False  ...       False                   False       False         False                 False
18    False          False                   False     False                    False  ...       False                   False       False         False                 False
19    False          False                   False     False                    False  ...       False                   False       False         False                 False
20    False          False                   False     False                    False  ...       False                   False       False         False                 False
21    False          False                   False     False                    False  ...       False                   False       False         False                 False
```

Looking For Missing Data in data-frame

Rather than printing out the data-frame with True/False as entry, we can extract the relevant information by adding a `.sum()` along with the previous command. With this we can find total number of missing values for each column.

>>>

```
color                      19
director_name              104
num_critic_for_reviews     50
duration                   15
director_facebook_likes    104
actor_3_facebook_likes     23
actor_2_name               13
actor_1_facebook_likes     7
gross                      884
genres                     0
actor_1_name               7
movie_title                0
num_voted_users            0
cast_total_facebook_likes  0
actor_3_name               23
facenumber_in_poster       13
plot_keywords              153
movie_imdb_link            0
num_user_for_reviews       21
language                   12
country                    5
content_rating             303
budget                     492
title_year                 108
actor_2_facebook_likes     13
imdb_score                 0
aspect_ratio               329
movie_facebook_likes       0
dtype: int64
```

Adding another .sum() returns the total number of null values in the data-set.

```
print "total null values: ", movies_df.isna().sum().sum()
```

```
>> total null values: 2698
```

One of the easiest ways to remove rows containing NA *is to* ***drop*** *them,* either when all column contain NA or any column contain NA. Let's start with dropping rows that contain NA values in any of the columns.

```
clean_movies_df = movies_df.dropna(how='any')
```

```
>>>

new dataframe shape:  (3756, 28)
old dataframe shape:  (5043, 28)
```

So dropping rows containing NA values in any of the columns resulted in almost 1300 rows reduction. This can be important for data-sets with less number of rows where dropping all rows with any missing value can cost us losing necessary information. In that case we can use `pandas.DataFrame.fillna()` method to ***fill NA/NaN values*** using a specified method. Easiest way to fill all the NA/NaNs with some fixed value, for example 0. We can do that simply by

```
movies_df.fillna(value=0, inplace = True)
```

Instead of filling up all the missing values with zero, we can choose some specific columns and then use `DataFrame.fillna()` method as below —

```
movies_df[['gross', 'budget']] = movies_df[['gross', 'budget']].fillna(value=0)
```

For columns with 'object' dtypes, for example 'language' column, we can use some words like "no info" to fill up the missing entries.

```
movies_df['language'].fillna("no info", inplace=True)
```

Another `method` to fill the missing value could be **`ffill`** method, which propagates last valid observation to the next. Similarly **`bfill`** method uses next observation to fill gap.

```
movies_df['language'].fillna(method='ffill', inplace=True)
```

Another effective method is to ***use the mean of the column to fill the missing values*** as below

```
movies_df['budget'].fillna(movies_df[budget].mean(), inplace=True)
```

For more details on how to use Pandas to deal with missing values, you can check the Pandas user guide document on missing data.

Duplicate Data in a Data-Frame: DataFrame.duplicated( )

Apart from missing data, there can also be *duplicate rows* in a data-frame. To find whether a data-set contain duplicate rows or not we can use Pandas DataFrame.duplicated() either for all columns or for some selected columns. **`pandas.Dataframe.duplicated()`** returns a Boolean series denoting duplicate rows. Let's first find how many duplicate rows are in this movies data-set.

```
duplicate_rows_df = movies_df[movies_df.duplicated()]

print "number of duplicate rows: ", duplicate_rows_df.shape

>>>
number of duplicate rows:  (45, 28)
```

So there are 45 rows with duplicate elements present in each column. We can check this for individual column too —

```
duplicated_rows_df_imdb_link= movies_df[movies_df.duplicated(['movie_imdb_link'])]

>>>
(124, 28)
```

So there are 124 cases where imdb link is same, another way to check the same, is to use **pandas.Series.unique()** method. Let's see:

```
>>>
4919
```

So total number of unique links are 4919 and if you have noticed that duplicate links were 124, adding them gives (4919 + 124 = 5043) total number of rows. It is necessary to select the unique rows for better analysis, so at least we can drop the rows with same values in all column. We can do it simply using **pandas.DataFrame.drop_duplicates()** as below

```
print "shape of dataframe after dropping duplicates", movies_df.drop_duplicates().shape
```

```
>>>
shape of dataframe after dropping duplicates (4998, 28)
```

Binning Data: pandas.cut( )

Another very important data processing technique is ***data bucketing or data binning***. We will see an example here with binning IMDb-score using **pandas.cut()** method. Based on the score [0.,4., 7., 10.], I want to put movies in different buckets ['shyyyte', 'moderate', 'good']. As you can understand movies with score between 0–4 will be put into the 'shyyyte' bucket and so on. We can do this with the following lines of code

```
op_labels = ['shyttte', 'moderate', 'good']
category = [0.,4.,7.,10.]

movies_df['imdb_labels'] = pd.cut(movies_df['imdb_score'], labels=op_labels, bins=category,
include_lowest=False)
```

Here a new column 'imdb_labels' is created containing the labels and let's take a look on it —

```
>>>

      movie_title     imdb_score     imdb_labels

209     Rio 2               6.4           moderate
210     X-Men 2              7.5             good
211     Fast Five           7.3             good
212     Sherlock Holmes:..  7.5             good
213     Clash of the...     5.8           moderate
214     Total Recall        7.5             good
215     The 13th Warrior    6.6           moderate
216     The Bourne Legacy   6.7           moderate
217     Batman & Robin      3.7             shyttte
218     How the Grinch..    6.0           moderate
219     The Day After T..   6.4           moderate
```

To fully capitalize **pandas.cut()** method, you can check the docs.

Detecting Outliers in a Data-Set:

Most of the times for Exploratory Data Analysis (EDA), ***outlier detection*** is an important segment, as, outlier for particular features may distort the true picture, so we need to disregard them. Specifically, outliers can play havoc when we want to apply machine learning algorithm for prediction. At the same time outliers can even help us for anomaly detection. So let's see how we can use Pandas to detect outliers in this particular data-frame.

Seaborn Box Plot:

Box plot is a standard way of visualizing distribution of data based on median, quartiles and outliers. Probably you already know what exactly are these quantities but still I made short review in the figure below.

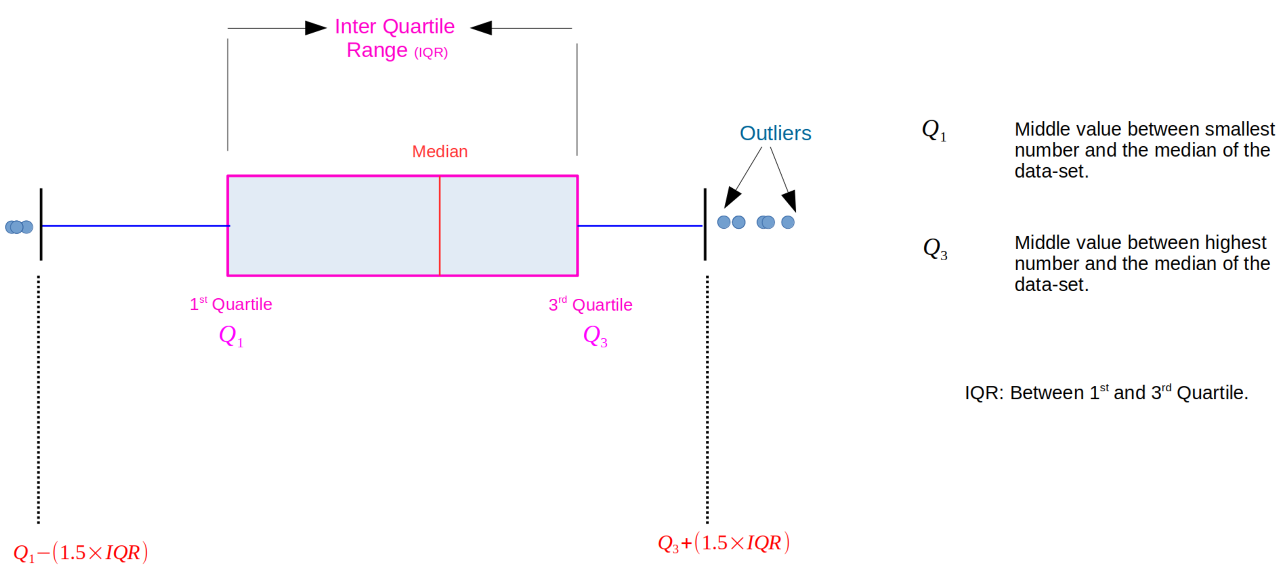

Figure 1: Schematic of Box Plot (Source: Author)

We can use python data visualization library Seaborn to plot such box plots. Let's plot the distribution of number of actors who featured in the movie poster using a box plot.

```
sns.boxplot(x=movies_df['facenumber_in_poster'], color='lime')
plt.xlabel('No. of Actors Featured in Poster', fontsize=14)
plt.show()
```

The code above results in the plot below

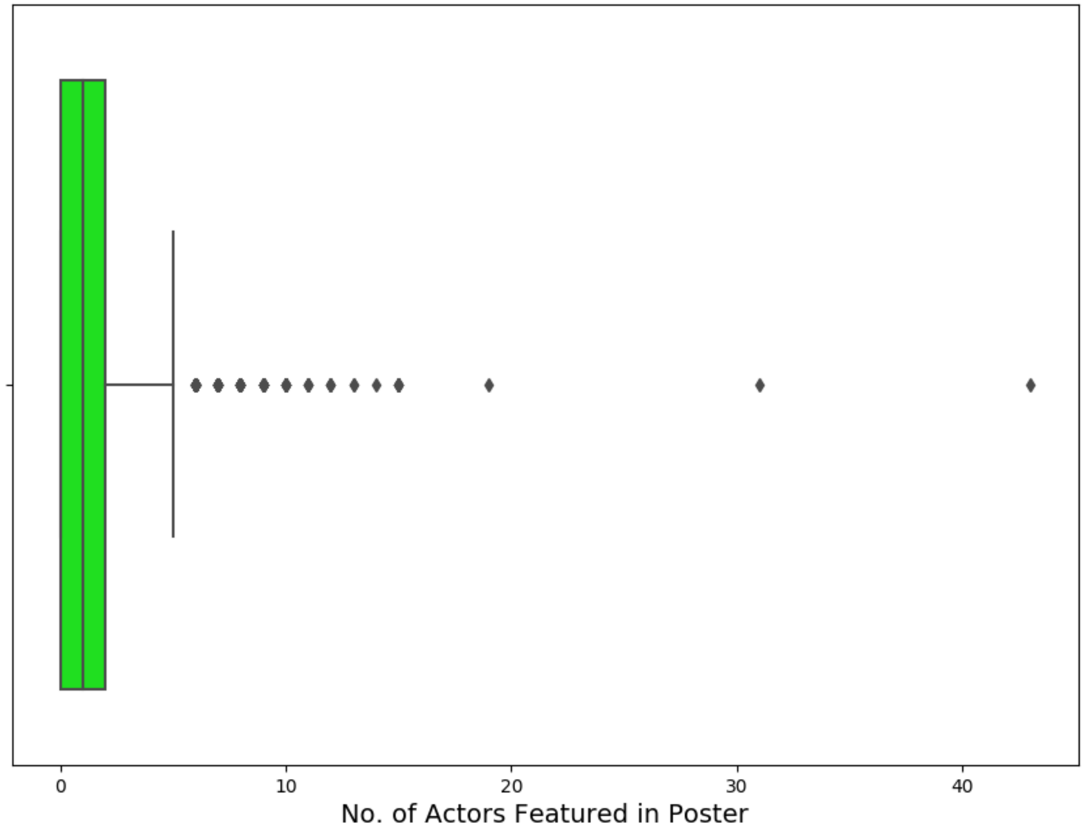

Figure 2: Too many outliers in number of faces featured in movie poster

Let’s check the movie with maximum number of actors (faces) that featured in the movie poster.

```
print movies_df[['movie_title', 'facenumber_in_poster']].iloc[movies_df['facenumber_in_poster'].idxmax()]
```

```
>>>
movie_title             500 Days of Summer
facenumber_in_poster                    43
```

So maximum number of faces (43) were featured in movie ‘500 Days of Summer’ . Let’s see a basic statistical details of this column ‘facenumber_in_poster’ with `pandas.DataFrame.describe()` method.

```
>>>
```

```
count    5030.000000
mean        1.371173
std         2.013576
min         0.000000
25%         0.000000
50%         1.000000
75%         2.000000
max        43.000000
```

With this, probably the box plot makes a lot more sense to you know.

Another way to detect outlier is to use Z Score. Let's see how that works.

Z Score and Outliers:

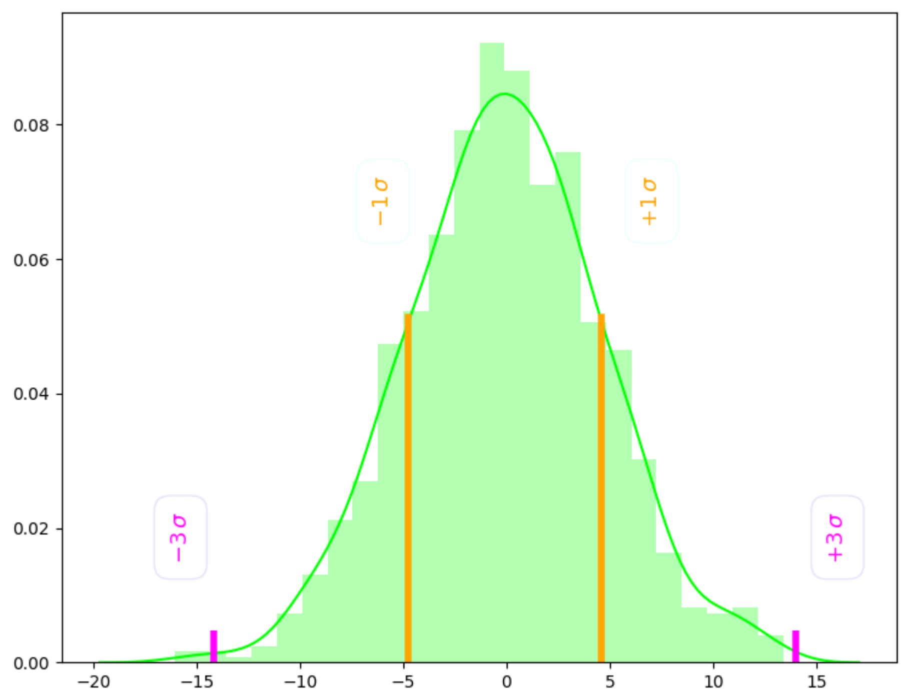

Figure 3: 1σ and 3σ Standard deviation on a normal distribution with 0 μ. (Source: Author)

Z score is a number (dimensionless) that signifies how much standard deviation a data point is, from the mean. **Z score simply can be defined as —**

Z =(X-μ)/σ, where μ is the population mean and σ is the standard deviation, X is one element in the population.

To plot the figure below, I have used normal distribution `numpy.random.normal()` and, in a normal distribution almost all the values — about 99.7%, fall within 3 σ deviation from the mean (for the plot here μ = 0). The way we can use Z score to reject outliers, is to consider the data points which are within 3 units of Z score. This can be done for all columns with 'non object' type data using `scipy.stats` as below.

1. Check the data types of all column in the data-frame **(DataFrame.dtypes).**

```
>>>
data types:
color                         object
director_name                 object
num_critic_for_reviews       float64
duration                     float64
director_facebook_likes      float64
actor_3_facebook_likes       float64
actor_2_name                  object
actor_1_facebook_likes       float64
gross                        float64
genres                        object
actor_1_name                  object
movie_title                   object
num_voted_users                int64
cast_total_facebook_likes      int64
actor_3_name                  object
facenumber_in_poster         float64
plot_keywords                 object
movie_imdb_link               object
num_user_for_reviews         float64
language                      object
country                       object
content_rating                object
budget                       float64
title_year                   float64
actor_2_facebook_likes       float64
imdb_score                   float64
aspect_ratio                 float64
movie_facebook_likes           int64
```

2. Create a new data-frame excluding all the 'object' types column **DataFrame.select_dtypes**

```
movies_df_num = movies_df.select_dtypes(exclude=['object'])

print "shape after excluding object columns: ", movies_df_num.shape

>>>

shape before : (3756, 28)
shape after excluding object columns:  (3756, 16)
```

3. Select elements from each column that lie within 3 units of Z score

```
movies_df_Zscore = movies_df_num[(np.abs(stats.zscore(movies_df_num))<3).all(axis=1)]

print "shape after rejecting outliers: ", movies_df_Zscore.shape
```

```
>>>
shape after rejecting outliers:  (3113, 16)
```

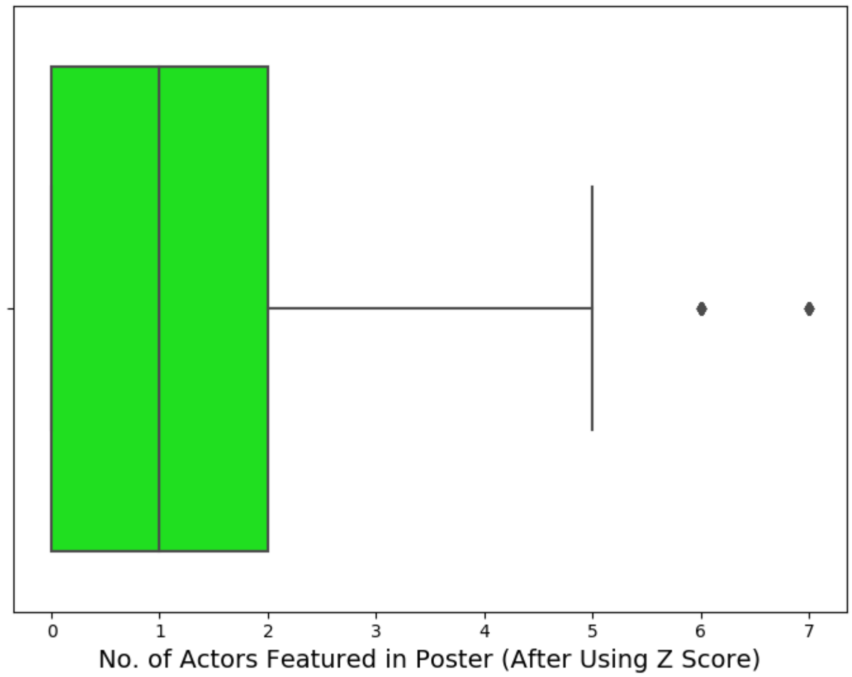

Figure 4: Box plot of number of faces featured in a movie poster. After applying the Z score method.

We can check the effect of the above steps by plotting again the box plot for ‘facenumber_in_poster’. Here one can see the difference compared to figure 2, where we had the box plot considering *all elements* in the ‘facenumber_in_poster’ column.

These are some ways one can prepare the data for analysis and applying machine learning algorithm for prediction. Effectively preparing the data-set can help a lot for comprehensive analysis and, I wish that this post will help you to prepare a data-set more methodically for further analysis. Depending upon the problem and data-set you may have to decide, choose and repeat these processes to interpret what are the effects, so, good luck exploring your data-set.

Stay Strong and Cheers !!

Codes used for this post are available on my Github.