IS 532/CS513 DataBeasts Final Project Report

By: Ryan DeVries, Erin Lowe, Min Seo Mo, and Reuven Birnbaum
Box-Link: https://uofi.app.box.com/s/ickkd0oi259tgh5888zkdxd4f4pwqryt

Introduction and Overview

The DataBeast's final project involved cleaning a dataset of individual sow service, or artificial insemination, records. The dataset was obtained from a fellow graduate student in the Applied Epidemiology Lab at the College of Veterinary Medicine. This graduate student received it from a pork producer with swine breeding herds. The student's main goal is to use this dataset for training/testing machine learning techniques related to Sow Farm performance prediction. The goals of our cleaning project are:

- To tidy up the dataset so that this student can easily use the data for accurate machine learning techniques
- To develop a workflow so that cleaning of additional data points can be automated for real-time predictions
- To compare the use of multiple data cleaning programs to assess an optimal workflow for this scenario

Throughout this project, our group kept the following research questions in mind: "What is the most efficient data cleaning program(ie. What program requires the least amount of steps to complete the task)?", "What issues will we encounter with the dataset as we clean it?", "How clean does the dataset need to be for machine learning or artificial intelligence applications?", and, "What's an example of a use case that this dataset will never be clean enough to accomplish?" After data cleaning in multiple programs and checking integrity constraints, we hope to have answers.

Our data cleaning project goals involve the creation of a cleaning workflow to clean the dataset for advanced computing and provide a template for additional data points from these farms or similar datasets from other farms. To accomplish our goals, we created an extensive data cleaning items list (Appendix B). We planned to test our data cleaning steps using three different data cleaning software tools: OpenRefine, TableauPrep, and Data Wrangler, and create a pros and cons list for each of the programs. In the end we plan to provide a recommendation on the best program for use with our machine learning use case.

Initial Dataset Assessment & Use Case Discussion

Our dataset of individual sow service records had ~65,000 individual records and 53 columns. The data was collected from nine different pig farms in the year 2016, and are used on-site to assess productivity and for daily decision making. For our datasets schema, each row is a service record and unique based on the combined primary key of Farm, SowID, and Serv date. Individual sows may be listed several times as they were served several times in 2016, and Sow ID's may be duplicated across farms. The full schema of the data can be found in Appendix A, along with the description of the column, initial comments we made, and initial cleaning comments. This table also describes the content of the database and some of the quality issues we initially noticed such as the ServResult column having dates not in 2016, EntryAge being an empty column, and the Farm column that needed to be changed to remain anonymous. There are also some general facts we listed about the dataset as the average person may not be familiar with the technical farm terms.

This data set is from a farm production software system. The software system has already in place several constraints on entries to columns to mitigate the need for cleaning and improve the use of the data. For example, we did not find an instance where a date did not contain all of the values for year, month, and date as these were likely required for the field at the point of data entry. We also did not find any misspellings of the sow farm names as it was likely created by drop-down list to eliminate the chance of those errors. As stated this data is used on-farm currently to help make decisions such as “Should I cull this sow?” or “How many piglets should I foster to another sow?” based on how many piglets the sow weaned last litter. Essentially, the dataset is clean enough to parse for basic information for use on the farm, but not clean enough for the use in advanced computer programs.

The main use case for cleaning our dataset is for use in machine learning and artificial intelligence training so that it could be used for sow farm performance predictions. A specific example would be the following question: “How many pigs are the group of just bred sows likely to wean in 135 days?”. By having clean data we ensure that these machine learning predictions can be as accurate as possible. We acknowledge that despite our best efforts, this dataset will never be perfectly clean for every use case. If, for example, someone wanted to use this dataset to assess service technician performance, “How successful is Technician A compared to Technician B”, we don’t believe that our data set will be clean enough for that purpose as we do not know if the technician IDs represent the correct IDs of people actually on the farm.

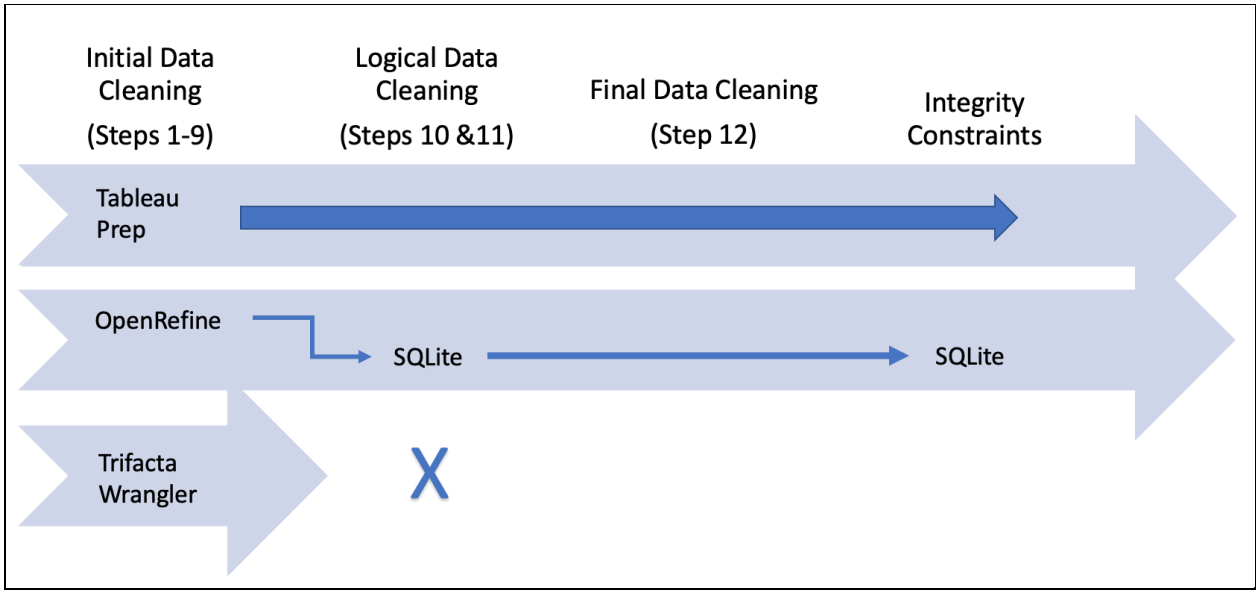
Data Cleaning Process and Details

Process Overview

To begin the project, we established a list of data cleaning steps. We outlined those steps in detail for use in parallel in all 3 of the applications, OpenRefine, Tableau Prep, and Trifacta Wrangler. We also created a list of what we termed 'integrity constraints'. These items were initially logical checks to ascertain if the data in each record was plausible. As an example, a sow could not be serviced (artificially inseminated) before she entered the herd. If the record showed that in fact her insemination date was prior to her herd entry date, we know that to be illogical and we would want to remove that record. As we worked further through the project we later determined that these items were not integrity constraints that assessed the quality of our data cleaning, but that they were, in fact, cleaning steps needed for this dataset for this use case. It was at this point that we added the logical cleaning steps to the previous cleaning list and then devised a few "true" integrity constraints to test our work.

Figure 1 shows the process that our work took in each of the 3 parallel workflows. It should be noted that Trifecta Wrangler had a size limit of 100MB for input data, which only captured a third of our original datafile. As such, we would not be able to accurately compare its results with the other programs we used, but could still compare the usability as compared to other programs. Therefore we decided to eliminate Trifecta Wrangler from the cleaning process after step 9, but continued the rest of the workflow using OpenRefine and TableauPrep. Also, when we got to our logical data cleaning steps, we made the decision to move the OpenRefine Workflow to SQLite within a Jupyter Notebook (details in Appendix C) as we found it easier to identify the logical inconsistencies in SQLite as compared to OpenRefine.

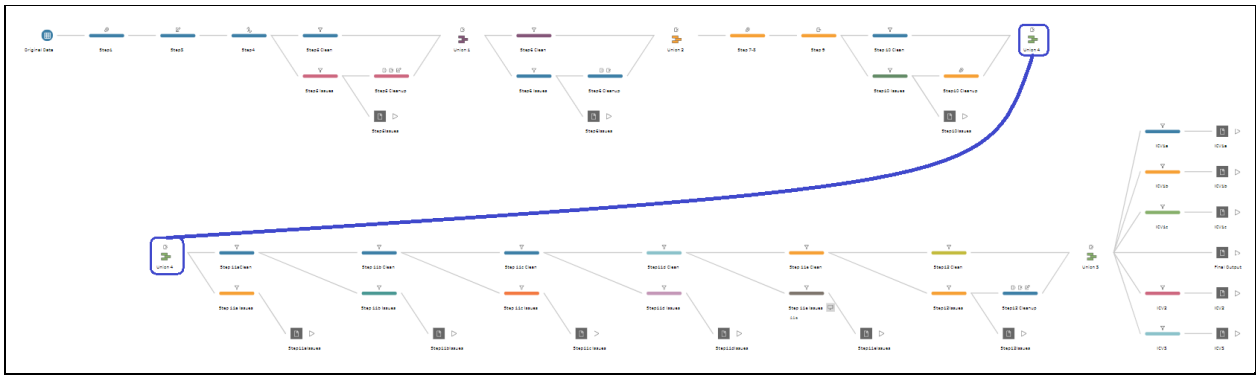
Figure 1: Data Cleaning Process Overview



A Few Specifics on Tableau Prep

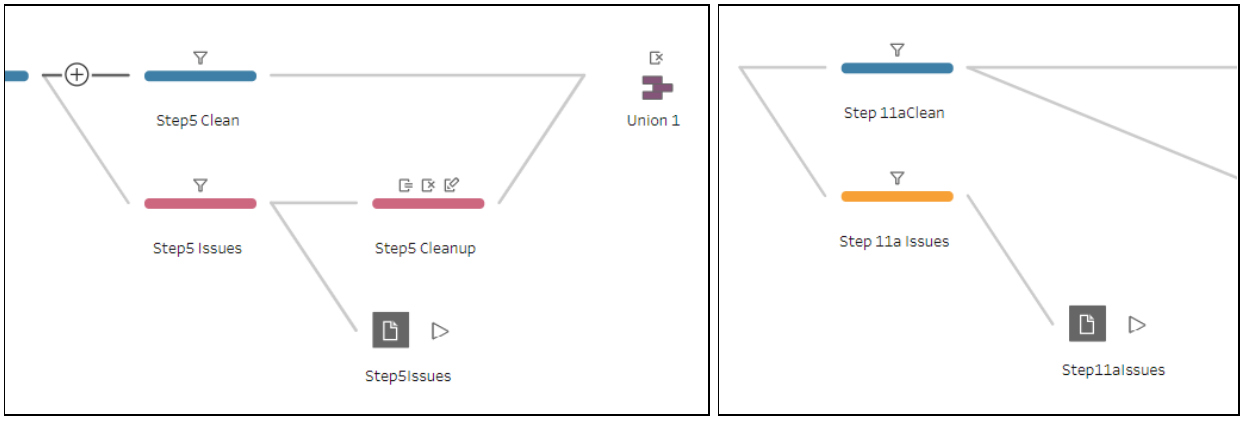
Tableau uses a graphical interface to represent the data pipeline like a flowchart with edges and various nodes. They coin a single flowchart where multiple datasets can be read in, processed, and outputted as "Flows" and are saved as .tfl files that can be shared and even inserted directly into other flows. Flows read left to right where the flowchart edges represent where the dataset is transferred to next, and the nodes represent certain operations on data (reading of files, performing selections/aggregations/etc, joining/unionizing from multiple datasets, or outputting the datasets to a file). Figure 2 shows the flow that we used to perform all the cleaning operations as well as extract all of our integrity constraint violations.

Figure 2: Entire tableau flow (blue line connect the same node)



Cleaning in Tableau Prep was performed in either three ways. Certain column wide operations like modifying the domain of attributes, changing names of columns, or replacing values were similar to OpenRefine, where Tableau Prep has direct options for those modifications. For cleaning steps which relied on modifying certain cells that did not meet our integrity criteria (Figure 3), we performed these operations in Tableau by branching the dataset into two separate paths where one path had a "filter" put on that only allowed values that needed to be cleaned and the other path did not. This allowed us to clean the observations that needed cleaning and then unionize them with the already clean observation to send them onto the next step. In addition, Tableau Prep's ability to output multiple files also allowed us to see which observations were originally dirty which we utilized as well. Lastly, for cleaning steps that required removals of the entire observations (Figure 3), the pathways were similar to what was previously discussed except we did not clean or unionize the dirty results. Once the cleaning was completed, we outputted the cleaned CSV file, as well as outputs corresponding to our integrity constraint violations. The output files are given in our Box-Folder, and select cleaning steps done with Tableau Prep are described in detail in Appendix D, as well as throughout the cleaning steps section where prudent.

Figure 3: Tableau operations to modify values (left), and delete observations (right)



Cleaning Steps: Detail and Rationale

To properly clean and process this dataset we performed the following cleaning steps. More specific details may be found in Appendix B.

1. [Farm]: Anonymize name to a 2 letter string.

Farm was anonymized primarily for use in this class, as our group did not want to report on the practices of specific farms. Although there may be a rationale to do so in our intended use case, we could not think of a reason to do so. We chose these 2 letter strings as they are memorable enough to viewers unfamiliar with the data to see that the same sow ID was represented multiple times in the same farm. This might be more difficult to discern if a series of numbers were used for instance.

2. Column management.

We found that [EntryAge] had no values in any of the rows and that [Own] had every row filled with the same value, ‘Y’. Initially we proposed to delete both columns. If this were the extent of the dataset it would make sense to remove these columns as they do not differentiate the rows in any way, but if we take a step back and look at the bigger purpose, it makes more sense to leave these columns in place, as is. This set of greater than 65,000 records is just one years records from a single system. The intent of the machine learning project is to be able to compile more records over time to be able to create dynamic real-time predictions of throughput. It could be possible that in 2016, this system did not use these fields, but in 2018 they might or perhaps this production system doesn’t use these columns, but another system does. We didn’t want to create

a workflow that may inadvertently lose information in the future because of a decision we made today.

3. Change column headings to remove spaces.

This cleaning step is debatable, but the intent was to remove spaces from headings as spaces in some programs can be problematic. Additionally, this allowed further standardization of the use of capitalization in headings.

4. 30 Columns to number; 9 Columns to date.

In OpenRefine, the CSV file reads in all of the cell values as text. This step is important for us to be able to change the column cell value type in order to make use of math and date comparisons, for instance. This is important in cleaning the data, but also in being able to use the clean data for its intended purpose. If 10 sows are served and you are trying to predict how many pigs they might have this next litter based on last litters performance, [LiveBorn] needs to be in a number format for an aggregate calculation like this.

5. [Technician1], [Technician2], [Technician3]

Expected values are 3 digit numbers that act as unique identifiers for an individual person on the farm or within the production system. However, there are values with less than 3 digits, values with more than 3 digits and values with characters “R” or “+”. The decision that we made to clean this section is to leave values with 3 digits, make any values with less than or greater than 3 digits or other characters as “Unk”, and leave blank any missing values. This differentiates an unknown technician, “Unk”, from not having a technician, “null”. Cleaning these fields may assist in machine learning if the algorithm begins to have more confidence in the outcomes from certain technicians compared to others. For instance the computer may find a different level of confidence in the service result from technicians with more services compared to fewer. There is one limitation to our cleaning effort; we do not know the true expected ID for the technicians. For example, it is possible that 202 is a technician in real life, but that several services were misentered as 220. 220 May be another actual technician at the time or 220 may not exist. Because we do not have that information we cannot clean the data further.

6. Clean service group

This value is used to ‘group’ sows physically within the farm, the sows may be penned together in a large pen or in a room. This is also used to assign weekly tasks such as “vaccinate 16-20’s” or “Pregnancy check 16-12’s”. In essence this is a farm-based shorthand for a cohort of sows

serviced in that week. Expected values based on this report of services in 2016 would be in the format YY-WW and between 15-53 and 16-52. We observed 9 records that do not meet this definition as that some values are outside of the range of 15-53 to 16-52 (2 records listed as 15-51 and 2 records listed as 17-52) and other values did not have all 4 digits (1 record listed as 16-201 and 4 records listed as 16-4). We decided to leave these values instead of replacing them individually because there are only 9 records among more than 60,000 values.

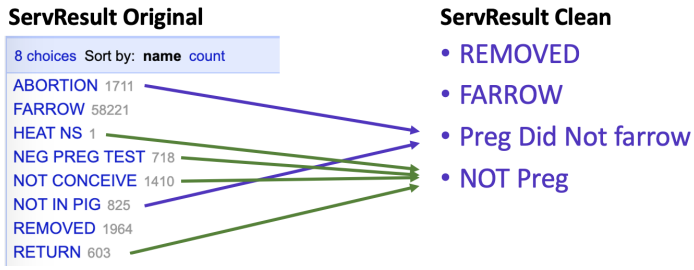
Initially this looks like an easy column to clean. The group is based on the service date, so perhaps we could create another column with the YY-WW based on the service date to make all of the values correct. The challenge to that solution is that the farm may start the first week of the year with the week that contains January first, or it may start the first week of the year with the first complete week of January. Further, they may opt to start the week on either Sunday or on Monday. With that understanding, we realized that a calculated "Service Group" is likely to not match the actual ServGroup used on-farm and even if we did make it match these farms, it is likely not to match future farms added into this workflow.

Then we reconsidered our use case. If this is a "hand entered field" for short-hand use on a farm, will it have any value in a dataset for a computer to make calculations from? It is likely that it will have minimal impact on algorithms as the algorithms would use the actual dates to be able to predict potential performance outcomes.and not this 'group' scheme. With that said, because it was minimally impacting we chose to leave the rows of data in the dataset, but to change the values of these 9 clearly incorrect service groups to "Unk". With this, the data for the sows service record remained with the data set, but at least for these 9 rows, the incorrect values for the service group could not be used.

7. Condense Service Result

We decided to condense the column ServResult. In the original dataset there were 8 values (Figure 4) and we condensed them down to 4 values that we believe would be meaningful in our machine learning use case.

Figure 4: ServResult Cleaning



FARROW refers to a sow that farrowed as a result of this service. There are 2 cases of sows that were pregnant but did not farrow: ABORTION means the sow aborted the litter between 35 and 110 days of gestation, and NOT IN PIG refers to a sow was found to not have pigs around the time of expected farrowing. There are 4 cases for a sow that was not pregnant as a result of the service: RETURN is a sow that returned or cycled as expected 21 days after this service and was bred again, NOT CONCEIVE also returned as expected but was not bred again, NEG PREG TEST is a sow found not to be pregnant by ultrasound at 28-34 days of gestation and HEAT NO SERVICE is a sow that returned and they chose not to service her this time and let her come back in heat again before breeding. Finally, REMOVAL is a sow that was removed from the herd with an unknown pregnancy status.

We narrowed down the results of the service to include FARROW, Preg Did Not Farrow, NOT Preg, and REMOVED to reduce any ambiguity for machine learning. It is not practical or necessary to differentiate a sow that was observed to lose her pregnancy, ABORTION, and a sow that went unobserved, NOT IN PIG. This is also true for those sows that did not become pregnant for several reasons. Instead, we felt it was practical to differentiate confirmed vs unconfirmed pregnancies and to further differentiate confirmed pregnancies as a successful farrowing and an unsuccessful farrowing while retaining REMOVED as an unknown result.

8. Clarify Removal Type

Here are clarifications for terminologies death, cull, and destroyed. Death literally means a sow died on the farm. Cull means a sow left the herd and went to “market” that typically enters the sausage market. Destroyed refers to an older, and “uncomfortable” term for animals with no hope of recovery, that were euthanized on-farm. We decided to replace ‘DESTROYED’ with ‘EUTHANIZED’ in order to soften the language and better reflect the true occurrence.

9. Clean FarrowDate

We noticed a problem with the FarrowDate column in that every service record, or row, has a farrow date, but not all of the service results are a farrowing. Table 1 below walks through an abbreviated set of records for 5 different individual sows in the dataset (depicted by color). We can see that the blue sow and the gray sow each have 3 services included in the dataset, that the result of each service was ‘FARROW’ and that the farrow dates relative to the service dates are all plausible. For the green and yellow sow, we find that the dates are also plausible when the service result is ‘FARROW’. In the case of the green, yellow and orange sow when the service result is not ‘FARROW’ we would then expect to have a null value for the farrowing date as we see with the null values in the ‘TotalBorn’ and ‘LiveBorn’ columns, but rather we are given the sows previous farrow date. It is not the case that this farrow date is wrong, as it was the date the

sow last farrowed, but it is not what we would expect and not a value that we want to leave in the data set for calculations and machine learning purposes.

Table 1 : Examples of the problem with FarrowDate

SowID	EntryDate	ServiceDate	ServiceGroup	Parity	ServResult	ServResultDate	FarrowDate	TotalBorn	Liveborn
W19367	2/2/14	1/26/16	16-04	5	FARROW	5/21/16	5/21/16	14	14
W19367	2/2/14	6/14/16	16-24	6	FARROW	10/7/16	10/7/16	16	15
W19367	2/2/14	11/4/16	16-44	7	FARROW	2/27/17	2/27/17	8	6
W19369	1/6/14	5/17/16	16-20	6	FARROW	9/10/16	9/10/16	18	18
W19369	1/6/14	10/7/16	16-40	7	REMOVED	1/9/17	**9/10/16**		
W19384	1/6/14	1/19/16	16-03	5	FARROW	5/16/16	5/16/16	18	18
W19384	1/6/14	6/10/16	16-23	6	REMOVED	8/5/16	**5/16/16**		
W19391	1/27/14	1/26/16	16-04	5	FARROW	5/18/16	5/18/16	15	14
W19391	1/27/14	6/17/16	16-24	6	FARROW	10/11/16	10/11/16	16	15
W19391	1/27/14	11/6/16	16-44	7	FARROW	3/1/17	3/1/17	16	16
W19392	1/6/14	1/22/16	16-03	5	NOT IN PIG	5/13/16	**12/25/15**		
	SERVED JAN-FARROW MAY, SERVED JUNE-FARROW OCT, SERVED NOV-FARROW FEB. => GOOD BELIEVABLE FARROW DATES.								
	SERVED MAY-FARROW SEPT, SERVED OCT… FARROWED IN SEPT? NO. SHE WAS REMOVED IN JAN. THIS IS HER PREVIOUS FARROW DATE.								
	SERVED IN JAN-FARROW MAY, SERVED JUNE… FARROWED IN MAY? NO. SHE WAS REMOVED IN AUG. THIS IS HER PREVIOUS FARROW DATE.								
	SERVED JAN-FARROW MAY, SERVED IN JUNE-FARROW OCT, SERVED NOV-FARROW MAR. => ALL GOOD BELIEVABLE FARROW DATES.								
	SERVED JAN-FARROW DEC OF 2015… NO. IN MAY SHE WAS NOT PREGNANT AT THE TIME OF HER EXPECTED FARROWING.								

For cleaning we decided that if the ServResult was Farrow, then there should be a farrowdate. If anything else, farrowdate should be ‘null’ and not her previous farrow date from her previous service. To accomplish this task in OpenRefine it seemed easiest to create a column called ‘REAL_FarrowDate’ based on the ServResult column with the GREL expression:

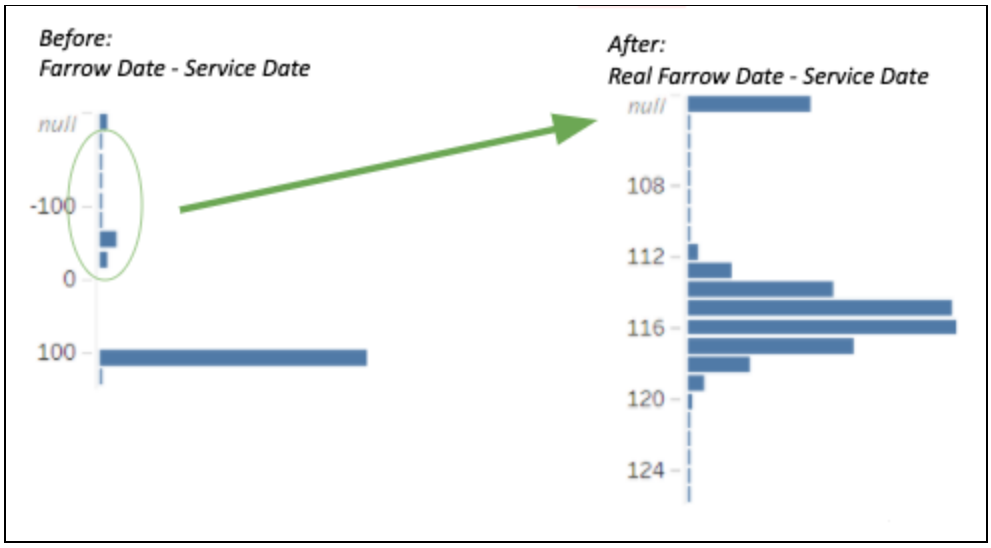
```
if(value.contains('FARROW'),cells['FarrowDate'].value,'')
```

In Tableau Prep this operation was performed similarly by creating the same column defined within the program as:

```
IF [ServResult] == "FARROW" THEN [FarrowDate] END
```

The Results of these changes can be seen in Figure 5. Before this new column was created, according to our dataset, a number of observations had a number of negative results for the difference between the farrowing date and service date. This change made the true farrowing timeline similar to what is expected - a distribution centered at 114 days.

Figure 5: FarrowDate Before and REAL_FarrowDate After Cleaning in Tableau Prep



10. True Abortions

Upon observation of the new distribution of REAL_FarrowDate (Figure 5, from TableauPrep in step 9), It appears that there are several sows at or less than 109 days of gestation. Generally, this is determined to be the cut-off for determining the difference between an 'abortion' and a farrowing. Upon further investigation, it was the case that several ServResults appeared to be misclassified as they farrowed very early in gestation and had no live born piglets.

In the OpenRefine workflow, a more complex logic was required to find and clean these records than was required previously, hence we decided to move our OpenRefine workflow to SQLite. Moving from OpenRefine to SQLite within a Jupyter Notebook is detailed in Appendix C.

In SQLite we created the query in Figure 6 and found sows misclassified as farrowed.

Figure 6: SQLite - True Abortions Query (subset of results)

```
%%sql
--Step 10 - Sows with True Abortions marked as Farrows
--(an abortion is defined as on or before 109 days of gestation and no liveborn piglets)
SELECT rowid, Farm, SowID, ServiceDate, REAL_FarrowDate,
    julianday(REAL_FarrowDate) - julianday(ServiceDate), LiveBorn
FROM servdata2
WHERE julianday(REAL_FarrowDate) - julianday(ServiceDate)<=109 AND Liveborn=0;
```

```
 * @/home/eejohnsn/Serv_data/servdatafin.db
 * sqlite:////home/eejohnsn/Serv_data/servdatafin.db
Done.
```

rowid	Farm	SowID	ServiceDate	REAL_FarrowDate	julianday(REAL_FarrowDate) - julianday(ServiceDate)	Liveborn
2991	CH	B677	2016-06-15T00:00:00Z	2016-09-30T00:00:00Z	107.0	0
7834	FR	Y183	2016-02-18T00:00:00Z	2016-06-06T00:00:00Z	109.0	0
8621	FR	Y4381	2016-03-21T00:00:00Z	2016-07-07T00:00:00Z	108.0	0

For these misclassified sows, we changed their ServResult from ‘FARROW’ to ‘Preg Did Not farrow’ and we changed the REAL_FarrowDate (created in step 9) to ‘’.

In Tableau Prep these changes were accomplished by first filtering out the observation which passed this expression,
`([REAL_FarrowDate]-[ServiceDate] <=109) AND [Liveborn]=0 AND [ServResult]="FARROW".`
These observations then have their ServResult replaced with “Pregnant did not farrow” and Real_FarrowDate replaced to NULL. They are then unionized with the observations that went though the opposite filter:
`([REAL_FarrowDate]-[ServiceDate] >109) OR [Liveborn]!=0 OR [ServResult]!="Farrow" OR ISNULL([Liveborn])`

11. Logical date orders

The following cleaning steps were performed to make sure none of our observations had any chronological inconsistencies. Observations not meeting the following criteria were removed.

a. EntryDate should be less than ServiceDate

As the entry date is the day the sow entered into the herd, it would not make sense if she had her first service before that day.

i. Incorrect if EntryDate>ServiceDate
ii. 0 results

b. ServiceDate should be equal to or 1 day less than ServDate2

Sows are services at least once and at most three times within a 2 day period. If the first service did not occur on the same day or a day before the second, our results have an integrity issue.

i. Incorrect if ServDate2 is not null AND ServDate2-Service Date >1
ii. 7 results.

Figure 7: SQLite - ServiceDate-ServDate2=0 OR1

```
%%sql
## the ServiceDate should be 1 day less than or equal to Servdate2
SELECT rowid, Farm, SowID, ServiceDate, ServDate2, ServDate3,
julianday(ServDate2)-julianday(ServiceDate) AS 'Dif2-Serv'
FROM servdata2
WHERE julianday(ServDate2) IS NOT NULL
AND (julianday(ServDate2)-julianday(ServiceDate)>1);
```

```
 * @/home/eejohnsn/Serv_data/servdata.db
 * sqlite:////home/eejohnsn/Serv_data/servdata.db
Done.
```

rowid	Farm	SowID	ServiceDate	ServDate2	ServDate3	Dif2-Serv
19146	KD	R3813	2016-07-03T00:00:00Z	2016-07-05T00:00:00Z		2.0
20301	KD	R6068	2016-09-05T00:00:00Z	2016-09-07T00:00:00Z		2.0
20876	KD	R5254	2016-10-07T00:00:00Z	2016-10-09T00:00:00Z		2.0
22033	KD	R6597	2016-12-10T00:00:00Z	2016-12-12T00:00:00Z		2.0
22052	KD	R6594	2016-12-11T00:00:00Z	2016-12-13T00:00:00Z		2.0
36296	SG	B22002	2016-08-05T00:00:00Z	2016-08-09T00:00:00Z		4.0
45731	SC	P3360	2016-07-03T00:00:00Z	2016-07-05T00:00:00Z	2016-07-06T00:00:00Z	2.0

c. ServDate2 should be = or 1day less than ServDate3

As before, if the second service date was not on the same day or a day before the third, our dataset has an integrity issue so we removed those observations.

i. Incorrect if ServDate3 is not null ANDServDate3-ServDate2 >1
ii. 0 results

d. The sow was served before she weaned her last litter.

This condition is put in as a sow can not be serviced before the day when she last weaned. As before we removed this observation.

i. Incorrect if ServiceDate < LweanDate
ii. 1 result

Figure 8: SQLite - ServiceDate<LweanDate

```
%%sql
-- IC:2 She was served before she weaned her last litter.
SELECT rowid, Farm, SowID, ServiceDate, Real_FarrowDate, LweanDate,
    julianday(ServiceDate) - julianday(LweanDate) AS 'DifLwean-Serv'
FROM servdata2
WHERE julianday(ServiceDate) - julianday(LweanDate)<0;
```

```
 * @/home/eejohnsn/Serv_data/servdata.db
 * sqlite:////home/eejohnsn/Serv_data/servdata.db
Done.
```

rowid	Farm	SowID	ServiceDate	REAL_FarrowDate	LweanDate	DifLwean-Serv
35655	PV	B10863	2016-12-02T00:00:00Z	2017-03-17T00:00:00Z	2016-12-05T00:00:00Z	-3.0

e. The Real_Farrrow date is less than ServiceDate

This constraint was also put into place as it would be impossible for a sow to farrow before she was even serviced. It should be noted that we discussed if we should instead place bounds on the difference between the service date and the true farrowing date as the time difference is centered at 114 days. After some lengthy consideration, we were unable to reason about the definite time differences we could safely reject, so we opted to choose the above constraint. We reasoned further that if our resulting dataset, or datasets cleaned by our pipeline, was ever used to do some machine learning predictions as discussed above, the algorithm would be trained to weight the confidence around an expected gestation length.

i. Incorrect if Real_FarrowDate-ServiceDate is less than ServiceDate
ii. 0 results in SQLite, and in Tableau Prep

12. If serveDateX is null then technicianX should also be null

As part of the servicing there should be a technician also associated with the service. We found issues where there were serve dates that were present but had a missing technician. Similar to Step 5, we found it most prudent to set the missing technicians as 'Unk' as it is not necessary to remove the entire record for this issue.

Lastly, it should be noted we did not see it prudent to introduce any official trimming of white spaces, as all the string operations we performed only replaced certain strings with others, which would not introduce whitespace as cleaning issues. We also took the position to not remove any observation with null cells as in our dataset, as null values do not present integrity issues. This is because, as we described above in a number of cleaning steps, the nulls provide an important role in our dataset to inform readers that an observation simply did not have a value for a certain attribute.

Results

Overall Changes to the Dataset

Table 2: Summary of cell changes to the dataset by each program

Step	Description	Tableau Prep	Open Refine	SQLite	Trifacta Wrangler[2]
1	Anonymize "farm'	65453	65453	N/A	21,748
2	Remove columns? NO	0	0	N/A	0

3	Adjust Headers	9	9	N/A	9
4	TO num, TO date	0[1], 496902	1441742, 496092	N/A	0,0[3]
5	Tech 1, 2, 3 to Unk	385, 214, 271= 870	385, 214, 271 = 870	N/A	N/A
6	Serv Group	9	9	N/A	9
7	ServResult from 8 to 4	5268	5268	N/A	2512
8	Removal reason hygiene	2785	2785	N/A	N/A
9	FarrowDate	7232	7232	N/A	709
10	True abortions	54	N/A	54	N/A
11a	Entry not less than service	0	N/A	0	N/A
11b	ServDate2-ServiceDate>1	7	N/A	7	N/A
11c	ServDate3-ServDate2>1	0	N/A	0	N/A
11d	Service<LweanDate	1	N/A	1	N/A
11e	REAL_FarrowDate<ServiceDate	0	N/A	0	N/A
12	ServeDateX ↔TechX (Service1, 2, 3)	98, 18, 0=116	N/A	98, 18, 0=116	N/A
Total Cells Modified		573,421	2,019,460	54	24,987
Total Records Deleted		8	0	8	0

1. Tableau Prep automatically imported those attributes as numerics.
2. Because of the free trial, the file is limited to 100MB and cleaning is restricted. It also shows 21,748 rows instead of 65453 rows.
3. Trifecta Wrangle automatically imported the described attributed to numerics and dates correctly

Cell modification and record deletion were equivalent for SQLite, OpenRefine, and Tableau Prep, which should be expected as the dataset was the same, and the operations we performed were synonymous on each of these programs. Owing to the size limit of our input data for Trifecta Wrangler, we had a smaller number of results found in our cleaning processes.

Data cleaning program comparison

While cleaning our dataset, we explored three different software programs which were recommended from project guidelines. We observed several pros and cons for each tool as we used it for our project, with our machine learning use case in mind.

- OpenRefine:
  - Pros:
    i. The program is free
    ii. Entire cleaning steps can be extracted to a JSON File. This is especially useful for users to easily share their processing pipeline to other users to read and use someone's pipeline without even needing to enter the OpenRefine program. This makes provenance easy to share and the data cleaning easy to repeat.
    iii. The user interface is pretty intuitive and someone can understand much of the functionality within a day.
  - Cons:
    i. To test integrity constraints, users must first export their cleaned data into a CSV and then import it to a database to check using SQL, datalog, etc.
    ii. If you want to edit a cleaning step within OpenRefine, the program automatically deletes any steps that follow. It can be extracted, but it is not convenient.
    iii. The use of GREL to create an expression is a nice feature, but editing a GREL expression is not possible, which makes the trial and error process, particularly if you are less familiar with GREL, challenging.
    iv. We found that using OpenRefine was very challenging for finding our "illogical" records and then repairing them. Instead we opted to shift our workflow for those steps to SQLite.
        1. Note: that is also a cumbersome process in removing the clean data, establishing a new database, creating a table and importing the clean data into the database. A connection to the database must then be established in the Jupyter notebook for use.

- Tableau Prep
  - Pros:
    i. The graphical interface is extremely user friendly. After just a 5-minute video introduction to the software, we were comfortable navigating the basic features, and the more advanced features could generally be found easily within the supporting documentation.
    ii. Frequently used in industry
    iii. Large support community both within and outside the company

- Cons:
    i. The program is quite expensive if you don't work for an organization with available keys. It can be up to $70/month to utilize Tableau Prep, and $12/month to even view "flows" within the program.
    ii. Issues with Provenance. Most of this issue comes from the high price to view and modify the flow. Moreover, Tableau Prep does not provide a JSON-like file that users can pass in between each other to view and modify an abstraction of the data flow.
    iii. It was interesting that although Tableau Prep has a set-union operation, it does not have the set-not or set-intersection operations. Although there are other methods to perform these operations, it seems to be an oversight to not include at least one of them.

➢ Trifacta Data Wrangler
- Pros:
    i. Free, for small datasets (less than 100MB)
    ii. Provides a preview of faceted data automatically
    iii. Shows steps of cleaning history
    iv. Shows how many each unique values and patterns are in each column
- Cons:
    i. Price $419/month,
    ii. Similarly as Tableau Prep, Trifacta Data Wrangler suffers from a provenance perspective due to its price.
    iii. Similar to OpenRefine as it requires exporting CSV and then using SQL for IC's
    iv. Use of free software is limited for 100MB

To summarize our group's opinion, we found TableauPrep to be the most useful tool as all operations of our pipeline could be done with such ease that it outweighed the issues of the large price tag and provenance. It should also be noted that as we were only working with the free version of Trifecta Data Wrangler, we are likely missing out on many features, like a direct connection to relational databases, and tools for users to collaborate directly with each other on the same workflow.

Integrity Constraints

The following integrity constraints validate steps 5, 9, and 10 of our cleaning process.

1. If serviceDateX is null then Tech(X) should also be null
    a. Technician1 is null thus, ServiceDate is null
    b. Technician2 is null thus, ServDate2 is null

c. Technician3 is null thus, ServDate3 is null

2. Real_FarrowDate null implies ServResult not ‘FARROW’
3. ([Real_FarrowDate]-[ServiceDate] <=109) AND [LiveBorn]=0 AND [ServResult] ==’Preg Did Not farrow’

We opted to further test our procedures by taking subsets of our data and modified them to fail these checks (Messy_Data.csv). The results of these queries are shown below.

1a:

```
%%sql
SELECT *
FROM final
WHERE Technician1 IS NULL AND ServiceDate IS NOT NULL
```

Figure 9: SQLite IC check 1a

index	Farm	SowID	EntryDate	EntryAge	Entry1stServInt	P0HNSCount	ServiceDate	ServDate2	ServDate3	Technician1	Technician2	Technician3	Servi
0	CH	B5983	2015-12-15T00:00:00Z	None	17	0	2016-01-01T00:00:00Z	2016-01-01T00:00:00Z	None	None	346.0	None	
1	CH	B7995	2015-12-22T00:00:00Z	None	10	0	2016-01-01T00:00:00Z	2016-01-01T00:00:00Z	None	None	312.0	None	

1b:

```
%%sql
SELECT *
FROM final
WHERE Technician2 IS NULL AND ServDate2 IS NOT NULL
```

Figure 10: SQLite IC check 1b

D	EntryDate	EntryAge	Entry1stServInt	P0HNSCount	ServiceDate	ServDate2	ServDate3	Technician1	Technician2	Technician3	ServiceGroup	Parity	Se
9	2015-08-04T00:00:00Z	None	12	0	2016-01-01T00:00:00Z	2016-01-01T00:00:00Z	None	446.0	None	None	15-53	1	
6	2015-06-16T00:00:00Z	None	58	0	2016-01-01T00:00:00Z	2016-01-01T00:00:00Z	None	476.0	None	None	15-53	1	

1c:

```
%%sql
SELECT *
FROM final
WHERE Technician3 IS NULL AND ServDate3 IS NOT NULL
```

Figure 11: SQLite IC check 1c

SowID	EntryDate	EntryAge	Entry1stServInt	P0HNSCount	ServiceDate	ServDate2	ServDate3	Technician1	Technician2	Technician3	ServiceGroup	P
Y27702	2015-07-28T00:00:00Z	None	17	0	2016-01-01T00:00:00Z	2016-01-01T00:00:00Z	2016-01-01T00:00:00Z	412.0	346.0	None	15-53	
Y24658	2015-08-04T00:00:00Z	None	10	0	2016-01-01T00:00:00Z	2016-01-01T00:00:00Z	2016-01-01T00:00:00Z	476.0	312.0	None	15-53	

2:

```
%%sql
SELECT *
FROM final
WHERE REAL_FarrowDate IS NOT NULL AND ServResult != "FARROW"
```

Figure 12: SQLite IC check 2

/Date3	Technician1	Technician2	Technician3	ServiceGroup	Parity	ServResult	REAL_FarrowDate	ServResultDate	ServNBoars	ServNMates	ServNo	Farrow1st
None	212.0	346.0	None	15-53	1	Preg Did Not Farrow	2016-04-25T00:00:00Z	2016-04-25T00:00:00Z	1	2	1	
None	276.0	346.0	None	15-53	2	Abortion	2016-04-27T00:00:00Z	2016-04-27T00:00:00Z	1	2	1	

3:

```
%%sql
SELECT *
FROM final
WHERE ServResult = 'Preg Did Not Farrow' AND REAL_FarrowDate IS NOT
NULL
```

Figure 13: SQLite IC check 3

Technician2	Technician3	ServiceGroup	Parity	ServResult	REAL_FarrowDate	ServResultDate	ServNBoars	ServNMates	ServNo	Farrow1stServInt	WeanServInt
346.0	None	15-53	1	Preg Did Not Farrow	2016-04-25T00:00:00Z	2016-04-25T00:00:00Z	1	2	1	25.0	4.0

We then passed these queries into our cleaned dataset, either with SQLite or with Tableau Prep, and found that we had no integrity constraint violations (CSV outputs of the ICV using Tableau Prep are given in our Box Folder).

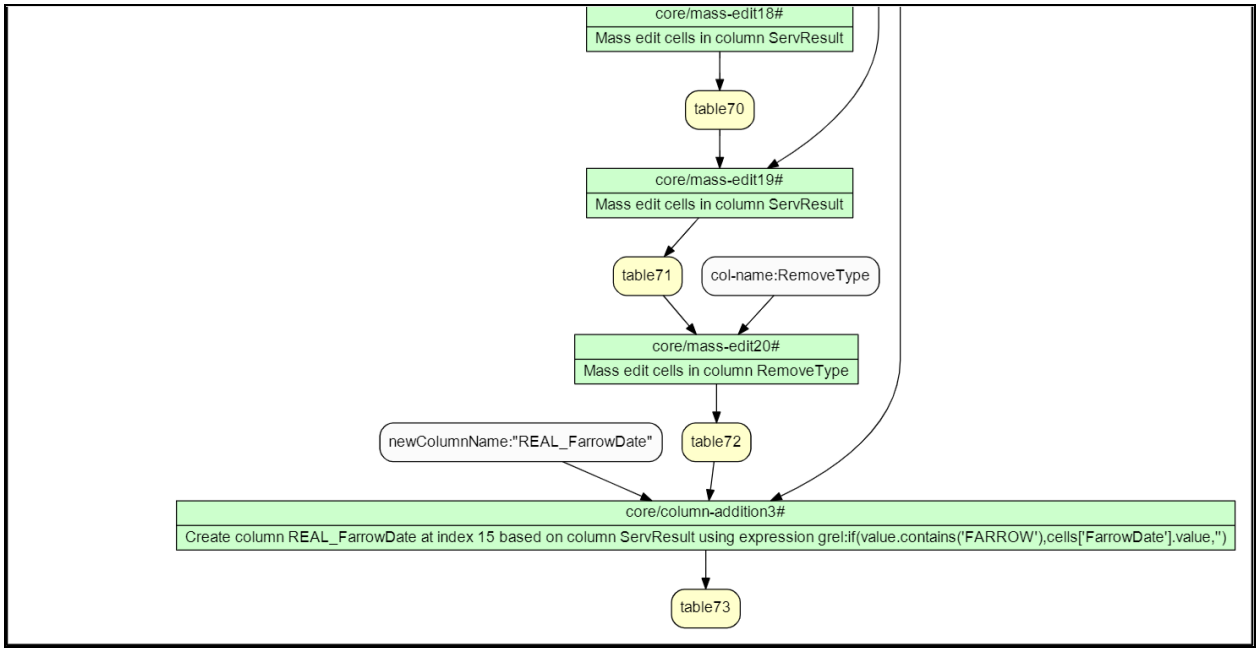
YesWorkflow Model

We used the JSON file from OpenRefine to create a YesWorkflow graph. This was not done using the TableauPrep cleaned data, due to the ability of the Tableau software to handle integrity constraint checks within the program.

Our YesWorkflow model ended up being very large, so much so that we did not want to insert a screenshot of the entire workflow into the report for fear of interfering with the formatting. Thus, we have provided the full YesWorkflow PDF, in both linear and parallel form, in the box folder for our supplementary materials. We created our YesWorkflow using python in the command line and GraphViz to create the PDFs from the JSON file obtained after data cleaning in OpenRefine. It is important to note however that this .JSON file does not include any of the cleaning items past step 9.

The key input for this workflow was the dirty data .CSV and you can see by the green boxes in Figure 14 the outputs of each individual cleaning step. The key output was just a singular cleaned data file, with new columns created, changed value types, and GREL filtering. Our workflow was very linear(which can be seen in Appendix B), and almost all of the steps could have been done in any order. The only dependencies is step 7, cleaning of the ServResult column, needed to be completed before step 9, creation of REAL_FarrowDate, to accurately create the new column. A snapshot of the overall workflow can be seen below, showing step 9, the creation of REAL_FarrowDate.

Figure 14: A Portion of the Yes Workflow Output, using the OpenRefine JSON



Conclusions and Future Work

Overall, we believe that we have made significant strides to improve the quality of the dataset for use in machine learning and artificial intelligence. Our group agrees that after developing and executing our cleaning steps, the data was not as dirty as we originally thought. If we think back

to the OpenRefine homework assignment, that dataset had many issues and we were able to use many different features of OpenRefine for data cleaning. Here, the dataset looks very different from that example. As discussed in our initial dataset assessment, this is likely due to the constraints that the production system software likely has already in place. We can't fix problems of human error if the service date is not null, but the technician is missing, do we have to assume the date was entered incorrectly, or that they forgot to add the technician? We can guess that that's the issue, but how do we deal with it? There may be other valuable data in that record. We had many discussions of when to change the value of a cell, to "Unk" for instance, and when it was appropriate to remove the record entirely.

We believe that the researcher using this dataset for machine learning will notice the changes we made to the dataset, and appreciate the detailed documentation we created to use in whole, or in part to create a realistic workable workflow in the program of the researcher's choice.
Beyond cleaning the dataset, we compared three data cleaning software programs: OpenRefine, TableauPrep, and Data Wrangler. Through our testing, our group reached the consensus that TableauPrep is the better software for creating a data cleaning pipeline. OpenRefine and Data Wrangler are great programs, but integrity constraints can not be checked or fixed directly in the program, instead, requiring the cleaned dataset to be added into a SQL/Datalog database for testing. In addition, Data Wrangler's free version only has 100MB of space, which can be an issue for larger datasets, which makes it slightly worse than OpenRefine in our eyes. However, Data Wrangler does provide some nice graphs to visualize the data entered into the software. While TableauPrep has a significant cost, it provides the most extensive data cleaning experience and workflow development within a single program. We acknowledge that it lacks somewhat in that it is not possible to extract the workflow process in an easily accessible and re-creatable way, which has implications on provenance. This may be detrimental if the machine learning process and techniques will be published in the scientific literature with the intention of other researchers to be able to repeat the work. On the other hand, it may be an advantage if the machine learning process and techniques would be used by a firm selling the results as a service, where they would want to keep the provenance private.

The biggest challenge for our group was in understanding integrity constraints. We started with a list of nine, but after much discussion determined that most were not actually integrity constraints. Instead, these were logical filtering of the data and not integrity constraints as we learned in class. After a group discussion, we identified actual integrity constraints, for testing with SQLite. This was the biggest challenge for us, as it led to confusion between group members on what cleaning steps needed to be completed before integrity constraints were checked.

The next logical step for this project would be to hand our dataset back to the graduate student for his opinion and critique of our work. It may also be interesting to see some of the initial algorithm prediction results and the visualization of those predictions using the original dataset as compared to our clean dataset. Overall, we're proud of the work we have accomplished for this project.

References

[1] ”Queensland pig industry: Industry terms and definitions.” *Queensland Government, Department of Agriculture and Fisheries,*
https://www.daf.qld.gov.au/business-priorities/agriculture/animals/pigs/industry/terms-definitions

[2] Silva, G.s., et al. “Monitoring Breeding Herd Production Data to Detect PRRSV Outbreaks.” Preventive Veterinary Medicine, vol. 148, 2017, pp. 89–93., doi:10.1016/j.prevetmed.2017.10.012.

Accompanying Files in BOX

https://uofi.app.box.com/s/ickkd0oi259tgh5888zkdxd4f4pwqryt

Resources	**File**
Appendix A: ServData Key	Appendices.zip
Appendix B: Cleaning Steps	Appendices.zip
Appendix C: CSV to SQLite steps	Appendices.zip
Appendix D: Tableau Query Details	Appendices.zip
Yesworkfow serial/parallel file	YesWorkFlow.zip
Yesworkflow linear/parallel graphs	YesWorkFlow.zip
All files needed to check ICs on Jupyter notebook with dirty data	IC Checks.zip
OpenRefine_SQL clean output file	OR_SQLClean.csv
Jupyter Notebook for SQL to .csv	csv_out_file creation.ipynb
Jupyter Notebook for Steps 10-12	Servdata Steps 10&11&12.ipynb
OpenRefine midpoint steps 1-9 clean output file	ServDateCleanOR1to9.csv
OpenRefine midpoint steps 1-9 JSON	Clean1to9redo.rtf
SQL database file: servdatafin.db	servdatafin.db
Tableau Outputs except for cleaned dataset	TableauOutputs.zip
Tableau Flow	TableaFlow.tfl
Tableau Prep clean output file	TableauPrepClean.csv
Tableau Prep flow zoomed	TableauFlowZoom.pdf
Original Dirty ServData .csv:	Original.csv