Jupyter Mining Summary

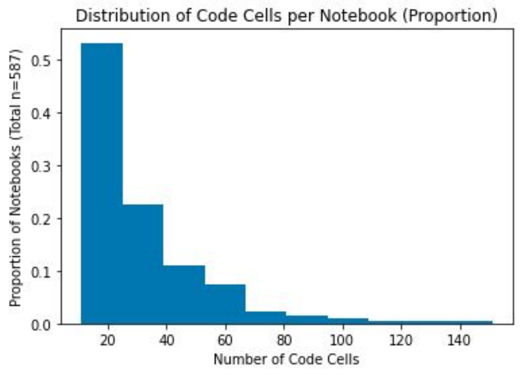
Objective

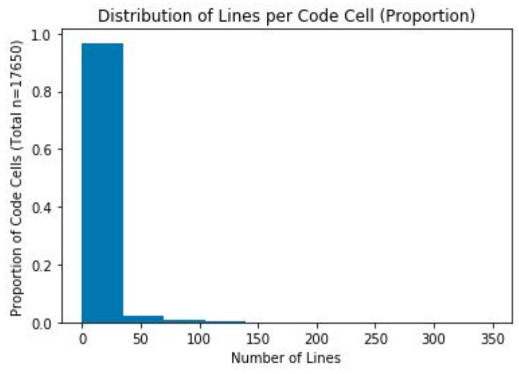
This project aims to gain insight as to how users conduct data science by studying Rule's Corpus of ~1.25 million notebooks. In particular, we hope to use code analysis and notebook inspection to understand the patterns and strategies that users adopt in performing data science tasks. Example research questions include: do people spend more effort in performing data cleaning over visualizations? How do users move from one step of the data science task to the next?

Filtering Notebooks and Preliminary Analysis

To begin, we worked with only the sample size of 6529 notebooks from 1000 different repositories. Even this sample set was noisy with incomplete notebooks and notebooks with other languages, so we filtered the sample set. Besides ensuring the language of the notebook was python and that there were at least 10 code cells to tackle the problem of incompleteness, we also filtered out notebooks containing terms such as "homework", "lecture", "assignment", and "introduction to", since these notebooks are often tutorials or intended for instruction, and therefore are less likely to emulate *real-world* data analysis workflows. Finally, we enforced that each notebook must import both pandas and matplotlib, because these were two packages we decided data science notebooks would use and these were the packages whose functions we classified into different steps of data science. To filter, we search for the appropriate strings in the text of the code cells of the notebook.

Typically, ipython notebooks files are stored in a json format which has a "cells" entry, and of the 6529 notebooks this is true of 5442 notebooks which we deemed analyzable. After applying the rest of the criteria, we were left with 584 notebooks. Out of curiosity, we also looked at the number of notebooks which were executed with a strictly increasing execution count, and we discovered there to be only 91 of the 584 notebooks that follow these criteria. We also investigated the distribution of code cells across different notebooks and well as the distribution of lines across the different cells.





Categorizing Functions

Category Definitions

To understand the semantic functions of the code in the notebook, we devised a set of terminology to classify the data science task. We used the most popular pandas and scikit learn functions to help us determine this. From this article, a look over pandas documentation, and a manual inspection of the notebooks, we determined what pandas functions were frequently used. Afterwards, based on what we believed the user was generally trying to achieve by calling a function, we classified these functions into the following categories:

- createFns = ["read_csv", "DataFrame", "Series", "copy"]
- cleaningFns = ["isnull", "drop", "fill", "replace", "rename", "astype", "set_index", "loc","iloc", "index", "reset_index","astype", "query"]
- printFns = ["head", "tail", "shape", "info", "describe", "value", "columns", "print"]
- plotFns=["plot", "plotting"]
- groupFns = ["group", "apply", "sort", "pivot"]
- joinFns= ["append", "concat", "join"]
- statsFns = ["describe", "mean", "corr", "max", "min", "median", "std", "sum"]

We also looked through all scikit learn documentation regarding their subpackages and categorized them as follows:

- preprocessingFns = ['preprocessing', 'impute', 'feature', 'decomposition', 'discriminant_analysis', 'random_projection', 'sampling', 'compose']
- modelFns = ['classifier', 'cluster', 'clf', 'estimator', 'naive_bayes', 'svm', 'neural_network', 'ensemble', 'dummy', 'gaussian_process', 'kernel', 'manifold', 'mixture', 'multiclass', 'multioutput', 'neighbors', 'semi_supervised', 'tree']
- postprocessingFns = ['covariance', 'metrics', 'inspection', 'model_selection']

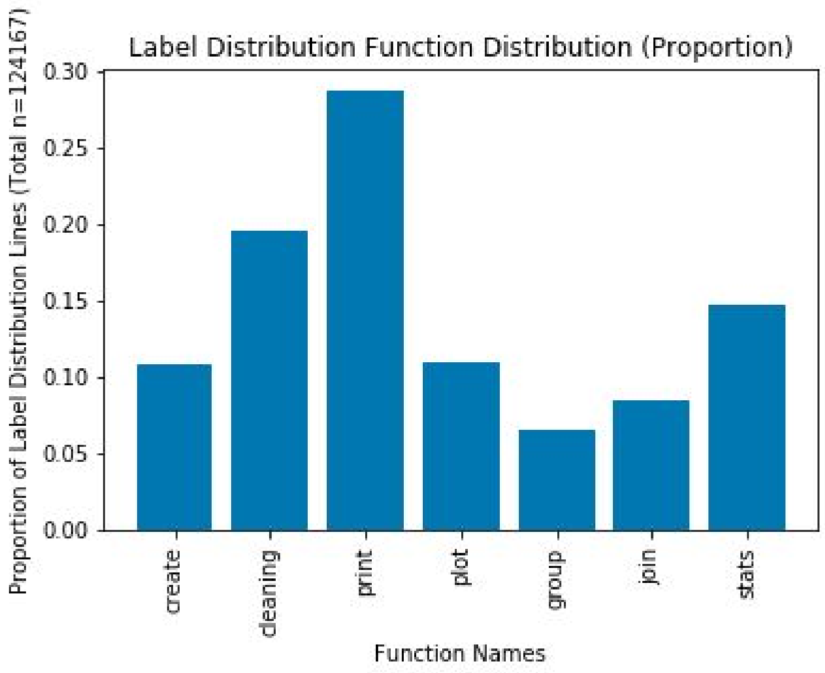
We also parsed the matplotlib import and used the import alias to search for instances of matplotlib functions used. We categorized all matplotlib functions as ‘plot’.

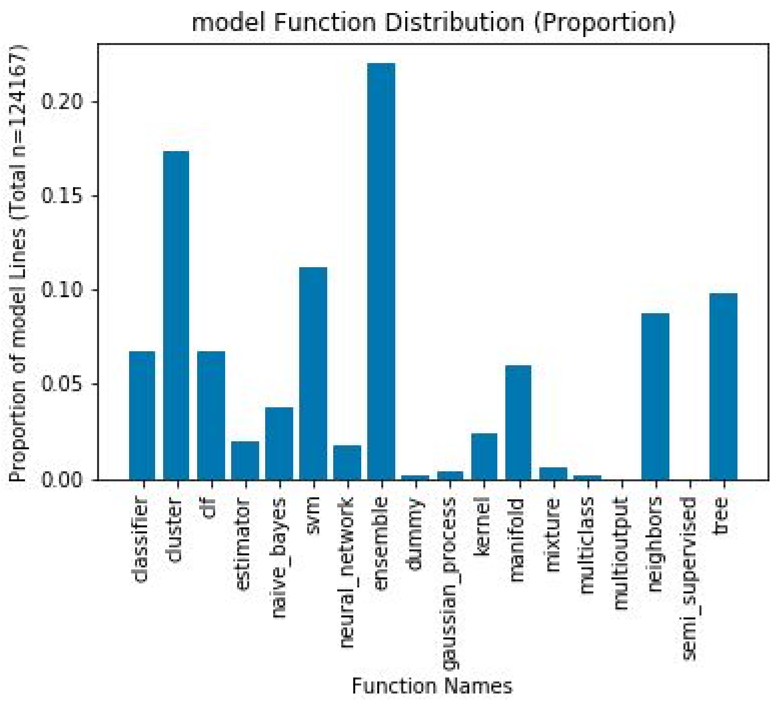
Parsing Methodology

The process of parsing the code for the presence of the categories involved searching the text of a line of code for “.” prepending the function name or “(“ appended after the function name. If a line had more than one occurrence of the functions we were searching for, the raw count would be increased for each function that appeared but the line overall would only be categorized by the last category function that appeared. For example, a line reading `df.loc[:,:].head()` contains both `loc` which is part of the cleaning category and `head` which is part of the printing category. The counts for both those categories would increase by one, but the line overall would be classified as doing printing.

Frequency Visualizations

We compared the frequency across categories but also across functions within a category.





Graphing Time Series

Introduction of Macro-Categories

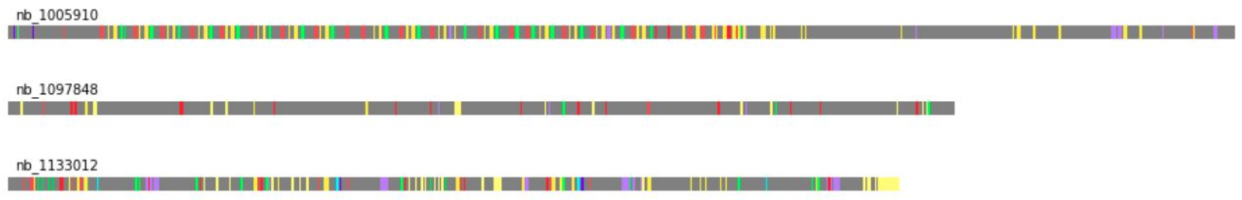
We wanted to explore how users move from one category to another throughout the notebook, so we wished to graph a time series in which colors would correspond to certain categories. However, we had too many categories to manage, so we decided to group the categories together based on what we determined their functionality and used a similar color scheme when plotting lines corresponding to those categories. We group categories and assigned colors as follows:

- Ingestion (reds): join, create
- Inspect (yellows): print, plot
- Wrangling (greens): group, cleaning, preprocessing
- Mode (blues): model
- Compute (purples): post processing, stats
- Other (grey)

Time Series Examples and Observations

Below are some examples of the notebook time series graphed. We'd like to highlight a few points here that presented some challenges later.

1. First, the notebooks are of vastly different length and the time series captures that assigning one sliver for each line, colored depending on what the line was classified as doing. Therefore, the lengths of the graphs are proportional and if the bar is twice as long, that means the corresponding notebook has twice as many lines.
2. Second, as we see with the first notebook, there are repetitions of patterns that occur over and over again in a single notebook. Upon closer examination, it appears that these sections are copy and pasted with slight modifications/variable changes between the blocks of cells.
3. The majority of the graph is grey, meaning that the function was classified as 'other.' Upon examining those portions of the notebook, we discovered that those lines mostly fall under one of two categories which we'll call untracked packages and python syntax/declarations (if statements, loops, var definitions).



Clustering Notebooks

Purpose and Methodology

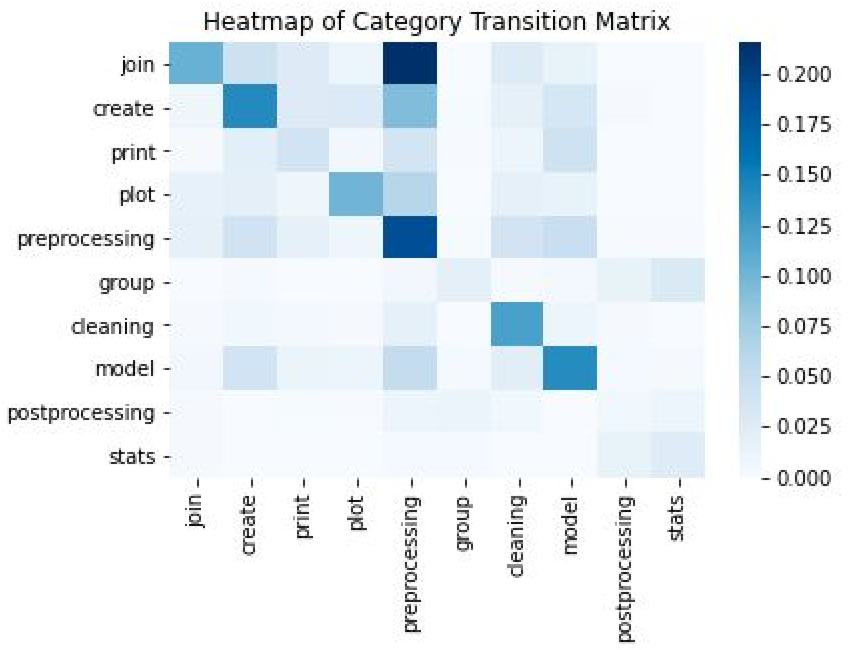
We decided that even the time series of the sample set of 500+ notebooks was too overwhelming to look all at once. Therefore, we decided to cluster the notebooks as a preliminary step to group together similar notebooks and understand salient patterns and features in the dataset. We used agglomerative clustering with a euclidean distance metric. The main issue we encountered here was that the notebooks are all of different sizes. Normalization of length by upscaling all the notebooks to the maximum length led to slow computation times, and we were concerned that downscaling certain notebooks would lead to rounding issues and loss of information as most categories only exist in one line chunks.

Features Used

When considering features, we did not account for the 'other' category since doing so would overwhelm many of our features. The 'other' category is too prominent and not meaningful enough to be included. Instead, for each of the notebooks we computed to following information that resulted in an array of length 202:

- We computed the transition probabilities between categories excluding 'other'. We set up a transition matrix similar to a Markov transition matrix and treated each of the categories as states. We then parsed through the series, calculated the corresponding probabilities, and used the flattened 2D matrix as our features.

    - Note that unlike Markov chains, rows in our matrix do not always sum to 1, for if a category does not appear in a particular notebooks, that corresponding row and column both sum to 0. Furthermore, by discounting the ‘other’ category, two categories that have a high probability of transition may not necessarily have corresponding lines that are adjacent (there could be ‘other’ lines intermixed).

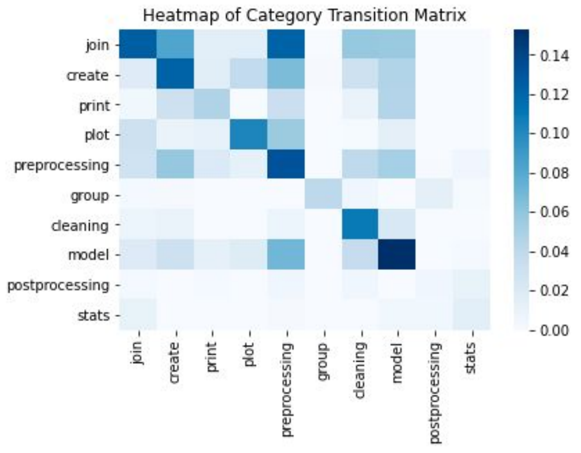


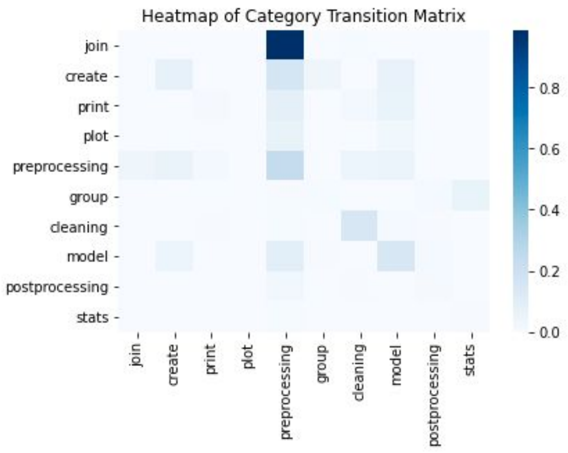
- We computed both raw counts and total proportion of lines for each category excluding ‘other.’
- We repeated both of the above with our macro-categories.
- Using Rule’s analysis we determined the top packages imported ('numpy', 'matplotlib', 'pandas', 'sklearn', 'os', 'scipy', 'seaborn') and included the boolean array of whether those packages were included in the notebook.

Examining Clusters

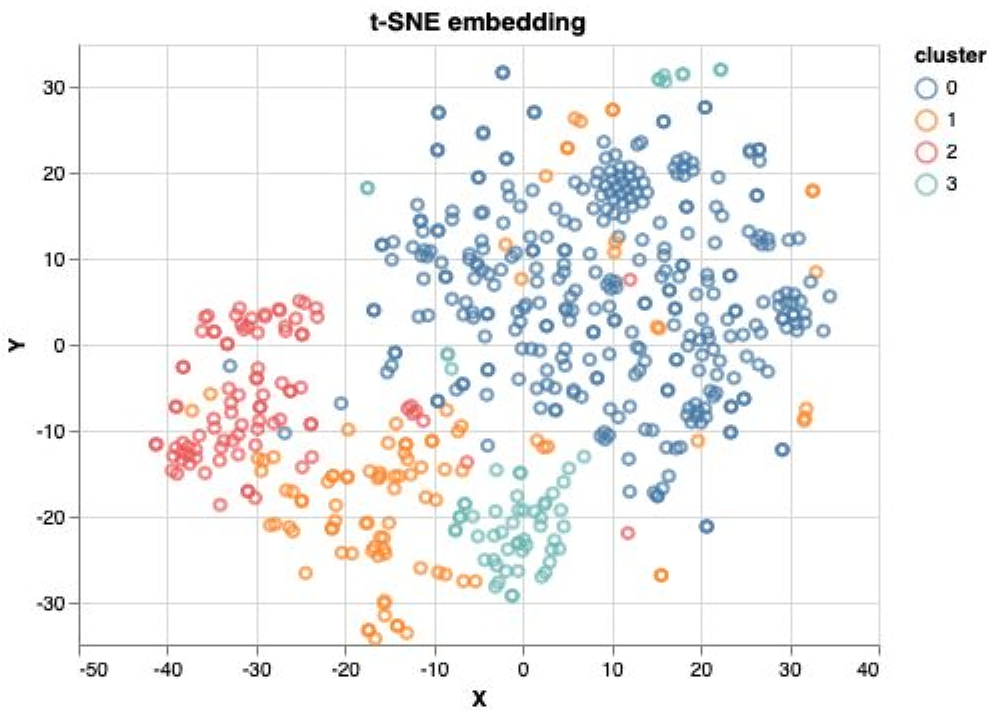
We implored a few different strategies to try and examine the clusters a bit more closely. First, we looked at the time series graphs for each cluster. While some clusters seemed to have more of certain color appear, there were still no discernible patterns between the different clusters by inspecting the time series.

We then looked towards the transition matrices for each individual cluster where there were interesting differences to be found. Below are the comparisons of clusters 1 and 2 and here, we see the effect of how the higher frequency of certain functions within clusters drastically changes how the transition probabilities appear.



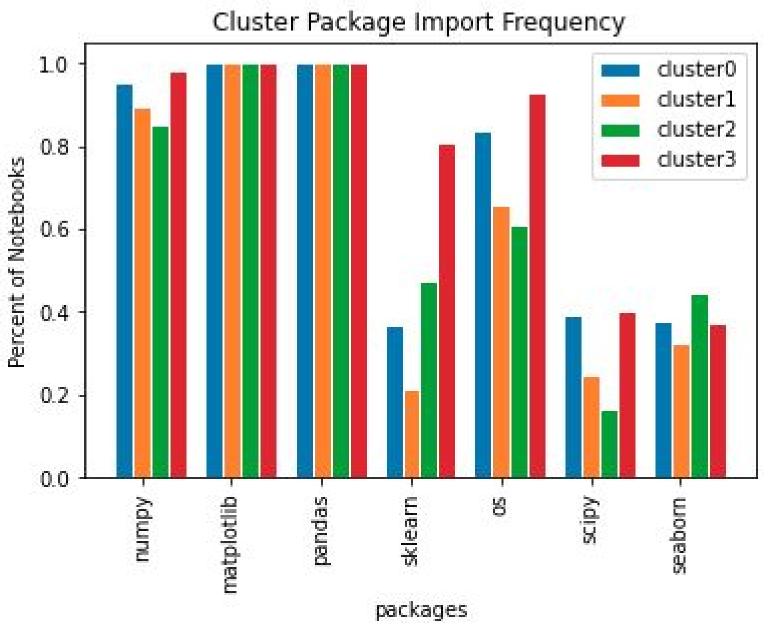


We also looked at the 2D projections of the clusters, and this has by far been the most helpful in providing some sort of confidence that our clusters capture some of the correct information. For starters, notebooks that are particularly tightly clustered are actually different variations of the same notebook. Furthermore, in cluster 1, there does seem to be some pattern of data science notebooks that are doing data wrangling as opposed to the other clusters which have much higher proportions of machine learning. An ongoing exercise is to examine what makes certain notebooks outliers and to generally use the distance between points in a cluster to further improve upon features for clustering as well as determine what are the characteristics of particular clusters.

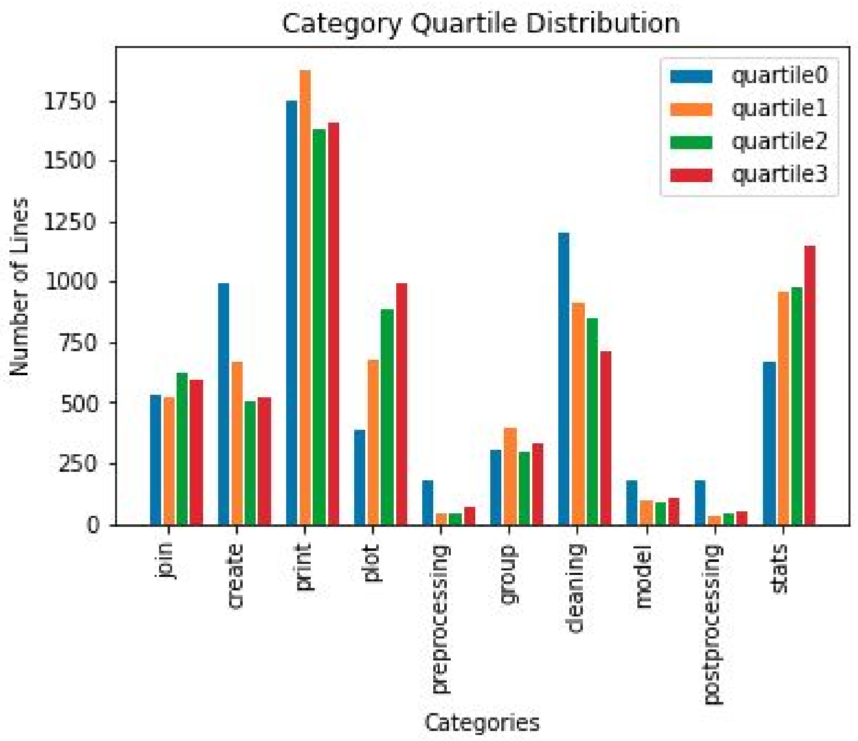


Another consideration made was the presence of certain packages. Although they were used as features, they didn't always seem to be successful in separating clusters further.

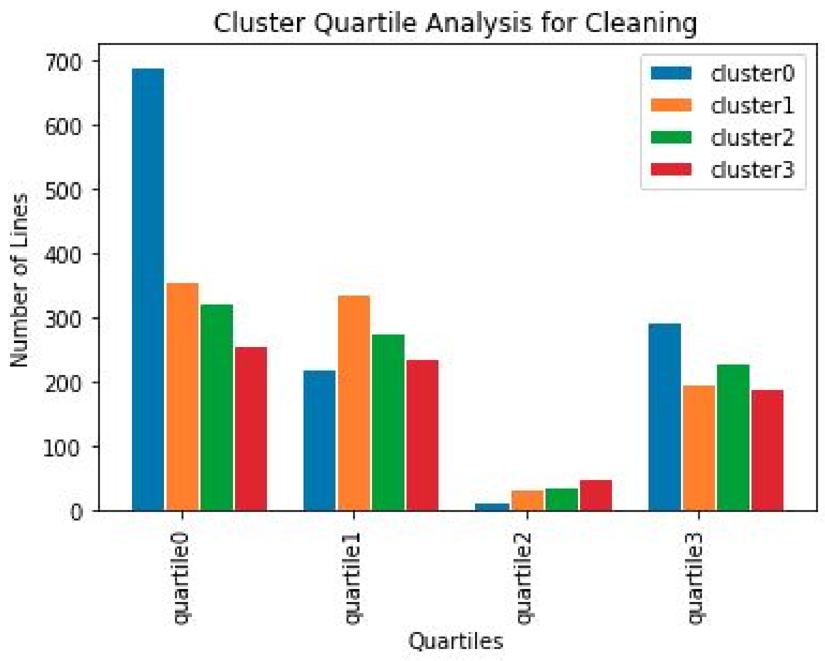
However, the frequency of imports was also compared across clusters. The interesting takeaway from this graph is that it corroborates the earlier discovery that cluster 1 does not have as many notebooks doing machine learning, since the relative frequency for sklearn (a primary ML package) is lower in that cluster. Again, matplotlib and pandas categories don't show much (will remove) since those categories were part of the filter criteria.



At this point, we realized one of the major weaknesses of our features was that while transition probabilities captured relative movement from one category to another, there was no sense of where in the whole notebooks those functions took place. Thus, we examined different quartiles in our notebooks and looked at how frequently certain functions occurred in those different quartiles. Some quartile distributions make a lot of sense; in particular, `creating` and `cleaning` happen more frequently in the first quartile while `stats` and `plotting` happen more frequently in the later quartiles. However, `post processing` confusingly has a high frequency in the first quartile.



A closer analysis at the quartile distribution for `cleaning` across the different clusters also leads to some interesting patterns. Again, relative frequency of the function could be affecting the charts, and there would be greater confidence if we conducted this on the entire dataset.



Existing Limitations

Noted w/ comments in-line. Further discussion / revisit below.

Filtering

Why do we enforce pandas/matplotlib? How can we better exclude demos/homeworks? Is this what we want to exclude?

Function Classification

Return to the idea of scraping pandas libraries and getting a more complete list of all the functions used. Furthermore, one function can be used for two purposes. `Group` could be used for wrangling and plotting

Improve on how functions are detected. Right now, if statements, loops, and function declarations/calls are not detected.

Clustering

First, I'm still confused on why we can't upscale all our notebooks. Let's revisit this.

Do our clustering features make sense? Aditya brought up the point of how scale is different. I removed raw counts beforehand anyways since they were noisy features.

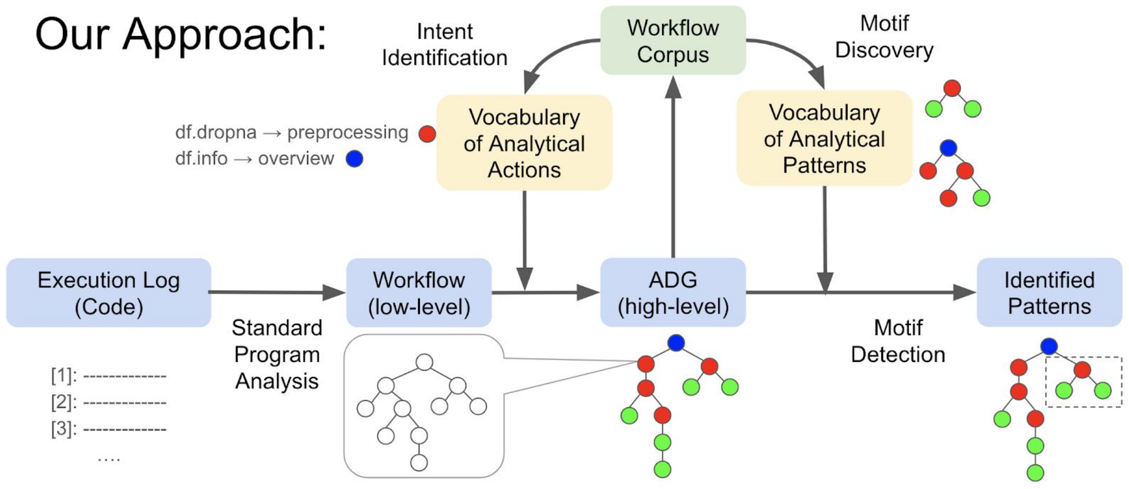
Future Directions

- Improving existing work:
    - Research questions:
        - How does a notebook breakdown based on cleaning, visualization, modelling? Are the steps mostly interspersed or sequential?
    - Copy-and-paste behavior
        - Related paper https://arxiv.org/pdf/2005.13709.pdf
            - extensive analysis of copy-and-paste behavior, showing that visualization code is the most frequently copy-and-pasted
        - Unanswered questions:
            - What is changed in copy-and-pasting?
            - Can copy-and-paste define a "split point" in the analytical decision graph analysis?
    - Scaling up: Conduct analysis on full corpus of 1.25 million notebooks.

        - Validation: Go line by line for N notebooks and check how well function detection works. Document exactly what belongs to ‘other’. Previous documentation of what we observe in notebooks has been too informal.
        - Develop a set of rules / generalize on certain patterns (eg. 60% of notebooks in cluster 1 go through a create, clean, plot pattern)

Future directions:

- Could things we could do:
    - manual analysis of notebooks:
        - are the classifications accurate?
        - look at couple dozen notebook to draw the DAG
        - are the current action currently complete enough to capture what we want?
        - can we build some AST check/dynamic checkers to capture these patterns?
    - nice cleaned notebooks v.s. total sloppy notebooks
        - Can we use nice notebooks and their markdown explanation to get at what types of alternatives people want to do? And compare them with the graphs from the sloppy notebooks. (Before and after “gather” notebooks)
        - what is the canonical graph? comparison on what people things would happen after they clean things up
    - Is this analysis something that could feed into a paper about Lux as a motivation study? Thinking about Lux as a tool that offers multiverses.
- Multiverse analysis/Constructing analytical decision graph



Our Approach:
Intent Identification
Workflow Corpus
Motif Discovery
df.dropna → preprocessing
df.info → overview
Vocabulary of Analytical Actions
Vocabulary of Analytical Patterns
Execution Log (Code)
[1]: -------------
[2]: -------------
[3]: -------------
....
Standard Program Analysis
Workflow (low-level)
ADG (high-level)
Motif Detection
Identified Patterns