Towards Understanding Data Analysis Workflows using a Large Notebook Corpus

Mohammed Suhail Rehman
University of Chicago
suhail@uchicago.edu

KEYWORDS

computational notebooks; data workflows; lineage extraction

ACM Reference Format:
Mohammed Suhail Rehman. 2019. Towards Understanding Data Analysis Workflows using a Large Notebook Corpus. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3299869.3300107

1 INTRODUCTION AND MOTIVATION

The advent of big data analysis as a profession as well as a hobby has brought an increase in novel forms of data exploration and analysis, particularly ad-hoc analysis. Analysis of raw datasets using frameworks such as `pandas` and `R` have become very popular[8]. Typically these types of workflows are geared towards ingesting and transforming data in an exploratory fashion in order to derive knowledge while minimizing time-to-insight. However, there exists very little work studying usability and performance concerns of such unstructured workflows.

As a starting point, we found that computational notebooks such as Jupyter[5] (formerly `IPython Notebook`) have become an increasingly favored medium to combine prose, code, and visualizations to document and share data science workflows. In our work, we use a recently published notebook corpus[8] to find usage trends within the popular python data analysis framework, `pandas`[6]. As a result, we are able to construct lineage graphs of pandas notebook workflows, and show some general trends in data analysis within these notebooks. We envision the use of this corpus and our lineage inferencing technique to allow for a myriad of uses, ranging from on-the-fly optimization of data analysis workflows, to the use of common design patterns to synthetically generate workflows for future work.

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06.
https://doi.org/10.1145/3299869.3300107

2 BACKGROUND AND RELATED WORK

The use of computational notebooks have been surveyed in [2, 4, 8]. Provenance tracking in notebooks is explored in a number of works [1, 7, 9], all of which rely on the execution of the notebook with associated data, which is cumbersome or even impossible for retrospective lineage extraction on a large corpus. [3] has explored a novel version system that improves the usability of notebooks given it's ad-hoc and exploratory nature, but is also intended to be used in an online manner.

Our work is based an a recent paper by Rule *et al.*[8]. Their resulting corpus of over 1 million Jupyter[5] notebooks will be the focus of our study, as we explore the specific use case of notebooks that contain data analyses using the popular python library `pandas`[6] and generate lineage graphs for a subset of notebooks in this corpus.

3 APPROACH

The dataset that we are using for our analysis consists of approximately 1.25 million Jupyter Notebooks crawled from Github in July 2017. In order to isolate the notebooks relevant to our study, we filtered this corpus for all notebooks that contain an import of the pandas library. For this paper, we were able to isolate a sample of $\sim 50K$ notebook files that use the `pandas` library, using a regular expression search for the library import statement.

Each Jupyter notebook file is encoded as a JSON file and typically contains separate code cells (further divided into source and output cells), as well as markdown cells, which may contain documentation and/or commentary for the notebook. We were able to isolate the source code cells and parse Python code strings using the Python's `ast` library. Specifically, we looked for invocations of pandas functions using the alias name, and variable assignments of such invocations. The resulting variables are typically pandas `DataFrame` objects, from which specific transformation methods can be tracked. Thus for each notebook, a directed graph can be constructed where the vertices are the various objects such as

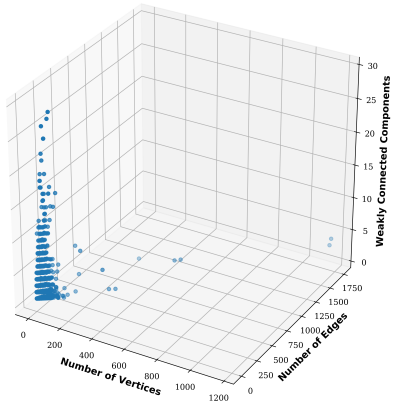

Figure 1: Number of vertices, edges and components from all the extracted graphs

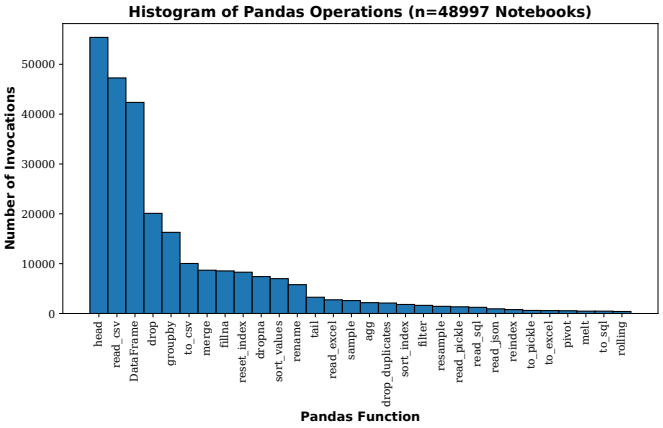

Figure 2: Histogram of pandas function invocations

variable names, data sources or sinks and the edges represent various transformations applied by pandas methods.

Given the wide range of transformation primitives and the duck-typed nature of Python, we focused on simple variable assignments and function calls. Expressions containing chained function invocations and/or Python's slice notation are currently beyond the scope of this paper and are planned for future investigation.

After extracting the lineages from the sample of Jupyter notebooks, we ran simple algorithms to analyze the resulting graphs, specifically targeting the number of weakly connected components, determining if the graph is a directed acyclic graph (DAG), and finding the longest path in each DAG. The results are presented in the next section.

4 RESULTS

From the sampled notebooks (N=48,975), we were able to recover 27,349 partial lineage graphs using our static inference methods for 44,012 notebooks (4963 notebooks were skipped due to errors in parsing and inference and graphs with less than one edge or vertex were filtered out). The average time to infer lineage is roughly 0.018 seconds on a desktop machine per notebook, allowing for this type of processing to be done online and interactively. An example extracted lineage graph is presented in Appendix A.

Most inferred graphs are small, as plotted in Figure 1, and generally have a few number of connected components. A few outlier graphs were observed, with the largest graph containing 1171 vertices and 1749 edges, from a notebook that was analyzing footballer stats from 585 individual CSV files.

We also plot the most common transformation functions in Figure 2. We find the most common operation to be `head`, unsurprisingly, since users likely want to display the first few rows of a dataset before continuing the data workflow. We also find that users are most likely to import data via csv (`read_csv`) and perform grouping (`groupby`) primitives as indicated in the histogram, as well as sorting (`sort_values`) and null-value cleaning (`fillna`) primitives.

The number of acyclic and cyclic graphs were 10,710 and 16,639 respectively. Cycles generally appeared in the graphs due to some form of variable reuse within the notebook. We also found that long chains of transformations are not typical; the longest chain of transformations we found was 14, with 97% of all chains being of size 3 or below.

5 DISCUSSION AND FUTURE WORK

This work presents a promising start to understanding data analysis workflows and automatic lineage inference / reconstruction through static analysis of computational notebooks. We plan to extend the inferencing technique to capture all types of pandas operations, and not just simple variable assignments and function invocations.

Given the performance of our extraction technique, we can foresee its use in interactive contexts to provide helpful hints and suggestions in optimizing runtime performance and improve space/memory usage within the Jupyter environment. As an example, a common anti-pattern such as performing a selection after a join could be detected in an interactive notebook session, and the system could automatically provide a hint to the user to invert the order or operations for better performance.

We also envision being able to fashion a random data workflow generator using the statistics inferred from the generated lineage graphs. This would be useful for benchmarking purposes and generate a synthetic workloads for performance evaluation and experiments in data analysis workflow lineage.

REFERENCES

[1] Lucas AMC Carvalho, Regina Wang, Yolanda Gil, and Daniel Garijo. 2017. NiW: Converting Notebooks into Workflows to Capture Dataflow and Provenance. In Proceedings of Workshops and Tutorials of the 9th International Conference on Knowledge Capture (K-CAP2017).

[2] Project Jupyter. 2015. Jupyter Notebook UX Survey. https://github.com/jupyter/surveys/tree/master/surveys/2015-12-notebook-ux.

[3] M. B. Kery and B. A. Myers. 2018. Interactions for Untangling Messy History in a Computational Notebook. In 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). 147–155. https://doi.org/10.1109/VLHCC.2018.8506576

[4] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E John, and Brad A Myers. 2018. The Story in the Notebook: Exploratory Data Science using a Literate Programming Tool. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. ACM, 174.

[5] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks-a publishing format for reproducible computational workflows.. In ELPUB. 87–90.

[6] Wes McKinney. 2012. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. " O'Reilly Media, Inc.".

[7] João Felipe Nicolaci Pimentel, Vanessa Braganholo, Leonardo Murta, and Juliana Freire. 2015. Collecting and analyzing provenance on interactive notebooks: when IPython meets noWorkflow. In Workshop on the Theory and Practice of Provenance (TaPP), Edinburgh, Scotland. 155–167.

[8] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18). ACM, New York, NY, USA, Article 32, 12 pages. https://doi.org/10.1145/3173574.3173606

[9] Sheeba Samuel and Birgitta König-Ries. 2018. ProvBook: Provenance-based semantic enrichment of interactive notebooks for reproducibility. In Proceedings of the ISWC.

A SAMPLE NOTEBOOK CODE AND EXTRACTED LINEAGE

Here we present a code sample from the corpus (Listing 1), as well as a visualization of the output graph created using our lineage extraction technique (Figure 3).

Listing 1: Code listing from one of the notebooks in the corpus

```
import numpy as np
import pandas as pd

%matplotlib inline

import matplotlib.pyplot as plt
data = pd.read_csv('trades/2011-07.csv')
price_usd = pd.DataFrame(data["Money"]/data["Bitcoins"], \
          columns=['Price_USD'])
data = data.join(price_usd)

import datetime
data["Date"] = pd.to_datetime(data.Date)
data.head()
plt.rcParams['figure.figsize'] = (19.0, 12.0)

K = 20
grouped = data[data.Type == 'buy'].groupby('User_Id')
btc_vol = grouped.sum()['Bitcoins']

sorted_buyers = btc_vol.sort_values(ascending=False,inplace=False)
top_buyers = sorted_buyers[:K]
buyers = np.array(top_buyers.index)
grouped = data[data.Type == 'sell'].groupby('User_Id')
btc_vol = grouped.sum()['Bitcoins']

sorted_sellers = btc_vol.sort_values(ascending=False,inplace=False)
top_sellers = sorted_sellers[:K]
sellers = np.array(top_sellers.index)
unique_users = np.unique(data["User_Id"])
n_users = len(unique_users)
buy_sell_mat = np.zeros((n_users,n_users))


def user2index(user_id):
    return np.where(unique_users == \
           user_id.reshape(np.array(user_id).shape[0],1))

matrix_index = pd.DataFrame(user2index(data["User_Id"])[1], \
             columns=['Matrix index'])

data = data.join(matrix_index)
trades_grouped = data.groupby('Trade_Id')
```

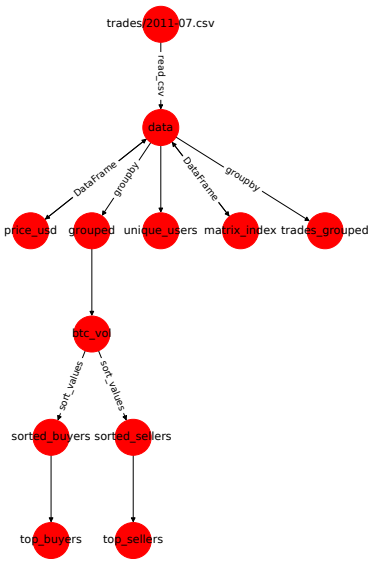

Figure 3: Lineage extracted from the example code in Listing 1