PyPandas: A data cleaning toolkits for Spark

Pei-Lun Liao
New York University
pll273@nyu.edu

Chia-Hsien Lin
New York University
chl566@nyu.edu

Shang-Hung Tsai
New York University
st3127@nyu.edu

ABSTRACT

Data cleaning is a huge challenge in data mining and machine learning tasks. In this paper, we propose PyPandas, a data cleaning toolkits running on Spark, to provide further solutions to data cleaning tasks. PyPandas provides advance cleaning features such as outlier detection, data scaling and text pattern searching and replacement. We have released PyPandas on GitHub.

1 INTRODUCTION

In recent years, as storage devices become cheaper, storing big data in thousands of computers is easier than before. People start thinking about how to extract useful information from the huge datasets. The term, Big Data, was created to describe this phenomenon[18]. As the result, data mining[13, 22] and machine learning[26] get popular nowadays. The quality of data is one of the key factors to successful data mining[3]. Hence, data cleaning becomes a challenge in the first step of data management and analysis[5, 14, 23].

To process huge datasets efficiently, distributed systems and parallel computing frameworks[7, 9, 27] were introduced. Spark[28] is one of the most popular open source projects for industry and academia. It is built on Hadoop[27] and provides a way to manage distributed memory. PySpark[20] is an extended library for Python programmers to work with Spark.

In the Python machine learning and data mining ecosystem, Pandas[19] is the most popular data management library. Pandas dataframe provides convenient ways to collect data and several data management methods for missing value filling, data indexing and data profiling. In PySpark SQL module, PySpark provides Pandas-liked DataFrame to handle data indexing, data integration, and simple data cleaning tasks such as missing value handling and duplication removal.

However, PySpark DataFrame does not support advance data processing features like outlier detection and data scaling. Although there exists several clustering and scaling algorithms in the PySpark ML module, the integration with PySpark DataFrame is not helpful and can be difficult to use for data cleaning. User has to go back and forth to process their dataset.

In this paper, we propose PyPandas, a data cleaning toolkits built on top of PySpark, that integrates PySpark ML module to support user friendly data processing features such as outlier detection and numerical data scaling. These features are important for the machine learning tasks and are supported in Scikit-Learn[21], a mainstream machine learning library, as well.

Furthermore, text data is another common data type in our storage. Text data is hard to process because of different languages, newly invented words, typos, abbreviations, urls and emoji etc. Although we have NLTK[1], a popular text cleaning library, there is no intergation with PySpark DataFrame. In this paper, we also intergate NLTK within our toolkits and provide customized tools for users to clean text data easily.

The features of our library are summarized below.

- User friendly data cleaning toolkits built on PySpark
- Outlier detection and numerical data scaling
- Text processing such as url detection, punctuation removing, pattern searching and replacement.

2 RELATED WORK

There exists a number of open-source library for data cleaning and parallel data computing.

- Optimus[15] is a framework that can perform distributed data cleaning and preprocessing. This framework works well with Spark and its DataFrame can scale in big cluster. Optimus comes with helpful tools such as removing special characters and replacing null values.
- Dask[24] is another open-source library that supports parallel analytic computing. It provides scalable parallel data structures that extend interfaces like Pandas. At the same time, it offers low latency and high responsiveness.
- SparklingPandas[17] attempts to combine the power of Spark and Pandas to scale data analysis. It provides a Pandas-like API that is built using Spark's DataFrame class. Unfortunately, it only support spark v1.4 and Python 2.7, and its development has ended.

3 PROBLEM FORMULATION

In this paper, we focus on three major tasks, outlier detection, data scaling, and text pattern searching and replacement.

3.1 Outlier Detection

Outlier or anomalies detection is a challenging task in the field of data cleaning[4]. In this paper, we simply define outlier as the data that is far away from data centroid. Hence, given a dataset X, a clustering algorithm A[22], a distance measurement d, and the user-defined threshold t, the outlier is simply defined as following.

$$outlier = \{x | x \in X, d(x, x_{A_X(x)}) > t\}$$

where $A_X(x)$ assigns x into a cluster given the dataset X and x_c is the centroid of cluster c.

3.2 Scaling and Normalization

Scaling and normalization are common preprocess in data cleaning. It is crucial to several machine learning models and optimization methods. Scaling increases the convergence speed and chance to find good optimum[2]. The scaling problem could be formulated as below. Given an one dimensional numerical dataset X, we provide

https://github.com/shtsai7/PyPandas
Verison 0.2.1 for milestone report

a scaling method S such that scaled elements are bounded in same reasonable range $[a, b]$.

$$S_X(x) \in [a, b]$$

where $x \in X$ and $|a - b|$ should not be large.

To define the problem formally, we consider and implement several specific scaling methods.

- Standard scale

$$S(X) = \frac{X - \mu}{\sigma} \in \text{proper deviation range}$$

where μ is the mean of X and σ is the standard deviation of X

- Min-Max scale

$$S(X) = \frac{X - min(X)}{max(X) - min(X)} \in [0.0, 1.0]$$

- Max-Abs scale

$$S(X) = \frac{X}{max(X)} \in [-1.0, 1.0]$$

- P-norm Normalization

$$S(X) = \frac{X}{\|X\|_p} \in [-1.0, 1.0]$$

3.3 Pattern Searching and Replacement

Pattern searching and replacement is a common technique in cleaning text data. Given a string t and a regular expression[25] re, we replace the matched pattern with a user-defined string s. The replacement function R could be defined as following. Consider t as a concatenation of non-overlapping substrings $t = t_1 ... t_n$.

$$R(t) = r(t_1)...r(t_i)...r(t_n)$$

$$r(t) = \begin{cases} s & \text{if } M \text{ accept } t \\ t & \text{otherwise} \end{cases}$$

such that M does not accept any substring of $R(t)$

where M is the generalized nondeterministic finite automaton[25] corresponding to the regular expression re. We do not define replacement order and simply ignore it.

4 METHODS, ARCHITECTURE AND DESIGN

In this project, we build our library on top of PySpark. We handle PySpark DataFrame and implement useful data cleaning functions in our library. We integrate PySpark ML module and customize user define functions to perform advanced cleaning features such as pattern searching and replacement with regular expression. We survey the PySpark[20], Pandas[19] and Scikit-Learn[21] libraries to find out key features and create Pandas-like and Scikit-Learn-like API to provide easy usage and seamless adaptation.

4.1 Outlier Detection

4.1.1 Design. Our goal in outlier detection is to integrate the existing clustering algorithms in PySpark ML module to provide user-friendly solution for outlier detection. Users are able to select existing clustering algorithms. Furthermore, users are given the power to decide their own threshold value based on their knowledge about the dataset to filter outliers. To help users identify outliers and pick a proper threshold, we provide statistics summary of the clustering. Possible future work on outlier detection is to provide framework to register user defined clustering algorithms or implementing popular clustering algorithms such as DBScan[8] in our library.

4.1.2 Architecture. The implementation of the outlier detection functionality integrates the PySpark ml package. Our architecture hides the complexity of training and tuning clustering models, while it exposes easily accessible APIs for users to remove outliers. In addition, our data cleaning library implements User Defined Functions to compute distance between data points and cluster centers, as well as other summary statistics. At this time, our current remover can handle outliers with KMeans clustering algorithm. We will refactor our architecture to integrate all existing clustering algorithms in PySpark ML module. To simplify our architecture, we will use Factory Design Pattern to construct different types of outlier removers with the generic APIs.

4.1.3 Methods.

- KMeans[16]: Users can specify the number of clusters to use based their knowledge about the dataset. The system will choose random initial values, and perform 20 iterations of updates. Finally, a summary of the clusterings will be generated, which includes cluster centers, cluster sizes, distances, etc. User can then filter out outliers in each cluster by specifying a minimum distance from the cluster center, and all data points that are beyond that distance will be removed from the dataset.

4.2 Scaling and Normalization

4.2.1 Design. Standardization is a common requirement for many machine learning algorithms, since some optimization methods will not work properly without scaling or normalization[2]. Our goal in scaling and normalization is similar to the goal of outlier detection. We will intergrate existing scaling algorithms in PySpark ML module into our library. User can choose existing scaling algorithms to scale their datasets. The future works is to provide framework to register user defined scaling algorithms.

4.2.2 Architecture. Since the design is similar with the design of outlier detection, we will follow the architecture of outlier detection. We will use the Factory Design Pattern to allow users construct their desired scaling models. We are considering to implement a single interface between any existing models in PySpark ML module to DataFrame in PySpark SQL module. Hence, we will be able to provide any feature transformation with the existing machine learning models in PySpark ML module.

4.2.3 Methods.

Table 1: Common Regular Expression Pattern

Common Pattern	Regular Expression
URL	URL regular expression[6]
Leading space	`'^ +'`
Trailing space	`' +$'`
Consecutive space	`' +'`
Number	`'\d+'`
Not a word	`'[^\w\d\s]+'`
Blank	`'_+'`

Table 2: Selected dataset

	(Row, Col)	Size	Missing Value
311 Service Requests	(9M, 53)	6.01 GB	28%
Permit Issuance	(3M, 60)	1.43 GB	20%
Job Application Filings	(5M, 89)	2.88 GB	40%

- Standard Scale: Performs basic scaling on a particular column or a list of columns in a dataframe so that the values have unit standard deviation and zero mean.
- Min-max Scale: Rescales columns to a user defined range(e.g. [0, 1]) using min max scaling.
- Max-abs Scale: Transforms columns to range between -1 and 1 by dividing through the maximum absolute value in the columns. This operation does not shift/center the data, and thus does not destroy any sparsity.
- Normalization: Given a list of columns, the normalize function can generate a normalized feature vector having unit norm. The default p-norm value for normalization is 2.0, yet users can optionally specify the p-norm value.

4.3 Pattern Searching and Replacement

4.3.1 Design. To provide a useful text cleaning toolkits, we need to have the flexibility for user to search any pattern and replace the matched ones with any user defined string. At the same time, we have to provide common patterns so that users won't need to reinvent the wheel. For example, url pattern, leading space, trailing space or numbers are common search patterns that we should provide. Moreover, we would like to integrate NLTK[1], a popular natural language toolkit, into our library. With NLTK, we would be able to perform advanced cleaning tasks or feature transformation.

4.3.2 Architecture. In order to achieve those design issues, we extract the common logic and column feature transformation into our core functions. We design the architecture such that user defined functions can be applied on a single or multiple columns, and a new DataFrame will be provided. The user defined function could be a regular expression substitution function, a column scaling function or a natural language processing function in NLTK. Our architecture ensures the flexibility to various customized processing methods.

4.3.3 Methods. We define several common regular expression patterns in table 1 and provide simple cleaning functions to handle those patterns in text data.

5 EXPERIMENT

5.1 Dataset

Three datasets are chosen for the experiments. Each of them has different properties and can be used to test different parts of the library. The summary of datasets could be found in table 2.

- 311 Service Requests[10] This dataset contains all 311 Service Requests since 2010. The majority of data has string type, including both categorical data (e.g. City, Status) and variable length data (e.g. Address, Description). This can be useful in testing the string-related features of the library.
- DOB Permit Issuance[11] The dataset consists of a list of permits for buildings in NYC issued by Department of Building. It has diverse data types, including categorical, string, numerical, and geographical data. Hence, it provides a good test case for the versatility of the library.
- DOB Job Application Filings[12] The dataset stores job applications filed through the Borough Offices in NYC. There are many columns containing numerical data, such as fee and number of stories. It can be used to test the data scaling and outlier detection features.

5.2 Evaluation

We will evaluate our project by comparing it against other existing open-source libraries, including Optimus[15], Dask[24], and SparkingPandas[17].

5.2.1 Features. We will compare our data cleaning features with previous works, particularly those frequently used functionality in data science such as null value handling, special character removal, duplicate detection, etc.

5.2.2 Performance. We will evaluate the performance of our library by measuring the time and space consumed and drawing comparison with existing libraries. This performance evaluation will show that this library is efficient and can be useful in production.

5.2.3 Usability. We might assess the usability of our library by doing a survey among a random group of users. We will compare the installation process and APIs provided and generate scores of user experience.

5.3 Preliminary Results

At current stage, our experiments were run on the Dumbo clusters of NYU. We plan to migrate our settings to AWS since we will have the privileges needed to install other packages there. While on Dumbo, we don't have such privileges. We will perform comparison with other related works after we finish migration.

5.3.1 Features. In this section, we compare our features with previous works. The result is shown in table 3. We find that PySpark DataFrame provide easy-to-use data manipulation functions. Since PyPandas and Optimus extend the DataFrame, both support the same data manipulation features. In SparkingPandas, the project only aims to provide grouping methods on Pandas-liked dataframe and miss other important features in data manipulation. Dask also support many features in data manipulation.

Table 3: Evaluation: Features Comparison

	PySpark DataFrame[20]	PyPandas	Optimus[15]	SparkingPandas[17]	Dask[24]
Data manipulation					
Data integration	✓	✓	✓		✓
Data aggregation	✓	✓	✓	✓	✓
Data filtering	✓	✓	✓		✓
Data profiling	✓	✓	✓	✓	✓
Data sorting	✓	✓	✓		
Basic data cleaning					
Missing value filling	✓	✓	✓		✓
Missing value removing	✓	✓	✓		✓
Duplication removing	✓	✓	✓		✓
Outlier Detection					
Deviation with median			✓		
Clustering algorithms		✓			
Scaling and Normalization					
Standard scaling		✓			✓
Scale in range		✓			✓
Normalization		✓			
Text Cleaning					
Special character removing	✓	✓	✓		
Pattern searching	✓	✓	✓		
Pattern searching and replacement		✓			

PySpark DataFrame, Optimus, Dask, and our library all provide the basic data cleaning functions like missing value handling. In the outlier detection, Optimus detects outliers by calculating the deviation with the median of data. On the other hand, PyPandas provides advance detecting methods with clustering algorithms. Dask do support several clustering algorithms. Nonetheless, they are not integrated in their Dask DataFrame. Users have to train and tune the models to detect the outliers.

In scaling and normalization, Dask and PyPandas provide basic scaling methods like standard scaling or min-max scaling. However, Dask does not provide normalization on their Dask ML preprocessing module.

For the text cleaning, PyPandas provides more powerful pattern searching and replacement feature to let user find out any pattern and replace the matched one with their desired value. Moreover, PyPandas defines several common patterns for users to clean their text quickly without repeating themselves. In summary, PyPandas extends PySpark DataFrame and provides more easy-to-use features for users to clean their data.

5.3.2 Performance. We evaluate the performance of different parts of our library. The total of execution times of each task, including the time for Spark initialization, data loading, and data preprocessing, were all recorded and listed below.

- Outlier Detection Experiment: This experiment is performed on the DOB Job Application Filings Dataset. 15 columns that contain numerical values are chosen. We use K-Means Outlier Remover with three different k values [1, 2, 3]. For each k value setting, we run the clustering with different number of attributes (dimensions). The results are shown in table 4.
- Scaling and Normalization Experiment: This experiment involves all three testing datasets. We choose three columns with numerical values from each dataset as input. For each input, we test all scaling and normalization operations. The experiment results can be found in table 5.

Table 4: Outlier Detection Experiment Result

Dimension	k = 3	k = 4	k = 5
1	1 m 15.15 s	1 m 6.48 s	1 m 6.27 s
3	1 m 7.13 s	1 m 4.97 s	1 m 5.52 s
5	1 m 5.94 s	1 m 5.30 s	1 m 6.82 s
10	1 m 7.64 s	1 m 12.36 s	1 m 11.18 s
15	1 m 7.57 s	1 m 9.12 s	1 m 7.92 s

6 CONCLUSIONS

Due to the lack of advance data cleaning features on Spark, we propose PyPandas, a data cleaning toolkit built on PySpark. We provide advance data cleaning features such as outlier detection, data scaling, and text pattern searching and replacement.

Table 5: Scaling and Normalization Experiment Result

	311 Service Requests	Permit Issuance	Job Application Filings
Standard Scale	3 m 8.15 s	1 m 25.57 s	1 m 19.96 s
Min-max Scale	3 m 7.15 s	1 m 25.95 s	1 m 20.07 s
Max-abs Scale	3 m 7.15 s	1 m 24.48 s	1 m 19.90 s
Normalization	1 m 18.95 s	38.41 s	39.48 s

REFERENCES

[1] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python* (1st ed.). O'Reilly Media, Inc.

[2] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, New York, NY, USA.

[3] Li Cai and Yangyong Zhu. 2015. The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. *Data Science Journal* 14, 2 (2015), 1–10. https://doi.org/10.5334/dsj-2015-002

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.* 41, 3, Article 15 (July 2009), 58 pages. https://doi.org/10.1145/1541880.1541882

[5] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data Cleaning: Overview and Emerging Challenges. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 2201–2206. https://doi.org/10.1145/2882903.2912574

[6] Dans. [n. d.]. Regular Exp Url. ([n. d.]). https://www.regextester.com/53716

[7] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 226–231. http://dl.acm.org/citation.cfm?id=3001460.3001507

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*. ACM, New York, NY, USA, 29–43. https://doi.org/10.1145/945445.945450

[10] NYC government. 2018. NYC OpenData. (2018). https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9

[11] NYC government. 2018. NYC OpenData. (2018). https://data.cityofnewyork.us/Housing-Development/DOB-Permit-Issuance/ipu4-2q9a/data

[12] NYC government. 2018. NYC OpenData. (2018). https://data.cityofnewyork.us/Housing-Development/DOB-Job-Application-Filings/ic3t-wcy2

[13] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[14] Ihab F. Ilyas and Xu Chu. 2015. Trends in Cleaning Relational Data: Consistency and Deduplication. *Foundations and TrendsÂő in Databases* 5, 4 (2015), 281–393. https://doi.org/10.1561/1900000045

[15] Iron. 2018. Optimus. (2018). https://github.com/ironmussa/Optimus

[16] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. 2002. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 7 (July 2002), 881–892. https://doi.org/10.1109/TPAMI.2002.1017616

[17] Holden Karau and Juliet Hougland. 2015. SparklingPandas. (2015). https://github.com/sparklingpandas/sparklingpandas

[18] A. Katal, M. Wazid, and R. H. Goudar. 2013. Big data: Issues, challenges, tools and Good practices. In *2013 Sixth International Conference on Contemporary Computing (IC3)*. 404–409. https://doi.org/10.1109/IC3.2013.6612229

[19] Wes McKinney. [n. d.]. pandas: a Foundational Python Library for Data Analysis and Statistics. ([n. d.]).

[20] Amit Nandi. 2015. *Spark for Python Developers*. Packt Publishing.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[22] Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.

[23] Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 381–390. http://dl.acm.org/citation.cfm?id=645927.672045

[24] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Proceedings of the 14th Python in Science Conference*, Kathryn Huff and James Bergstra (Eds.). 130 – 136.

[25] Michael Sipser. 1996. *Introduction to the Theory of Computation* (1st ed.). International Thomson Publishing.

[26] Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

[27] Tom White. 2009. *Hadoop: The Definitive Guide* (1st ed.). O'Reilly Media, Inc.

[28] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1863103.1863113