Exploratory Data Analysis of DonorsChoose.Org Dataset Using Spark-Shell

Data Provided by Kaggle via Data Science for Good Competition Series

Monique Dargis
Graduate Programs in Software
University of St. Thomas
Saint Paul, MN 55105
monique.dargis@stthomas.edu

1 Data Source

The dataset for this project was obtained from Kaggle, a website that provides open datasets for use in data science projects. Kaggle regularly hosts competitions in which data scientists can compete to provide the best solution given a problem and a dataset. The dataset used for this project contains roughly 3.9GB of data spread over six .csv files and was provided by Kaggle as part of its Data Science for Good competition series.

The data originates from DonorsChoose.org, a website that allows public school teachers to request funding for classroom projects. The DonorsChoose.org funding model asks teachers to submit proposals outlining the goal of their project, resources needed, and cost of the project. Donors can then fund all or part of a project via crowd funding by donating toward funds for the purchase of specific resources. For the Kaggle competition, DonorsChoose.org provided information on roughly 4.69 million unique donations made over the course of five and a half years, as well as associated data such as project details and donor characteristics. A data dictionary with an account of the .csv files used for this project and features within those files can be found in Appendix 1.

The challenge posed in the Data Science for the Social Good competition was to build a model to "enable DonorsChoose.org to build targeted email campaigns recommending specific classroom requests to prior donors." The competition ended in June 2018, and winning entries utilized a variety of machine learning techniques including time series analysis and geospatial analysis. Such techniques are beyond the scope of this project, which will instead focus on exploratory data analysis.

1.1 Data Validation

One of the benefits of choosing a dataset provided by Kaggle is that statistics on the dataset are provided by Kaggle. A preview of the data was provided for each of the .csv files except for `Projects.csv` (which was too large to preview), and feature summaries were provided that included:

- Total number of valid, mismatched, and missing entries
- Number of unique values for a feature
- Mode of the feature
- For categorical features, the two to four most common categories and their frequency in the dataset
- For numerical features, the minimum and maximum values
- For numerical features, the mean and standard deviation

This information proved useful when validating results of the exploratory data analysis completed for this project. Additional validation of results was completed by loading .csv files into a python pandas dataframe in a jupyter notebook (see Appendix 2). This proved especially important for the `Projects.csv` file, which had no Kaggle preview.

2 Data Loading

Ultimately, only three of the six .csv files provided by Kaggle were used for this project - `Donations.csv`, `Donors.csv`, and `Projects.csv`, with a combined file size of 3.09 GB.

Because the data was provided in .csv files, the spark-csv package from databricks was used, specifically version 1.5.0 using Scala 2.10. An attempt was made to infer the data schema from each csv file, but errors in this process necessitated the creation of schemas for use within spark-shell. To complete this task, it was necessary to import all datatypes used in spark SQL.

For each .csv file loaded, the general process was to define a schema, including datatypes; import the .csv file as a data frame using that schema; create a case class to define the data structure; and convert the data frame to a data structure using that case class. At first, each data structure was converted to an RDD with all columns included. However, it soon became evident that smaller RDDs would lead to shorter compute times, and that some fields, such as Donor Cart Sequence, would not be used in the final analysis. Therefore, RDDs were only created when necessary for the task at hand and were limited to only the columns needed for analysis. An example of this is analysis of the amount given in each individual donation, which was explored early in the project. An RDD was created that contained only the data drawn from the Donation Amount column in the original .csv file, and statistics were generated based on that RDD.

2.1 Data Load Challenges

Loading the `Projects.csv` file proved a challenge. This file was initially loaded in the same way as the other .csv files with datatypes specified by a user-created schema. However, when an attempt was made to convert the data to an RDD, a `NumberFormatException` was thrown due to Spark attempting to convert a string value into a number. To explore this error, the .csv file was loaded with the schema inferred rather than user-specified, and Spark inferred that all datatypes should be given as string values. When the data was then converted into an RDD, the problem was obvious. When the `Projects.csv` file was loaded and converted to a data frame, the Project Essay field was incorrectly split among multiple columns. This caused parts of the essay to load into fields that had been defined as containing float data types, which caused a `NumberFormatException` when converting to an RDD.

Because no preview was available for the `Projects.csv` file on Kaggle.com, the file was loaded into a python pandas data frame so it could be explored. Curiously, the data loaded correctly into this data frame, and the data schema was correctly inferred by pandas, as seen in Appendix 2. Loading the data into a pandas data frame allowed examination of the entries in the Project Essay column, which revealed a curious structure to the essay itself. Each essay contained multiple breaks with the following characters:

```
<!--DONOTREMOVEESSAYDIVIDER-->
```

It is possible that this formatting caused issues when the Project Essay field was loaded from the `Projects.csv` file; however, attempts to rectify the load issue in Spark 1.6 did not prove fruitful. When the .csv file was loaded in Spark 2.0, which includes .csv file load support without third party packages, data loading was successful. However, because of the differences between Spark 1.6 and Spark 2.0, and the possibility of a complete project without use of the `Projects.csv` file, this approach was abandoned. Instead, when `Projects.csv` was loaded into a Spark data frame, a `DROPMALFORMED` option was invoked. This allowed the .csv file to successfully load into the user specified schema, but also reduced the dataset from 1,110,017 records (shown in pandas) to 355,954 records. Because of this drastic reduction in data, analysis on `Projects.csv` was limited, and data contained in this file was not joined with data from `Donations.csv` or `Donors.csv`.

3 Questions of Interest

Exploratory Data Analysis focused on a few main lines of inquiry related to donation statistics, donor characteristics, and likelihood of a project being fully funded. The first two lines of inquiry were explored using data from `Donations.csv` and `Donors.csv`, while the latter was explored using the reduced `Projects.csv` data set.

3.1 Donation Statistics

3.1.1 Generating statistics After `Donations.csv` was loaded into a data frame and then a data structure, the Donation Amount column was loaded into an RDD. The `.stats` command was used to generate donation amount statistics. The total number of donations was 4,687,884 (µ = $60.67, *SD* = $166.90, range = $0.01 - $60,000). The `.reduce(_+_)` command was used to accumulate the values of all of the donations, and revealed that all donations combined totaled $283,355,520. These statistics matched the figures given on Kaggle and in the pandas data frame.

Beyond understanding general donation information, it is useful to know the habits of individual donors. A new RDD named `DonorID_Amount` was created from two columns within the donations data structure – Donor ID and Donation Amount. This was then converted into a new RDD, `DonorID_Amount_reduced` using the `.reduceByKey(_+_)` action to accumulate the total amount that each unique donor contributed. The `.stats` command was then called on the accumulated unique donor contributions to reveal that 2,024,554 unique donors made contributions (µ = $140.48, *SD* = $2,437.80, range = $0.01 - $1,879,625.25). When the total number of donations is divided by the number of unique donors, we can see that each donor made, on average 2.31 donations.

Next, donation data was accumulated based on the date of the donation. This was done so that any patterns in giving based on date could be examined. Donation Date and Donation Amount columns were pulled from the donations data structure and converted to an RDD. The Donation Date information was in the format `"YYYY-MM-DD HH:MM:SS"`. For the purposes of this project, only the date was required, so this field was split on the space character using `.split(" ")`. This resulted in an RDD with form `Array[Array[String], Float]`. A map function was used to generate Key/Value pairs by selecting the first item in the Array of String values as the Key and the Float (corresponding to the Donation Amount) as the Value. `.reduceByKey(_+_)` was then used to accumulate the total amount of donations given on each date. This output was sorted from earliest date to latest date using `.sortByKey(ascending=true)` and saved to a text file. That text file was later used to generate the graph found in Appendix 3.

In addition to the total amount given each day, the total number of donations on a given date was accumulated. The Donation Date column alone was pulled from the donations data structure to generate an RDD. Again, Donation Date needed to be split and re-mapped as described in the previous paragraph. Instead of including Donation Amount, each Donation Date was mapped as a Key with a corresponding Value of 1. `.reduceByKey(_+_)` was then used to accumulate the total count of donations given on each date, with `.sortByKey(ascending=true)` again invoked and the RDD saved as a text file. This data is also included in the graph found in Appendix 3.

3.1.2 Analysis The Donation Amount by Date and Number of Donations by Date data was converted into an Excel file and a graph was generated, which can be viewed in Appendix 3. There is a pattern to the dates on which donations are made. Several peaks in donation correspond with matching drives promoted by DonorsChoose.Org, including Back-to-School Boost (early September/late August each year), Black Friday 2016 and 2017, #BestSchoolDay on 3/29/2017, and #FirstMillion on 1/25/2018. One item of note is that Black Friday drives tend to generate similar total dollar amount donated in comparison to the Back-to-School Boost drives, but a much larger total number of donations. This would indicate that the average donation given during the Black Friday drives is smaller than during some other initiatives. In addition to the peaks in donations generated by the matching drives promoted by DonorsChoose.org, December 31 of each year sees a peak in donations. End-of-year donations are a well-known phenomenon in fundraising as donors attempt to maximize charitable donation tax deductions for a given tax year.

3.2 Donor Characteristics and Influence on Donations

3.2.1 Generating Teacher-Donor Statistics After `Donors.csv` was loaded into a data frame and then a data structure, the Donor ID and Donor Is Teacher columns were loaded into an RDD. The `.count` command was used to check the total number of donors in the pool. This command showed that there were 2,122,640 donors included in the `Donors.csv` file, which matched the number given in the Kaggle dataset preview and the number generated by the pandas data frame. When compared to the 2,024,554 unique donors found in the `Donations.csv` file (*Section 3.1.1*), we can see that there are 98,086 donors in the DonorsChoose.Org system who did not make any donations in this dataset.

Next, the RDD was filtered on the Donor Is Teacher field, using `.contains("Yes")` followed by `.count`. There are 212,285 donors who are also teachers within the DonorsChoose.org system, comprising 10.00% of the total donor pool. This number also matches both Kaggle's dataset preview and the pandas data frame.

The total number of donations was 4,687,884 (μ = $60.67, *SD* = $166.90, range = $0.01 - $60,000). The `.reduce(_+_)` command was used to accumulate the values of all of the donations, and revealed that all donations combined totaled $283,355,520. These statistics matched the figures given on Kaggle and in the pandas data frame.

3.2.1 Teacher-Donor vs Non-Teacher-Donor Donations A left outer join was used to combine the `DonorID_Amount_reduced` and `DonorID_IsTeacher` RDDs. A simple join function was first attempted, but the reduced number of entries in the dataset indicated that there may be some donors in the RDD `DonorID_Amount_reduced` that were not present in the `DonorID_IsTeacher` RDD, so a left outer join was used instead. This join was then mapped to a `DonorID_Amount_r_IsTeacher` RDD using `.map{ case(x: String, (y: Float, z: (option[String])) => (x, (y, (z.getOrElse())))`. This map was necessary so that a filter of `z.tostring.contains("Yes")` could be used to separate the RDDs depending on whether the donor was a teacher. The result was an RDD of the form `Array[(x, (y, z))]` where `x` is the unique Donor ID, `y` is the total amount donated by that donor, and `z` shows whether the donor is also a teacher.

After filtering into two RDDs – one with only donors who were identified as teachers, and one with only donors who were not identified as teachers – statistics were generated on the Donation Amounts for both teacher and non-teacher RDDs using `.stats`. The total number of teacher-donors who made at least one donation was 210,239 (μ = $290.18, *SD* = $1030.81, range = $1.00 - $118,311.20). The total number of non-teacher-donors who made at least one donation was 1,814,315 (μ = $123.13, *SD* = $2,550.59, range = $0.01 - $1,879,625.25).

In addition to finding the total amount each donor gave depending on their status as a teacher or non-teacher, the `DonorID_Amount` and `DonorID_IsTeacher` RDDs were combined using a left outer join to show non-aggregated donation information. The same process as was used to generate `DonorID_Amount_r_IsTeacher` was used to create an RDD named `DonorID_Amount_IsTeacher`. However, with non-aggregated donation amount information, statistics could be generated on amount that teachers and non-teachers donated per individual donation. Teachers were responsible for a total of 1,339,221 individual donations (μ = $45.55, *SD* = $215.70, range = $0.01 - $21,299.95). `.reduce(_+_)` was run to discover that teacher-donors donated a total of

$61,006,864. Non-teachers were responsible for a total of 3,348,663 individual donations (μ = $66.71, *SD* = $142.34, range = $0.01 - $60,000.00), totaling $222,875,040 in total donations.

3.2.1 Optional Donations Donors have the opportunity to contribute an additional 15% of their donation directly to DonorsChoose.org to fund overhead for the organization. The Donation Included Optional Donation column was converted to an RDD and filtered using `.contains("Yes")` to see how many donations included this additional 15%. `.count` was applied to discover that 4,001,709, or 85.36%, of the donations included the optional donation. This matches the figure given on the Kaggle dataset preview.

The frequency of optional donations given by teacher-donors and non-teacher-donors was also examined. The Donor ID and Donor Included Optional Donation fields from the donations data structure were converted to an RDD named `DonorID_OptionalDonation`. A left outer join was then performed to join `DonorID_OptionalDonation` with `DonorID_IsTeacher`. As with the previous RDDs generated via left outer join, `.map{ case(x: String, (y: String, z: (option[String])) => (x, (y, (z.getOrElse())))` was used to ensure that `z` could be filtered with `.tostring.contains("Yes")` to separate the RDDs depending on whether the donor was a teacher. In the same filter function, `.contains("Yes")` and `.contains ("No")` were applied to `y` to filter based on an included optional donation. The result was four RDDs on which `.count` was called to find the frequencies of donations. Results can be seen in Table 1.

The frequencies of optional donations can be combined with previous results to find estimates of the total amount donated to DonorsChoose.org by both teachers and non-teachers. Teachers included optional donations in 1,031,318 (77.01%) of their 1,339,221 donations. With an average amount per donation of $45.55, this translates to an estimated total of $7,046,590.18 given by teachers to DonorsChoose.org. Non-teachers included optional donations in 2,970,391 (88.70%) of their 3,348,663 donations. With an average amount per donation of $66.71, this translates to an estimated total of $29,721,947.53 given directly to DonorsChoose.org.

		Donor Is Teacher	
		Yes	No
Donor Included Optional Donation	Yes	1,031,318	2,970,391
	No	307,903	378,272

Table 1. Donation Count by Donor Type and Optional Donation

3.3 Project Essays

As mentioned in Section 2.1, `Projects.csv` proved difficult to load into a data frame in Spark-Shell. The best solution to this data load problem resulted in a data set that was reduced to roughly a third of its original size. Analysis was completed on this dataset for practice with data cleaning and word count; however, the results of this analysis should be taken with a grain of salt. An attempt was made to complete work on the `Projects.csv` file using spark-submit rather than spark-shell. Differences between spark versions on Eclipse on the virtual machine (using spark 1.3) and the class cluster (using spark 1.6) led to multiple failures, and this approach was abandoned. Data loading, data cleaning, and word count were ultimately accomplished using spark-shell command line.

3.3.1 Word Count by Project Funded Status Each project included an essay, written by the teacher requesting funding, explaining the purpose of the project. Word count was performed on these essays to determine whether there were differences between the words used in essays of projects that were fully funded versus projects that were not fully funded.

After `Projects.csv` was loaded using the `DROPMALFORMED` option, as detailed in Section 2.1, Project Essay and Project Current Status fields were loaded into an RDD called `ProjectsEssay_Status`. The RDD was then split in two using a filter on the Project Current Status field. `projectsWC_Funded` contained all projects that were designated as "Fully Funded" and `projectsWC_NotFunded` contained all projected that were not "Fully Funded." The same mapping and filtering techniques were performed on both RDDs to obtain word count of the most commonly-used words for each type of project. First, a `.map` was performed so that only the essay was included, and then `.flatMap` was applied with `.split("\\W+")` to split each essay into its component words.

After mapping, several filters were required to remove non-words. First, a filter was applied to remove all of the essay dividers using `!(x.contains("DONOTREMOVEESSAYDIVIDER"))`. A filter on `_.length!=0` was used to remove null values, and a filter on `_(0).isLetter` was used to remove numerical values.

Finally, a map function was used to generate Key/Value pairs by mapping a lowercase version of each word as a Key and a corresponding Value of 1. `.reduceByKey(_+_)` was then used to accumulate the word count for each unique word and sorted by descending Value using `.sortBy(-_._2)`. The output for each RDD was saved to `project/projectsWC_Funded` and `project/projectsWC_NotFunded`.

3.3.2 Analysis of Word Frequency `Part-00000` of RDDs `projectsWC_Funded` and `projectsWC_NotFunded` were loaded into Excel and words were ranked based on their relative frequency within the essays of projects that were funded and projects that were not funded. Only the first partition of each RDD was used because these parts alone accounted for more than 2500 of the top ranked words in the essays of both funded and not funded projects. As would be expected, many of the top ranked words were prepositions and articles, and the top 10 most common words were identical among the two lists:

```
1.  to
2.  the
3.  and
4.  students
5.  a
6.  of
7.  in
8.  my
9.  will
10. they
```

Of the top forty most frequent words, the difference in ranking between the two lists did not vary by more than 3 places. However, further down the list there are were larger differences in rankings between the essays of projects that were funded and essays of projects that were not funded.

Table 2 shows words ranked in the top 200 of each list that had a difference in ranking of more than 20. A positive value in the final column indicates that the word was found more frequently in the essays of projects that were fully funded, while a negative value indicates that the word was found more frequently in the essays of projects that were not fully funded. Words related to reading and exploration `[book, play, library, music, build]` were ranked higher in essays of projects that were fully funded, while words related to technology `[ipad(s), computer(s)]` were ranked higher in essays of projects that were not fully funded.

4 Performance and Limitations

When all lines of code are copied into spark-shell and run one after the other, the total time to complete all data loading and transformations is 11 min, 47 sec. As the largest .csv file, it should come as no surprise that the transformations related to `Projects.csv` took the longest to complete. Each of the three transformations needed to obtain results related to `Projects.csv` took between 1 min, 50 sec and 2

Word	Rank – Project Fully Funded	Rank – Project Not Fully Funded	Difference in Ranking
book	109	139	30
access	110	86	-24
supplies	156	181	25
ipad	163	114	-49
play	165	200	35
library	172	202	30
music	173	209	36
build	197	222	25
future	207	182	-25
computer	208	161	-47
research	229	174	-55
ipads	239	150	-89
computers	304	199	-105

Table 2. Word Rank by Project Funding Status

min, 9 sec to complete. Default Spark-Shell values were used for memory, number of cores, and number of executors as the amount of time needed to complete transformations was not excessive. The use of Spark and a Hadoop cluster is certainly advised for a dataset of this size; an attempt was made to calculate the sum and mean of the Donation Amount column after `Donations.csv` was loaded to a pandas data frame and was abandoned after the python kernel ran for over an hour without producing a result.

Working with .csv files proved a challenge in terms of data loading. The source of the issues with the `Projects.csv` file load was never discovered in Spark 1.6. Spark 2.0 was able to handle the structure of `Projects.csv` file, which contained the most interesting data and would have provided rich opportunities for analysis, but because other work was done in Spark 1.6 there was a reluctance to switch Spark versions mid-project. Although the use of a much reduced `Projects.csv` dataset allowed some analysis, the results cannot be trusted because of the omission of two-thirds of the original data. With more time and further research into the differences between Spark 1.6 and Spark 2.0, it is possible this project could have been completed in Spark 2.0 and the full `Projects.csv` file used. This would have allowed the exploration of more interesting data questions, such as: whether male or female teachers are more likely to have their projects funded; whether donors are more likely to donate to projects in schools local to them; and the interaction between the date a project is posted and its likelihood of receiving full funding.

Although a cursory glance into word count was accomplished using the `Projects.csv` file, this avenue was not fully explored because the data was incomplete. With a more complete dataset, it would be interesting to remove stop words and use a stemming algorithm to get a more accurate picture of the words used most frequently in the Project Essay field.

Another difficulty was the use of Spark-Submit. Using the databricks spark-csv package in conjunction with two versions of Spark (Spark 1.3 in the virtual machine where jar files were created with Eclipse, and Spark 1.6 in the cluster where jar files were executed) proved to be too great a challenge to overcome in a short time frame. With additional time for exploration, it would be good to learn more about the differences between the Spark version and how to handle these differences when submitting a .jar file in Eclipse.

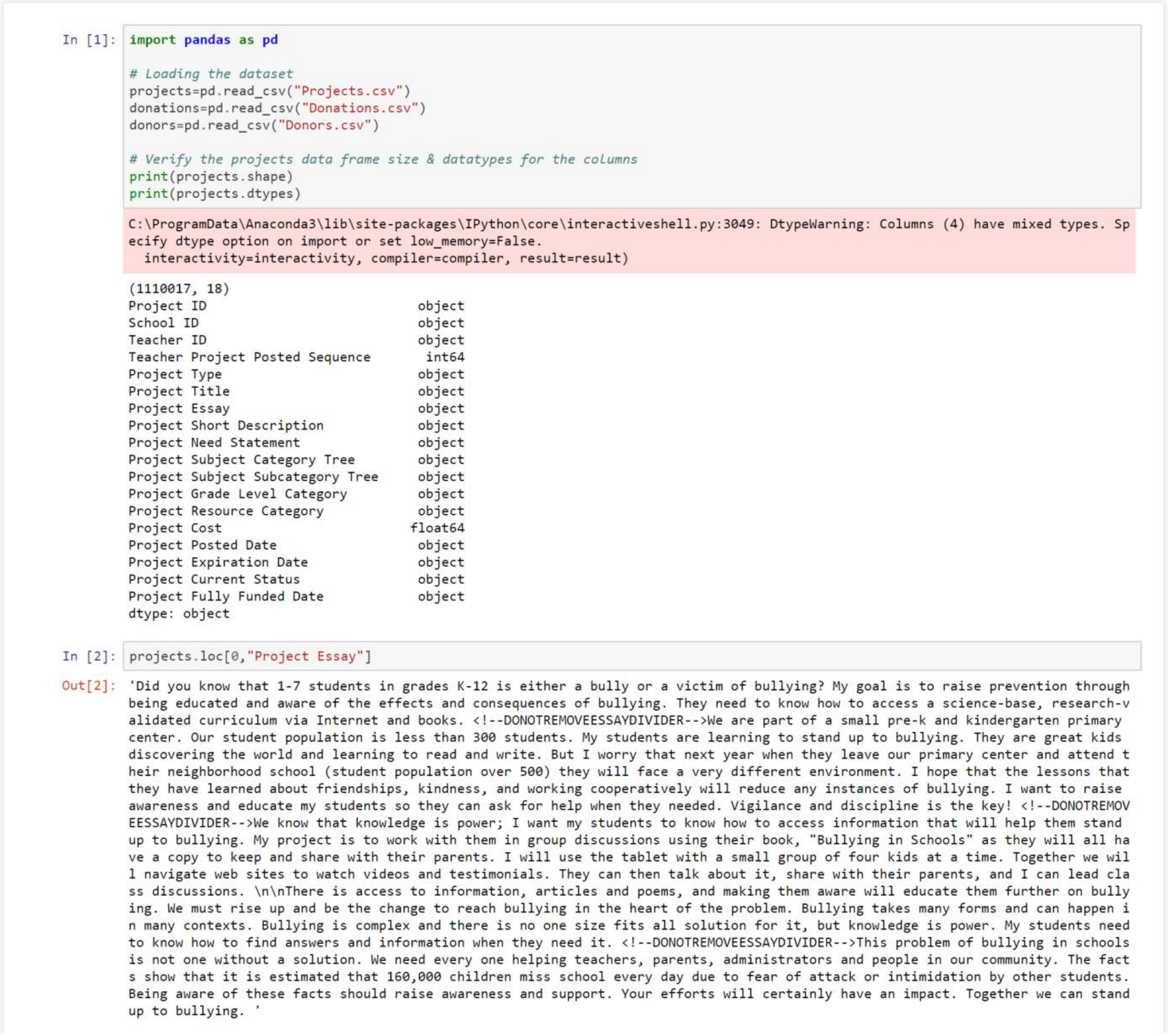
APPENDIX 1

All descriptions obtained from Kaggle's Data Science for Good: DonorsChoose.org competition page and discussion page.

File	Feature	Data type	Description
Donations	Project ID	String	Unique identifier of a project.
583.03 MB	Donation ID	String	Unique identifier of a donation.
	Donor ID	String	Unique identifier of a donor.
	Donation Included Optional Donation	String	Yes/No to give 15% of donation amount to DonorsChoose.org.
	Donation Amount	Float	Total amount donated for a project.
	Donor Cart Sequence	Integer	Project position on list of desired donations within Cart list.
	Donation Received Date	String	Date and time on which the donation was received.
Donors	Donor ID	String	Unique identifier of a donor.
118.24 MB	Donor City	String	The donor's city
	Donor State	String	The donor's state
	Donor Is Teacher	String	Whether or not the donor is also a teacher with a DonorsChoose.org teacher account (Yes/No)
	Donor Zip	String	The donor's zip code (only first 3 digits)
Projects	Project ID	String	Unique identifier of a project
2.39 GB	School ID	String	Unique identifier of a school
	Teacher ID	String	Unique identifier of a teacher
	Teacher Project Posted Sequence	Integer	The teacher's 1st, 2nd, 3rd…nth posted project
	Project Type	String	"Teacher-Led", "Student-Led", or "Professional Development"
	Project Title	String	Title of project given by teacher
	Project Essay	String	Contains essay text written by the teacher describing their students and how the items will be used in their classroom
	Project Short Description	String	The first 198 characters of the essay followed by ellipses. This used to appear on a project card as donors are browsing through the Search page. It is no longer used for this purpose.
	Project Need Statement	String	A description of resources requested written by the teacher. This text appears on a project card as donors are browsing through the Search page.
	Project Subject Category Tree	String	Every project can have either one or two subject categories. This is a comm-separated list of those subject categories.
	Project Subject Subcategory Tree	String	For each project subject category, there is also a project subject subcategory that contains greater specificity.
	Project Grade Level Category	String	One of four categories describing the grade level of students.
	Project Resource Category	String	The types of items being requested by a teacher. There were five resource categories prior to October 10, 2017. After this date,

			there are fourteen. Prior to October 10, 217, these categories were selected by teachers during project creation. After October 10, 2017, these categories were predicted via algorithm at roughly 95% accuracy.
	Project Cost	Float	This cost includes materials cost + fees associated with project fulfillment. It’s the total a teacher must raise to have a project fully funded and receive their materials.
	Project Posted Date	String	The date on which a project was published on the DonorsChoose.org site for people to donate to.
	Project Expiration Date	String	A project is typically live on the site for 4 months after it is posted. The expiration date is the date in which the project will no longer be eligible to receive donations.
	Project Current Status	String	“Fully Funded”, “Expired”, or “Live”. A project is typically live for 4 months after being posted. After that date, if a project is not “Fully Funded”, it is marked “Expired”. At any point in time after a project is posted, a staff member or the teacher can take the project off the site by “Archiving” the project for various reasons.
	Project Fully Funded Date	String	If the project was fully funded, this is the date on which the project received full funding.

APPENDIX 2



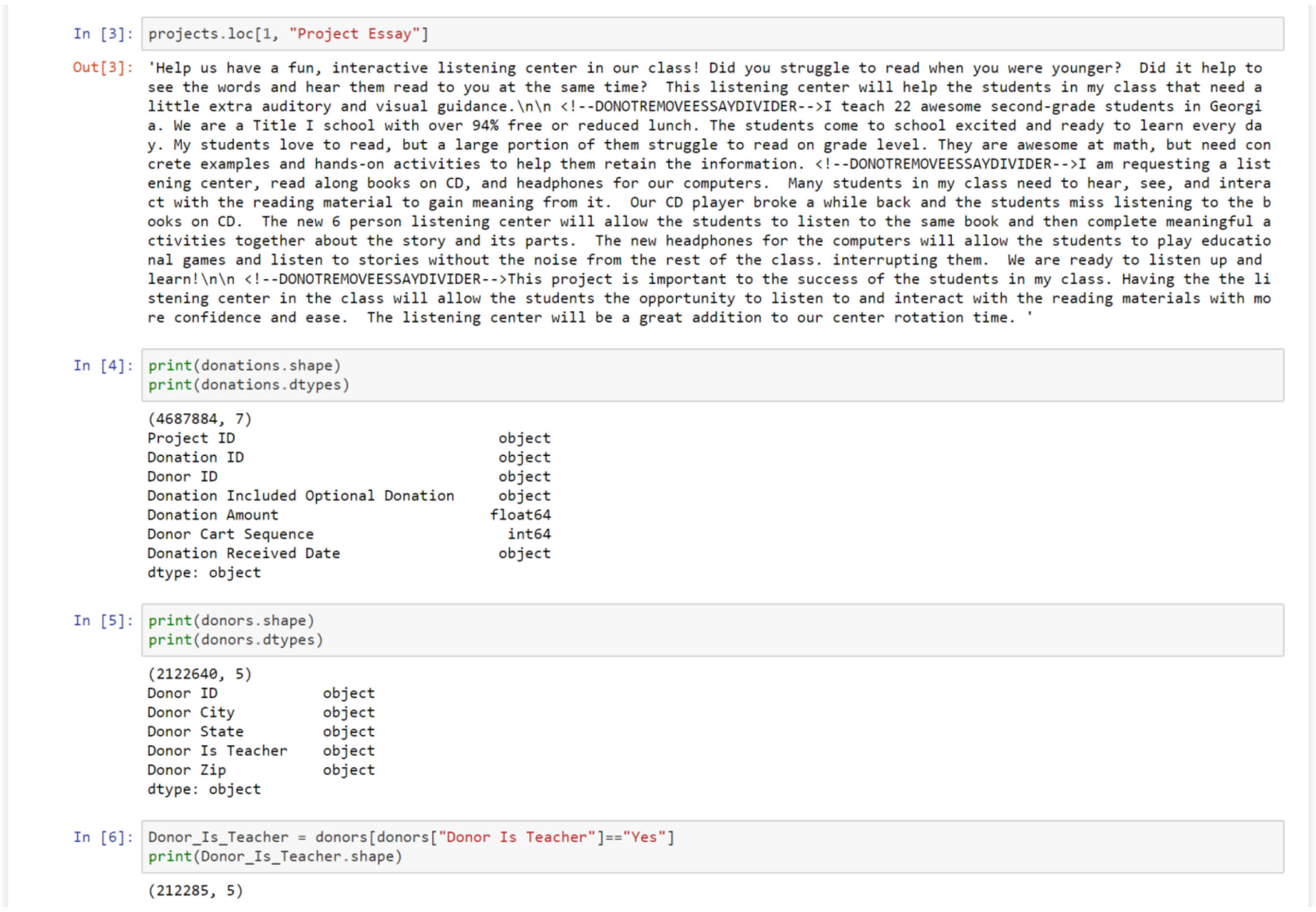
```
In [1]: import pandas as pd

        # Loading the dataset
        projects=pd.read_csv("Projects.csv")
        donations=pd.read_csv("Donations.csv")
        donors=pd.read_csv("Donors.csv")

        # Verify the projects data frame size & datatypes for the columns
        print(projects.shape)
        print(projects.dtypes)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3049: DtypeWarning: Columns (4) have mixed types. Sp
ecify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
(1110017, 18)
Project ID                            object
School ID                             object
Teacher ID                            object
Teacher Project Posted Sequence        int64
Project Type                          object
Project Title                         object
Project Essay                         object
Project Short Description             object
Project Need Statement                object
Project Subject Category Tree         object
Project Subject Subcategory Tree      object
Project Grade Level Category          object
Project Resource Category             object
Project Cost                         float64
Project Posted Date                   object
Project Expiration Date               object
Project Current Status                object
Project Fully Funded Date             object
dtype: object
```

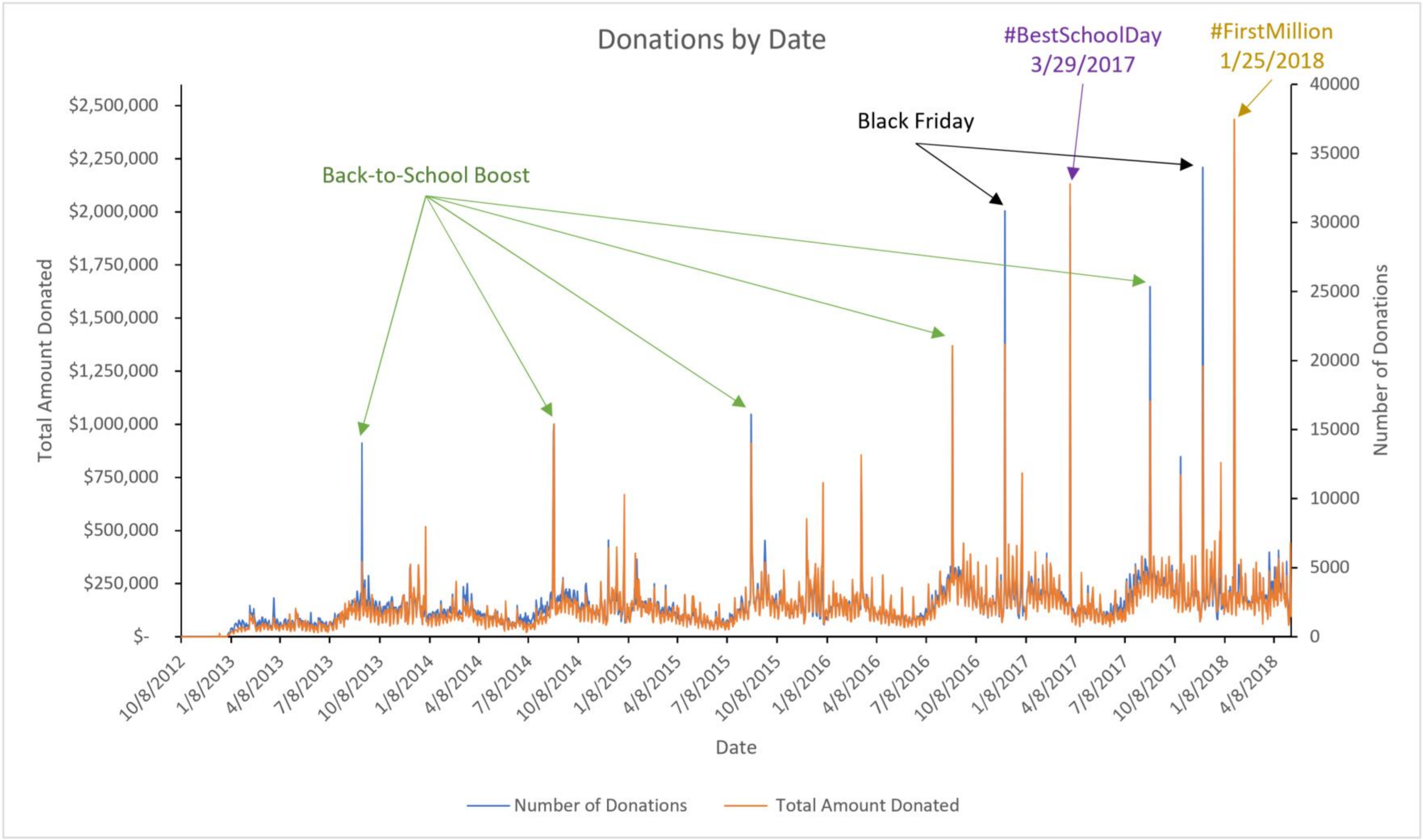
```
In [2]: projects.loc[0,"Project Essay"]
```

```
Out[2]: 'Did you know that 1-7 students in grades K-12 is either a bully or a victim of bullying? My goal is to raise prevention through
being educated and aware of the effects and consequences of bullying. They need to know how to access a science-base, research-v
alidated curriculum via Internet and books. <!--DONOTREMOVEESSAYDIVIDER-->We are part of a small pre-k and kindergarten primary
center. Our student population is less than 300 students. My students are learning to stand up to bullying. They are great kids
discovering the world and learning to read and write. But I worry that next year when they leave our primary center and attend t
heir neighborhood school (student population over 500) they will face a very different environment. I hope that the lessons that
they have learned about friendships, kindness, and working cooperatively will reduce any instances of bullying. I want to raise
awareness and educate my students so they can ask for help when they needed. Vigilance and discipline is the key! <!--DONOTREMOV
EESSAYDIVIDER-->We know that knowledge is power; I want my students to know how to access information that will help them stand
up to bullying. My project is to work with them in group discussions using their book, "Bullying in Schools" as they will all ha
ve a copy to keep and share with their parents. I will use the tablet with a small group of four kids at a time. Together we wil
l navigate web sites to watch videos and testimonials. They can then talk about it, share with their parents, and I can lead cla
ss discussions. \n\nThere is access to information, articles and poems, and making them aware will educate them further on bully
ing. We must rise up and be the change to reach bullying in the heart of the problem. Bullying takes many forms and can happen i
n many contexts. Bullying is complex and there is no one size fits all solution for it, but knowledge is power. My students need
to know how to find answers and information when they need it. <!--DONOTREMOVEESSAYDIVIDER-->This problem of bullying in schools
is not one without a solution. We need every one helping teachers, parents, administrators and people in our community. The fact
s show that it is estimated that 160,000 children miss school every day due to fear of attack or intimidation by other students.
Being aware of these facts should raise awareness and support. Your efforts will certainly have an impact. Together we can stand
up to bullying. '
```



```
In [3]:  projects.loc[1, "Project Essay"]

Out[3]:  'Help us have a fun, interactive listening center in our class! Did you struggle to read when you were younger?  Did it help to
         see the words and hear them read to you at the same time?  This listening center will help the students in my class that need a
         little extra auditory and visual guidance.\n\n <!--DONOTREMOVEESSAYDIVIDER-->I teach 22 awesome second-grade students in Georgi
         a. We are a Title I school with over 94% free or reduced lunch. The students come to school excited and ready to learn every da
         y. My students love to read, but a large portion of them struggle to read on grade level. They are awesome at math, but need con
         crete examples and hands-on activities to help them retain the information. <!--DONOTREMOVEESSAYDIVIDER-->I am requesting a list
         ening center, read along books on CD, and headphones for our computers.  Many students in my class need to hear, see, and intera
         ct with the reading material to gain meaning from it.  Our CD player broke a while back and the students miss listening to the b
         ooks on CD.  The new 6 person listening center will allow the students to listen to the same book and then complete meaningful a
         ctivities together about the story and its parts.  The new headphones for the computers will allow the students to play educatio
         nal games and listen to stories without the noise from the rest of the class. interrupting them.  We are ready to listen up and
         learn!\n\n <!--DONOTREMOVEESSAYDIVIDER-->This project is important to the success of the students in my class. Having the the li
         stening center in the class will allow the students the opportunity to listen to and interact with the reading materials with mo
         re confidence and ease.  The listening center will be a great addition to our center rotation time. '

In [4]:  print(donations.shape)
         print(donations.dtypes)

         (4687884, 7)
         Project ID                                 object
         Donation ID                                object
         Donor ID                                   object
         Donation Included Optional Donation        object
         Donation Amount                           float64
         Donor Cart Sequence                         int64
         Donation Received Date                     object
         dtype: object

In [5]:  print(donors.shape)
         print(donors.dtypes)

         (2122640, 5)
         Donor ID            object
         Donor City          object
         Donor State         object
         Donor Is Teacher    object
         Donor Zip           object
         dtype: object

In [6]:  Donor_Is_Teacher = donors[donors["Donor Is Teacher"]=="Yes"]
         print(Donor_Is_Teacher.shape)

         (212285, 5)
```

APPENDIX 3


Donations by Date
#BestSchoolDay 3/29/2017
#FirstMillion 1/25/2018
Black Friday
Back-to-School Boost
Total Amount Donated
$2,500,000
$2,250,000
$2,000,000
$1,750,000
$1,500,000
$1,250,000
$1,000,000
$750,000
$500,000
$250,000
$-
Number of Donations
40000
35000
30000
25000
20000
15000
10000
5000
0
10/8/2012
1/8/2013
4/8/2013
7/8/2013
10/8/2013
1/8/2014
4/8/2014
7/8/2014
10/8/2014
1/8/2015
4/8/2015
7/8/2015
10/8/2015
1/8/2016
4/8/2016
7/8/2016
10/8/2016
1/8/2017
4/8/2017
7/8/2017
10/8/2017
1/8/2018
4/8/2018
Date
Number of Donations
Total Amount Donated