Cleaning **O**n **De**mand with a **R**ecommender **S**ystem (CODeRS)

Ashley Schuliger and Christopher Vieira

March 3, 2022

A Major Qualifying Project submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the degree of Bachelor of Science

Authors:

Ashley Schuliger

Christopher Vieira

Date:

March 3, 2022

Report Submitted to:

Professor Rodica Neamtu

Worcester Polytechnic Institute

Acknowledgements

We would like to extend our sincerest gratitude to our MQP advisor, Rodica Neamtu. Her support, guidance, and expertise in the field motivated and encouraged us to produce our best possible work.

We would also like to express our deepest gratitude to our MQP co-advisor, Danielle Cote, for her participation and insightful feedback during our user studies. Without feedback from such a powerful expert in the Materials Science field, our project would not have reached and exceeded its potential.

Additionally, we would like to express our appreciation for Jack Grubbs and Matthew Gleason from the Materials Science department for providing their feedback on the functionality and design of our application during user studies. Their valuable feedback provided us avenues by which to improve our application far beyond our expectations.

Abstract

Data Science has become a popular tool for deriving solutions in a variety of domains, such as medicine, materials science, and finances. The performance of such data science applications depends on how well the data is cleaned and preprocessed. However, choosing the correct techniques can be a difficult task as this decision depends highly on the data itself. Thus, the automation of this process could greatly benefit those using data science by reducing human error and, in turn, creating more reliable and generalizable predictive models.

This project aims to develop an automated cleaning and preprocessing web application for non-technical users called CODeRS. We performed a literature review to find state-of-the-art techniques with a range of assumptions to recommend in our application. We implemented those techniques in an automated recommender system that provides the appropriate cleaning technique based on the dataset. Additionally, we developed a graphical user interface to simplify the user experience for those creating data science solutions in a differing domain. Furthermore, we developed this application in a modular fashion to ensure scalability, longevity, and flexibility. We deployed our final product and measured its functionality and design in a series of user studies with a group of materials scientists.

Table of Contents Name

Table of Figures

Table of Tables

1 Introduction

The recent and periodic increases in emerging technologies worldwide have led to the mass generation of data in a variety of domains [1], [2], [3]. This increase in available data has motivated experts in various fields to utilize data science solutions to derive insights and solve problems. However, a majority of real-world data often contain errors created during data collection. This may occur due to sensor failures, human error, or errors in the data collection process itself. If left untreated, dirty data can adversely affect the outcomes of data analyses and cause data-driven solutions to be inaccurate and unreliable. There are a variety of ways to clean and correct data errors. However, for experts in domains outside of data science, these techniques can be quite complicated to implement or understand. As a result, datasets cleaned by non-data scientists are often cleaned incorrectly or with a technique that introduces bias to the dataset. Thus, there is a need for an application that makes the cleaning process easy and understandable for users outside of the data science field.

1.1 Motivation

Many fields rely on data being properly cleaned and preprocessed prior to its use in analysis. However, the process of cleaning this data is not always trivial. Currently, data cleaning has yet to be standardized in many fields, leaving experts to discover cleaning techniques for themselves. Unfortunately, many state-of-the-art cleaning techniques are computationally complex and not suitable for non-data science fields. This lack of automation for ideal cleaning techniques leads domain experts to utilize simple, yet biased, methods. The materials science and medical fields exemplify these problems as outlined below.

Materials Science Example

In the field of materials science, processes, such as Nanomechanical Indentation and Nanomechanical Mapping, are utilized to collected large amounts of data. Nanomechanical Indentation involves applying a varying force onto an object using an indenter tip. Measurements on properties such as material hardness, load on the material, and the depth into the material are then recorded at an interval in milliseconds. Similarly to Nanomechanical Indentation, Nanomechanical Mapping collects a series of indentations in a grid format. A data tuple containing hardness, modulus, and depth values is collected for each indentation point. These two forms of data collection often result in massive datasets filled with null values and outliers. Analyses using such data is often inaccurate since large areas containing nulls cannot be analyzed, and thus, are not represented. Additionally, outliers are problematic because they lead to bias and produce incorrect analyses regarding the material altogether. If left uncorrected, materials being analyzed may appear to be more robust than they are. This can lead to significant issues in real-world applications where the material is used. Due to the massive size of such datasets, correcting these values by hand can take hours for a single dataset if an automated and standardized process is not used.

Medical Example

Predictive modeling is often used in the medical field to minimize risk. One crucial step in generating these predictive models and minimizing risk overall is cleaning the data. If data is not properly cleaned, data analyses using this dataset may have significant implications on patient health. In cases where predictive models are used to provided diagnoses, poor data analysis techniques may even lead to the loss of human life at worst [4]. Reducing this risk results in benefits such as improved diagnostics, high cost-effectiveness, enhanced operational efficiency, decreased remission rates, personalized medical care, and more.

Data cleaning is also beneficial in correcting raw data generated through studies in the medical field. One evolving area for this is clinical epidemiological research [5]. In this research, data cleaning is used to identify and correct errors to minimize their impact on the results of a study. Due to the relatively recent usage of data cleaning in this area, minimal guidance is currently available in peer-reviewed literature on how to set up and carry out cleaning efforts efficiently and ethically.

In order to address these problems, the goal of this project is to create an automated application called CODeRs to handle common anomalies in real-world data. Such anomalies include null values and outliers. Our application provides multiple options for handling anomalies in the dataset, including removing, ignoring, or correcting them. Additionally, our application recommends the appropriate cleaning technique to correct errors in a dataset to simplify the process for our users. Through literature research, we determined state-of-the-art detection and correction techniques, such as Multiple Imputation by Chained Equations and DBSCAN, for the anomaly values mentioned above, and we implemented them in the correction option defined in our application. This function allows non-data scientists to clean their data accurately without the difficulty of interpreting various anomaly detection and correction techniques found in top Computer Science and Mathematical conferences.

1.2 Project Goals

To create a tool most suitable to provide an easy-to-use service for cleaning and correcting data, we laid out the following objectives.

- Implement several state-of-the-art cleaning techniques including but not limited to outlier detection and null correction through research on relevant methods.
- Create a recommender system to automate the process of correcting nulls and detecting outliers in a dataset.
- Create a GUI to enable non-data scientists to clean their own datasets using our tool.
- Employ proper automated documentation, automated testing, and development practices throughout the project to ensure the longevity of the open-source project once our team has completed our MQP.

Through the completion of these objectives, we hope to provide an automated solution for data cleaning.

2 Background

In this chapter, we further establish the necessity for an open-source data cleaning application with an automated recommender system. We discuss the data gathering and collection process along with the properties of a dataset. Additionally, we discuss various cleaning techniques presented in literature and examine their credibility. After providing the necessary data science and statistical background, we examine existing platforms for data cleaning to determine the need for our application.

2.1 Properties of a Dataset

To make a recommendation on the proper cleaning techniques to use for a dataset, its attributes must first be defined and considered. Some important attributes of a dataset are its statistical properties. The distribution of a dataset can be represented by several metrics, such as the standard deviation, mean, and range of a dataset. The mean represents the average value of the dataset, the range represents the full spread of the dataset, and the standard deviation represents the average spread of a dataset. These values provide insight into the distribution of a dataset. However, they are highly non-resistant to outliers, with a single outlier being enough to highly affect their values. More resistant properties are the median and interquartile range. The median value is the value of the second quartile, while the interquartile range is the difference between the first and third quartile. These values also provide key insights into the distribution of a dataset while being much more resistant to a skew.

Another dataset attribute that is especially important for replacing missing values is the relationship between the points in the dataset. In a situation where all points are directly related to one another, such as a time-series, there is a natural order to the data point. If a value between two known data points is missing, it is possible to use some method or model, such as linear regression, to try to determine what this missing value would be. This may be represented by a time-series of temperature readings where we know the value must move in a continuous pattern. On the other hand, if the points are not sequential, a model could not be used to replace the value based solely on surrounding values. It instead may have to consider all of the data points for that column or the values for that record in the other data columns. An example of this may be temperature readings recorded in Fahrenheit and Celsius at random times and at random locations. If one value is missing but the other still exists, a very basic transformation can be applied to retrieve the original value.

One of the more complex attributes of a dataset is the number of data columns it contains. If the dataset contains a single list of data (i.e. one-dimensional), the dataset is univariate. An example of this could be temperature readings where only the degrees in Fahrenheit is recorded. It is also possible for data to be n-dimensional or multivariate. Such a dataset contains two or more columns. For example, a temperature reading could include the degrees in Fahrenheit, the time the reading was collected, and a wind reading. Depending on the dimensionality of the data, different cleaning and preprocessing methods may be more appropriate.

In addition to the above mentioned properties, the type of the dataset, such as point-wise and time-series, also plays a significant role. A point-wise dataset is the most common type of dataset generated. The order of the points does not matter, and they are not related to one another in any sequential form. For example, if we gathered twenty random temperature readings and only kept track of the temperature, this would represent point-wise data. Unlike point-wise datasets, time-series datasets' points are related to one another. In time-series, data values are gathered in some sequential fashion, similar in nature to a linked list or an array. The order of the values matters relative to the time, index, or other metric used to keep track of the order of the values.

Unlike point-wise datasets, time-series datasets have additional properties used to define their behavior. These properties include seasonality, trend-cycle, and remainder components [6]. To highlight the differences between these components, we created a sample data found at our GitHub repository. This dataset was created using a seasonality component, a trend-cycle component, and a residual component. Several outlier values were injected into the dataset as well. The visualization of these values can be seen below in Figure 1.

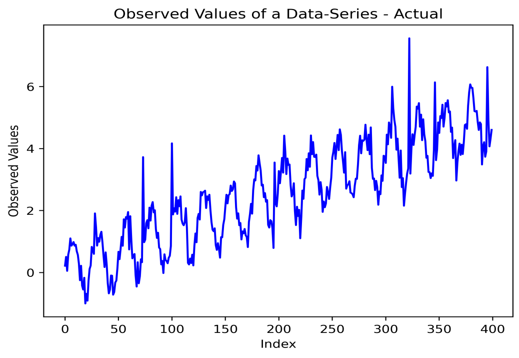


Figure 1: Time-Series Raw Data

The seasonality of a time-series represents the repetition in a time-series in a cyclical behavior. For example, this may be represented as retail prices being higher in November and December, following a seasonal cycle [7] [8]. The seasonality component of the dataset above can be shown below in Figure 2.

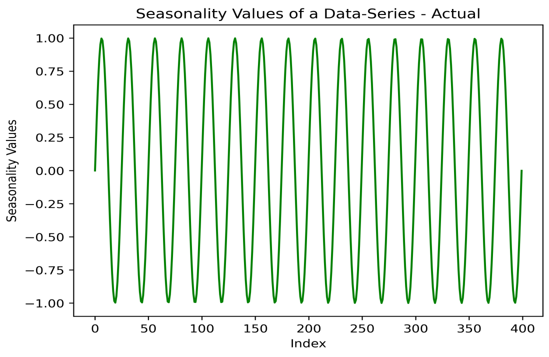


Figure 2: Time-Series Seasonality Component

Seasonality is commonly removed to determine long-term trends or errors. The second component mentioned is the trend-cycle component. While the seasonality represents repeating patterns that reoccur every x units of time, the trend-cycle represents the long-term behavior of a time-series [9]. The trend-cycle component of the dataset for this example can be seen below in Figure 3.

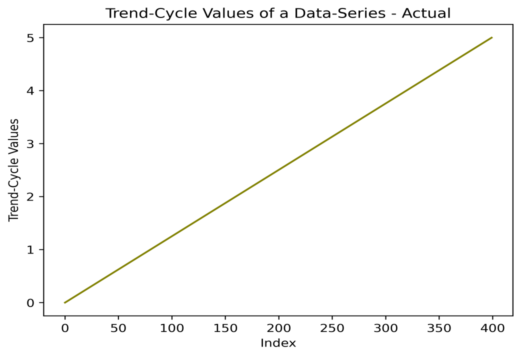


Figure 3: Time-Series Trend-Cycle Component

The last component of a time-series is the residual component. This value is simply the remainder after removing the seasonality and trend-cycle components of the time-series values. An example of this can be seen below in Figure 4. Outliers are especially apparent as they have larger residuals.

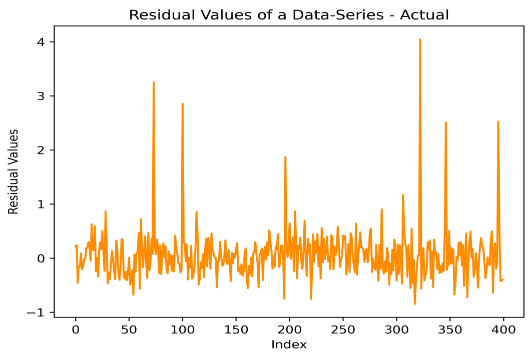


Figure 4: Time-Series Residuals Component

2.2 Cleaning Techniques

2.2.1 Outlier Detection

One of the major themes in data cleaning revolves around detecting and treating outliers. Possible outlier treatments include removal, replacement with an interpolated value, or replacement with a constant. Regardless of the treatment type, they must first be detected using some form of outlier detection technique. The baseline outlier detection methods implemented for this project are univariate statistical methods and multivariate clustering methods. Univariate outlier detection methods compare a single column of data to determine outliers. Multivariate methods discussed later in this section use two or more columns to determine if a value is an outlier or not.

One of the most common algorithms for detecting outliers is the z-score outlier detection method. This technique makes use of two metrics from a univariate data distribution: the mean represented as $\bar{x}$ and the standard deviation represented as s from a data distribution x [10]. An observation is considered an outlier if it is outside of the interval defined below in Equation 1.

$$(\bar{x} - ks, \bar{x} + ks) \tag{1}$$

The value k used in the interval above represents the number of standard deviations away from the mean a data point must be to be considered an outlier. The value of k is typically 2 or 3. This method is intended for use on normal distributions, where a k of 2 captures 95% of the data and a k of 3 captures 99.7% of the data as stated by the Empirical Rule [11]. A visual representation of this can be seen in Figure 5 below.

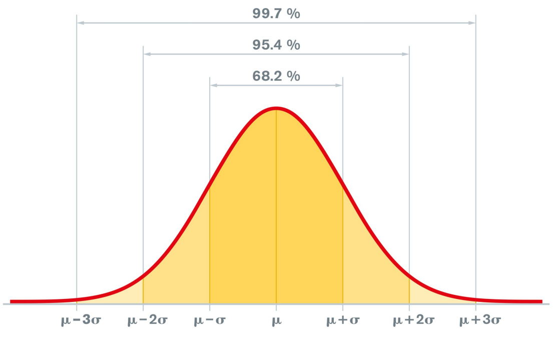


Figure 5: Empirical Rule

The equation for z-score defined above defines a set interval. However, it does not provide an indication of how a data point compares to another data point within a dataset. For example, if a data point is twenty standard deviations away from the mean, it has a stronger impact on the dataset than a data point three standard deviations. Using the interval method, a point is determined to be an outlier or not with no variation between the classification. There are several other methods to represent the above equation and do this, with one popular method shown in Equation 2 below.

$$\frac{|x-\bar{x}|}{s} > k \tag{2}$$

This equation calculates how many standard deviations the data point x is away from the mean and then compares it to a constant k. If the value is above k, the value is determined to be an outlier. Variations of this method also exist. One such method relies on another constant, α, which represents a confidence coefficient where $0 < \alpha < 1$ [12]. The α-outlier region of a normal distribution with mean $\bar{x}$ and standard deviation s can be represented by Equation 3 below.

$$out(\alpha, \bar{x}, s^2) = x : |x - \bar{x}| > z_{1-\alpha/2} * s \tag{3}$$

In this equation, z_q represents the quintile of the N(0,1). Similar to the above equations, if x is included in $out(\alpha, \bar{x}, s^2)$, it is defined as an outlier. For simplicity, this project uses the second equation as it is the most commonly used in literature. The results of utilizing this process on a generated dataset are shown below in Figure 6. The generated dataset can be found our GitHub repository.

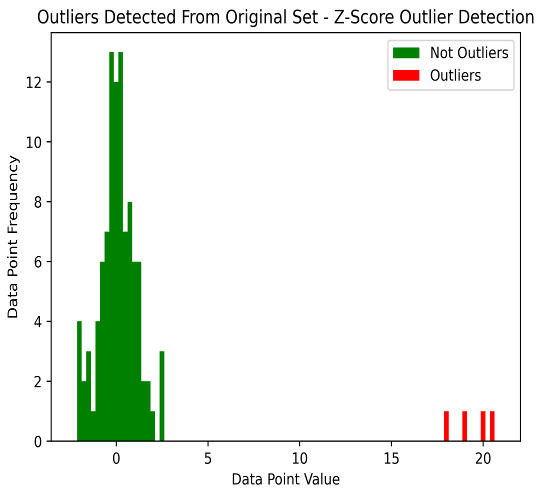


Figure 6: Normal Distribution with Outliers Identified Using Z-Score

The purpose of the z-score outlier detection method is to detect outliers. However, the way it determines its metrics is flawed. The two metrics that it uses, the mean and standard deviation, are highly susceptible to skew. If we follow the Empirical Rule of the normal curve, we can see the data is distributed in a continuous and decreasing frequency away from the mean. The issue with this is that some outliers do not follow this trend. For example, an outlier can be magnitudes larger than the data distribution itself. Consider the example located at our GitHub repository where a normal set of 100 values is generated with a mean of 0 and a standard deviation of 1 using NumPy. When we calculate the mean and standard deviation, we receive a (mean, standard deviation) of (0.07941666293687392, 0.9670394771137057). Now consider if an outlier with a value of 10000 is inserted into the dataset. This value skews the mean and standard deviation significantly, resulting in the new (mean, standard deviation) value of (99.08853134944245, 990.091614451334) as shown in Table 1. A visual representation of the results of the z-score outlier detection can be shown in Figure 7. When z-score is used to identify outliers on this new dataset, the results are inaccurate since the statistics used to identify outliers no longer accurately identify the dataset. With these values, a point at 2000 would be considered not an outlier, even though any data scientist would deduce it to be one. Due to this issue, a better outlier detection technique that is not heavily impacted by outliers should be used.

Table 1: Table highlighting effects of Outlier Insertion

	Mean	Standard Deviation	Median	Interquartile Range
Before	0.07941666293687392	0.9670394771137057	0.09436841421237988	1.1561178267491579
After	99.08853134944245	990.091614451334	0.09914921583524362	1.1902531587620824

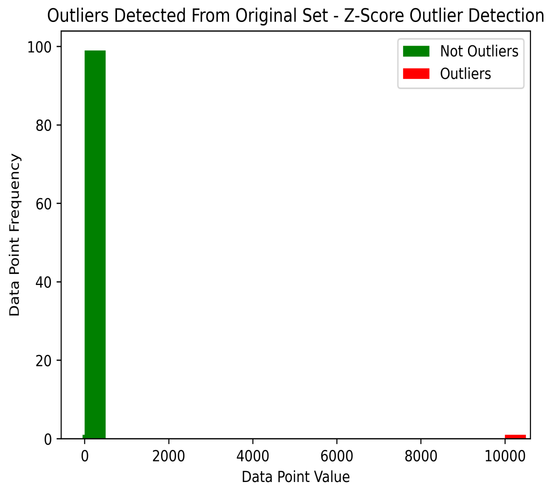


Figure 7: Data with Large Outliers with Outliers Identified Using Z-Score

The Boxplot method [10] is another univariate statistical method that solves this problem by utilizing metrics that are not easily skewed. Instead of the standard deviation, the interquartile range (IQR) is used. The IQR is defined as the difference between the first quartile (Q_1) value and the third quartile (Q_3) value. Instead of using the mean, the average of the first quartile and third quartile of the dataset ($\frac{Q_1+Q_3}{2}$) is used. Similar to the z-score method, the interval which contains all points not marked as outliers can be represented by Equation 4 below.

$$(Q_1 - k * IQR, Q_3 + k * IQR) \tag{4}$$

In this equation, k is a constant that varies depending on the type of outlier being identified. Two such classifications of outliers are *mild outliers* and *extreme outliers*. A data value x is considered an *extreme outlier* if it lies outside of the above interval when k equals 3. A data value is just considered a *mild outlier* if it lies within the above interval when k equals 3 but outside of the interval when k equals 1.5. These constants were chosen by comparison to the normal curve. Similar to z-score outlier detection, the above equation can also be written as the Equations 5 and 6.

$$x < Q_1 - k * IQR \tag{5}$$

$$x > Q_3 + k * IQR \tag{6}$$

These equations represent the comparison to a single data point. If either statement is true, the data point is an outlier. Given the name of the method, it can also be visually represented by a boxplot as shown below in Figure 8.

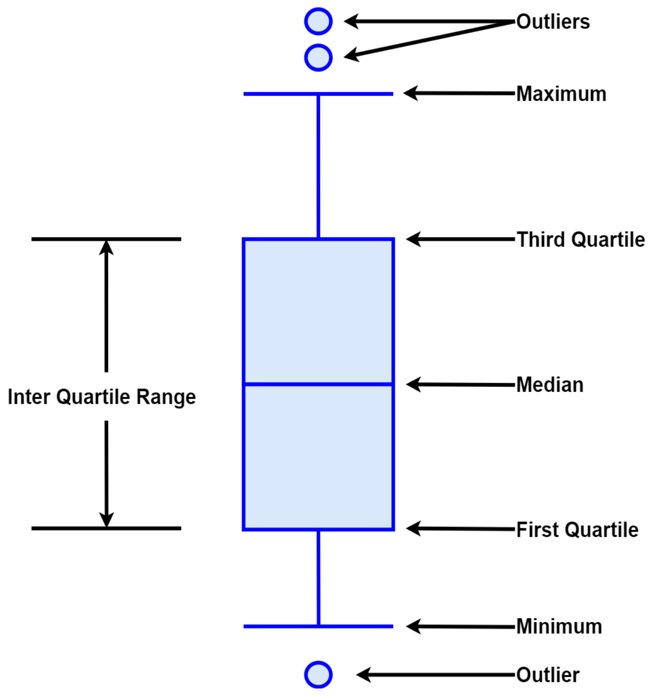


Figure 8: Properties of a Boxplot

To show the effectiveness of this method in comparison to z-score outlier detection method, we identified outliers usin the Boxplot method on the same dataset used above from our GitHub repository. Using the same distribution with a (mean, standard deviation) value of (0.07941666293687392, 0.9670394771137057) and ($\frac{Q_1+Q_3}{2}$, IQR) value of (0.09436841421237988, 1.1561178267491579), the outlier value of 10000 is added to the dataset. While the mean and standard deviation were skewed a noticeable amount as previously stated, the ($\frac{Q_1+Q_3}{2}$, IQR) value becomes (0.09914921583524362, 1.1902531587620824) as shown previously in Table 1. These two values only shift a minimal amount considering the obvious change in the mean and standard deviation. This highlights the robustness of the Boxplot method over the z-score method in identifying outliers.

Though the above methods are effective for univariate distributions, they are not as useful when data values are represented by tuples and have multiple values that simultaneously need to be compared to determine if they are an outlier or not. This is where multivariate outlier detection methods become useful.

The first multivariate outlier detection method discussed is Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [13] [14]. DBSCAN is a form of clustering that separates a dataset into areas of high-density data points and low-density data points. The goal of this algorithm is to identify the sections of high-density points into separate clusters. To be classified as a cluster, this algorithm requires that points have a minimum number of points in that cluster. The process completes its clustering by creating a circle around each data point and classifying each data point based on the number of points within a radius. This is referred to as a point being density-reachable to another point. For example, X is density-reachable from Y when X is in radius of Y. X is density-connected to Y when there exists a point O where both X and Y are density-reachable from O. In this algorithm, data points are either classified as core points, border points, or outliers. Core points have at least a minimum number of points that are density-reachable. A border point is density-reachable from a core point but does not have enough points that are density-reachable to be considered a core point. Lastly, outliers are points that are not density-reachable to any core points. All points that are density-connected then become a separate cluster.

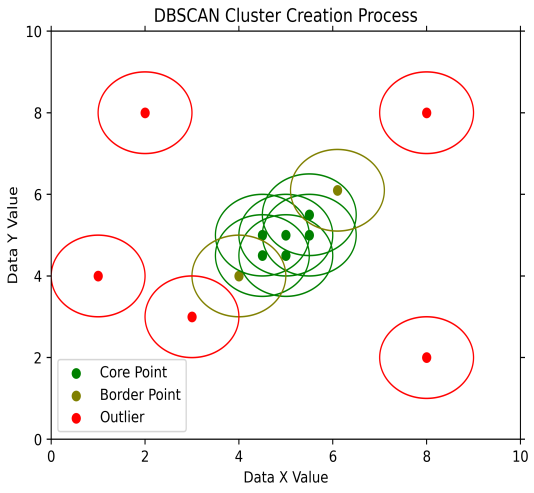


Figure 9: DBSCAN Cluster Selection Process

This process is visualized in Figure 9 above. All green and olive points are included within a cluster while red points are not included in a cluster and are defined as outliers. Once this process is complete, there are several clusters from the density-connected point sets and a number of outliers that did not fit into the requirements of being density-connected with a minimum number of points. Two examples of this clustering can be seen below using two different datasets. Points that are green, olive, or orange are within a cluster, while the red points are defined as outliers.

The data in the dataset shown in Figure 10 is scattered throughout the plot with three densely populated areas. The dataset can be found in our GitHub repository. The clustered results of this process can also be shown below in Figure 10.

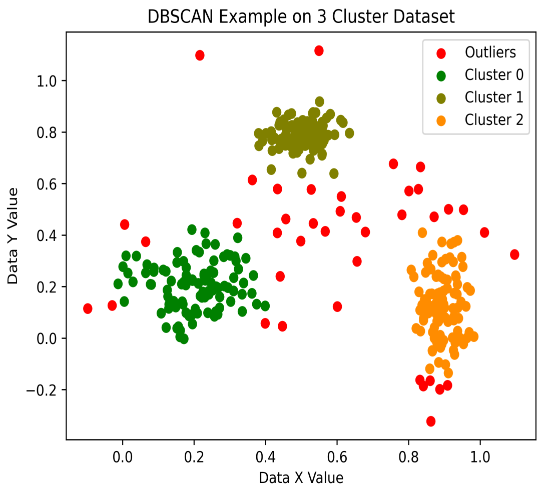


Figure 10: DBSCAN Results with Normal Clusters

The DBSCAN method was successfully able to identify the densely populated areas and identify the noise surrounding them. This example shows normal data with fairly simple shapes. DBSCAN works with any shape of densely populated data, such as the data shown in Figure 11. The data in this figure has one cluster similar to those in the previous dataset, but it also has a densely populated curve. This curve represents sine values with random noise added in. The code to generate this dataset can be found in our GitHub repository. DBSCAN can successfully identify the cluster even with the shape having x values larger and smaller than the spherically shaped data point cloud as shown in Figure 11. Though the above case shows two-dimensional data, DBSCAN can be used with any n-dimensional dataset. Two-dimensional datasets were chosen for ease of visualization with increased complexity.

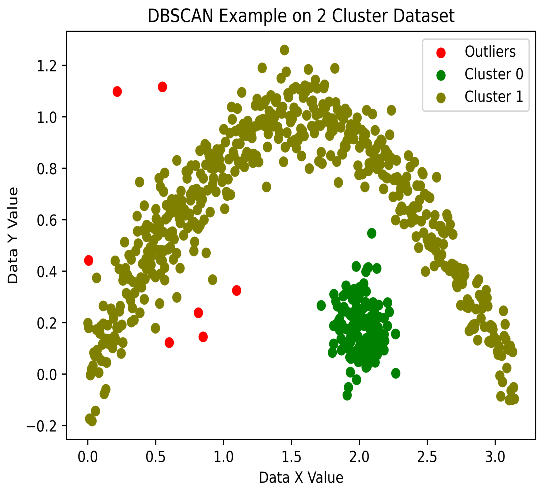


Figure 11: DBSCAN Results with Differently Shaped Clusters

Another multivariate method that can be used is K Nearest Neighbors (k-NN) [13]. The k-NN algorithm uses three constants, m, k, and d, where $m < k$. For each region of a data distribution, it checks if m of the k nearest neighbors are within a distance d. If at least m of the k nearest neighbors are within this distance, the region is classified as being normal. If there are less than m neighbors within distance d of the region, the points inside of the region are classified as outliers. We performed outlier detection with k-NN on another generated dataset as shown in Figure 12 below. The data generation and entire process for this dataset can be found at our GitHub repository.

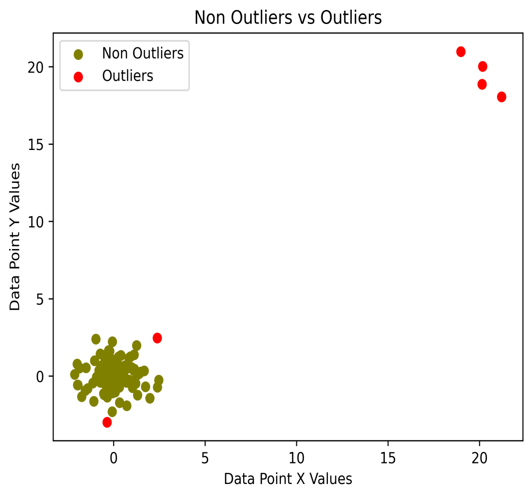


Figure 12: Outliers Identified with k-NN

The k-NN algorithm runs without a distance being known ahead of time, and thus, cannot identify outliers on its own. Instead, k-NN calculates a mean distance metric for every data point that represents the maximum distance that a data point needs to be within to be considered a normal point. For example, if a point has a y-axis distance value of 0.1 and an x-axis distance value of 0.2, then the distance value would need to be 0.15 or greater for that point to not be an outlier. The calculated mean distance metric for a set of data points is shown in Figure 13.

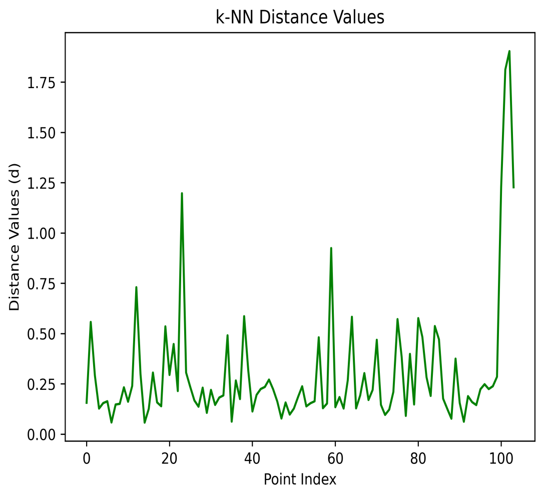


Figure 13: k-NN Distance Values

Once this has been completed, an arbitrary cutoff needs to be assigned to perform outlier identification. The cutoff chosen for this example was 0.8. All points with a distance value above this are now considered outliers, as shown in the plot above in Figure 12. At the end of this process, k-NN has successfully identified several points which would be marked as outliers.

Up until this point, all datasets that have been discussed have been point-wise. Essentially, each data point has no set relationship with other data points. Other forms of datasets, such as time-series, have a clear sequential relationship with each other. To deal with this type of dataset, an additional outlier detection method must be considered.

One method for detecting outliers for data in a time-series format is seasonal decomposition.

Seasonal decomposition is the process of separating a time-series into several components, including the seasonality, trend-cycle, and remainder component [6]. Through this decomposition, the original data values are compared to a new curve created by the combination of the decomposed elements, as done by statsmodels's library. A residual values can then be generated to highlight the difference between the two curves, with higher residuals resulting in points that are more likely to be outliers. We perform seasonal decomposition on a previously used dataset, which can be found in our GitHub repository. Figure 14 shows the decomposition of the dataset in terms of seasonality, trend-cycle, and residuals. Outliers are marked on the residual plot in red.

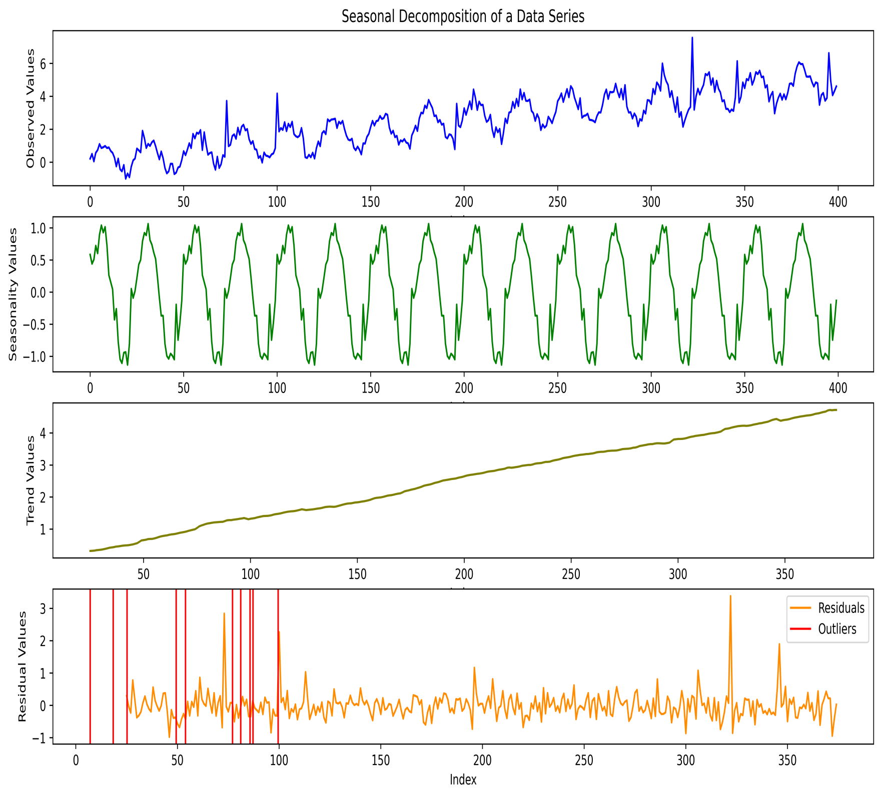


Figure 14: Time Series with Outliers Identified Using Seasonal Decomposition

When the seasonal decomposition results are compared to the components of the generated dataset, the residuals often have the greatest variation out of the three components. This is because any change in both the trend or seasonality is reflected in the residual values, causing variations to be more significant in the residuals. This is problematic as the residual value for each point needs to be robust and not easily skewed since it determines outlier identification. Furthermore, this method is not commonly covered in literature, resulting in minimal standardization of the method. Additionally, there is no clear cut-off for residuals to determine outliers. Due to this and its lack of implementation on multivariate time-series datasets, there is a need for another time series outlier identification method that is not easily skewed.

A time-series outlier detection method that is far more promising, standardized, and popular is the isolation forest method. Isolation forests detect outliers by measuring the average number of steps it takes to isolate data points [15]. This method determines a point to be an outlier if it takes fewer steps than another point because it is deemed more isolated from the other values in the dataset. To further show how this method works, we performed isolation forests on a dataset we generated by artificially adding random values at a rate of 3% to a time-series dataset as shown in out GitHub repository. For each of these features, the isolation forest method determines how likely each data point is to be an outlier based off of every component value at an index. For example, a point with all component values being outliers is much more likely to be classified as an outlier compared to a point with one component that is an outlier. Once this process is complete, the following result is generated in Figure 15.

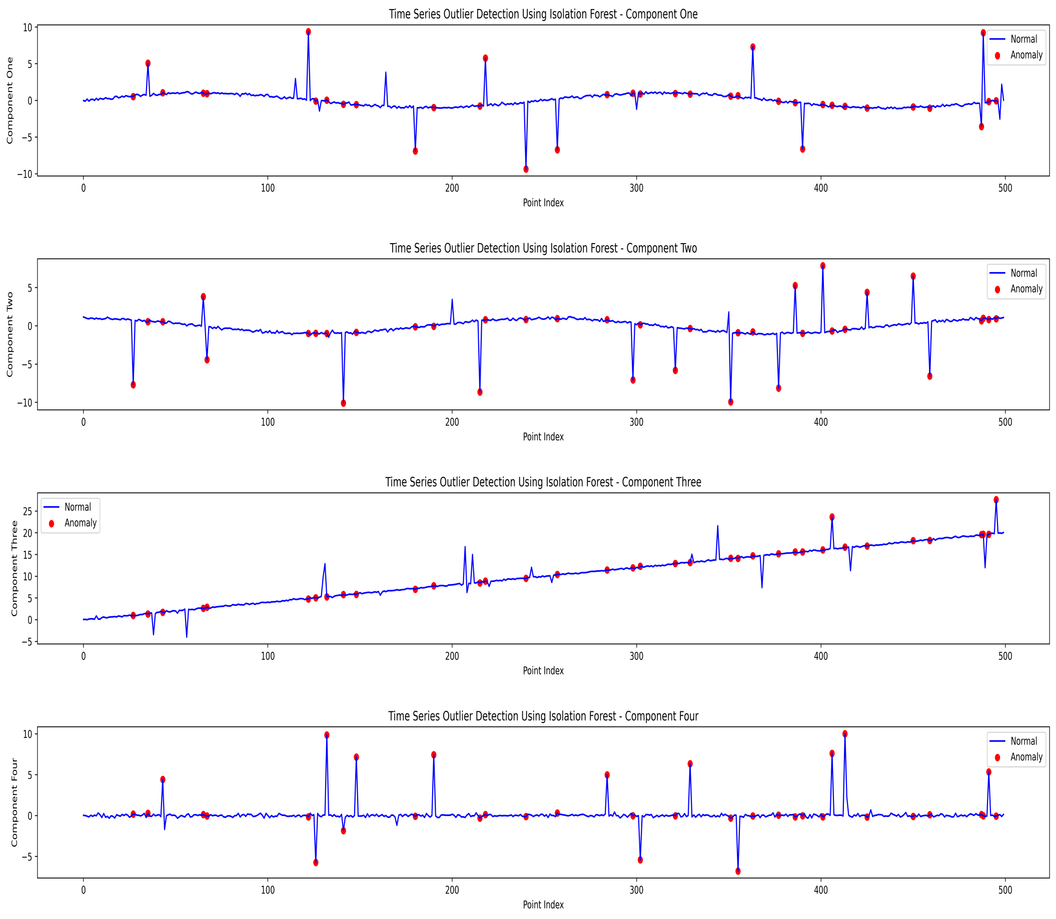


Figure 15: Time Series with Outliers Identified Using Isolation Forests

Once the process is completed, we can see that the majority of outliers injected into the dataset have been determined as outliers. After completing this process, we know which tuple is an outlier but not necessarily which component determined this classification. As shown in Figure 15 above, if a point is an outlier then it is marked as an outlier for every component of the time series data.

2.2.2 Null Correction

A missing value is defined as a value that does not hold meaning within a dataset, such as "NaN" or Null. The presence of missing values in a dataset is a common issue in various domains. In 2004, Peng [16] reviewed 989 randomly selected studies in the fields of applied education and psychology and found that approximately 16% of the datasets contained null values. Clinical trials also suffer from large proportions of missing data. A clinical trial for weight-loss [17] was unable to draw accurate conclusions due to a 59% subject dropout rate. This recurring problem in a variety of fields limits the conclusions that can be drawn using statistical and data-driven approaches. Missing data often leads to misleading and biased results when building predictive models and can have huge implications in crucial studies, such as in the field of medicine [18]. For example, a clinical trial on patients with heart failure [19] reported that the lack of real data in their datasets contributed to the inconsistency in results of similar studies. Furthermore, with the exception of KNN and a few others, most machine learning algorithms cannot handle missing values and produce poor results or return errors. Thus, these values must be handled in the cleaning process in order to salvage the accuracy of predictive models.

Mechanisms of Missingness

In order to properly impute a missing value, the mechanism by which it is missing must be considered [20]. Mechanisms of missingness describe the cause for a value's absence. Whether a value is missing randomly or intentionally can have a large impact on the performance of future predictive models, so it is important to define the mechanism for a given dataset. There are three mechanisms of missingness that are commonly defined in literature and depicted in Figure 16: Missing Completely at Random (MCAR), Missing at Random (MAR), and Missing Not at Random (MNAR).

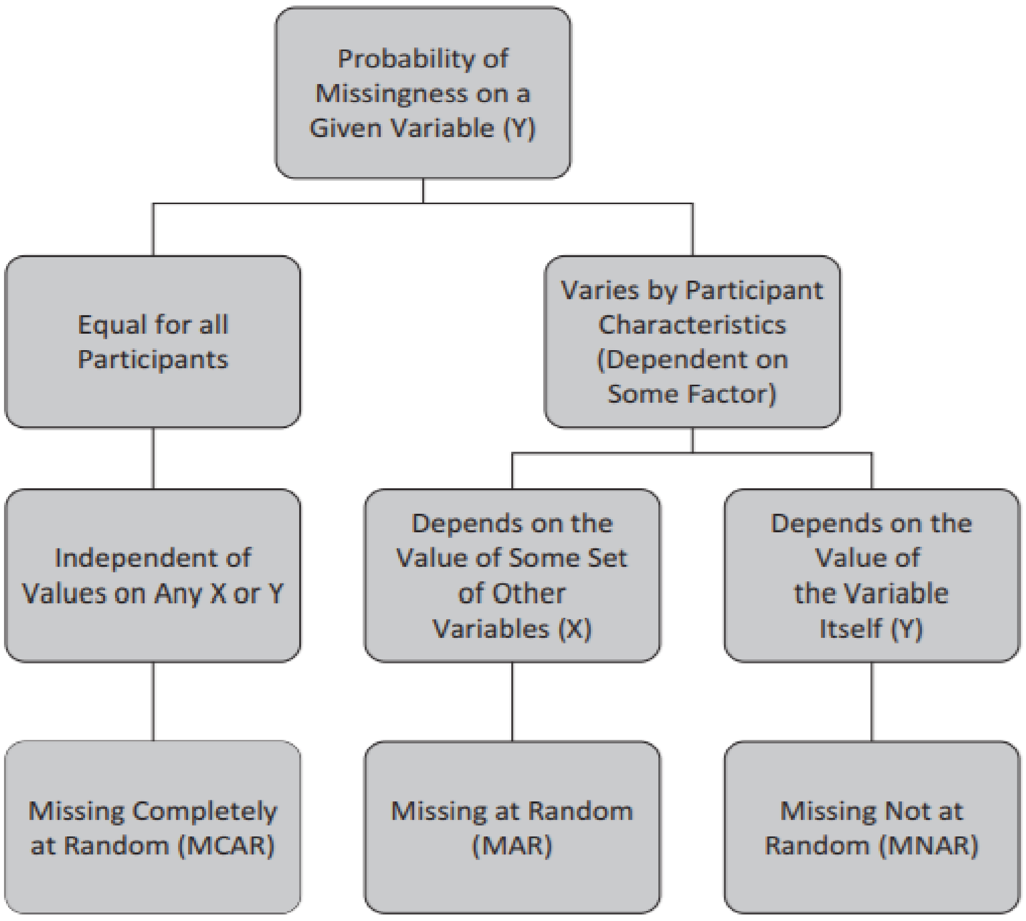

Probability of Missingness on a Given Variable (Y)
Equal for all Participants
Varies by Participant Characteristics (Dependent on Some Factor)
Independent of Values on Any X or Y
Depends on the Value of Some Set of Other Variables (X)
Depends on the Value of the Variable Itself (Y)
Missing Completely at Random (MCAR)
Missing at Random (MAR)
Missing Not at Random (MNAR)

Figure 16: Mechanisms of Missingness Explained

Values that are defined as MCAR if the probability of its missingness is not dependent on any values in the dataset, whether they be observed or missing. In this case, the missing values do not affect the representation of the population distribution in the sample dataset. For example, in real-life medical examples, this can occur when a subject needs to miss a day of the study due to an outside conflict. Since the schedule conflict does not occur due to any factors in the study itself, this missing data is MCAR. The missingness of values that are MAR, on the other hand, relies heavily on other observed values in the dataset. The probability of such values being missing is directly related to values that are present in the dataset but not related to the missing values themselves. For example, in a depression study, participants that identify as male may be less likely to respond to questions or admit the severity of their depression than participants who identify as female [21]. The third mechanism, MNAR, differs greatly from MCAR and MAR. The probability of missingness for MNAR data is highly dependent on both observed and missing values in the dataset. The missingness of these values is not random and occurs mainly due to the method of data collection. For example, in a materials science study dealing with powders, the values of the diameter of particles may be recorded as null if the data collection machine cannot record values below a certain value. As such, these missing measurements are dependent on the missing value itself, and thus, are difficult to predict and recover without collecting the data again. In literature, MCAR and MAR mechanisms are considered ignorable since the bias produced by them can be reduced and the general distribution of the dataset can be recovered. MNAR is considered not ignorable since this mechanism leads to high bias that cannot be reduced through traditional null handling methods.

Handling Missing Data

The best technique for handling missing values must be considered based on the given dataset in order to reduce bias and a loss of valuable information. Unfortunately, various techniques, such as mean imputation and list-wise deletion, that do not properly handle these values for all datasets have been widely adopted. In various fields outside of computing, there is a lack of importance placed on handling missing values [22], which leads to the usage of such techniques due to their simplicity and wide accessibility. As a result, the analysis and development of techniques for handling missing data have been of particular interest in literature [18, 23, 24].

Listwise Deletion (complete-case analysis) is a commonly used technique in which all records containing missing values are removed. At a first glance, this method is advantageous due to the simplicity of its implementation and the lack of bias derived from inaccurate missing value imputations. In particular, in large datasets that contain values categorized as MCAR, listwise deletion may produce unbiased results since the removal of such random missing values preserves the distribution of the sample. However, in smaller datasets, listwise deletion introduces bias and reduces sample sizes, which results in inaccurate and inconclusive results in data-driven approaches [25]. Furthermore, this technique introduces severe bias [26] and offsets the mean and variances in datasets containing values either MAR or MNAR since the removed records are not highly represented in the sample [27]. The bias introduced and valuable information lost by using this technique has caused various sources to deem listwise deletion as one of the worst techniques for handling missing values [27].

The downfalls of deletion techniques paved the way for handling missing values by imputation. Imputation techniques work to replace missing values with an estimate based on observed values in the dataset. Most imputation techniques are categorized as either single or multiple imputation. *Single Imputation* techniques calculate one estimate for a missing value for replacement. Single Imputation is commonly used in non-computing fields due to its simple implementation. However, many of these methods introduce bias and underestimate the standard error of the dataset. One such inadequate method, *mean/median/mode imputation*, has been widely adopted in a variety of fields without consideration of its implications. This method replaces each missing value in a column with the mean, median, or mode of the observed values in that column. This naive approach, although simple, severely skews and biases a distribution and does not attempt to estimate the value that was missing. As a result, mean/median/mode imputation should not be used under any circumstances in order to achieve accurate results.

Another more effective single imputation, *Hot-Deck Imputation* [28], uses observed records in the dataset to estimate the missing value. In this approach, records in the dataset are sorted based on the similarity of their observed values and those of the record with a missing value. Once sorted, the missing value is imputed using the observed value of a randomly selected record from this group [22]. This non-parametric technique is often used in surveys as a way to achieve a more realistic dataset since the estimates are based on real values. Furthermore, the replacement with values in the current dataset preserves the distribution, unlike mean imputation [29]. However, similar to many other single imputation methods, Hot-Deck Imputation tends to underestimate the true standard error of the dataset [22]. Moreover, the nature of this approach forces continuous variables to be categorized, which loses important information [29].

Another similar, more evolved, single imputation method overcomes this limitation while still relying on observed data. *Regression Imputation* is a technique in which missing values are imputed using a regression model based on the observed features. The dataset produced by this technique achieves more reasonable estimates than previously mentioned single imputation methods as it utilizes the natural patterns found in the observed data. However, the correlations between features artificially increase [29] as a result of imputation and may force stronger linear relationships than were previously present. An added error term to the resulting linear regression line can be used to increase the variance and reduce feature correlations to better simulate the real-life data [30]. Unfortunately, this method may cause interpolation of the missing values and increase the bias of standard error estimates [30]. In general, single imputation methods are a simple way to impute missing values, but they often result in biased and inaccurate estimates. Since these methods provide only one estimate for each missing value, they ignore the uncertainty of their imputation and treat the value as if it was never missing. This level of precision needs to be reduced in order to provide a properly imputed dataset.

Multiple Imputation [31] is a popular imputation that can fix these problems. In contrast to previously mentioned imputation techniques, Multiple Imputation aims to preserve the characteristics of the underlying dataset, rather than focusing on a case-by-case analysis. This intuitive approach builds a model from observed values in the dataset to estimate the missing value, similarly to single imputation. This step is performed multiple times, resulting in m "complete" datasets with the missing values imputed. Once m datasets have been produced, statistical analysis is performed on each dataset to derive parameter estimates for the model, and the resulting estimates are pooled to produce a final multiply-imputed dataset. This approach using multiply-imputed datasets preserves the uncertainty that is introduced by missing values by producing unbiased parameter estimates [32]. Multiple Imputation further maintains the natural variability and relationships between features present in the population [31]. Various studies have shown Multiple Imputation to outperform other imputation techniques for datasets with a low sample size, a large proportion of missing data, and a non-normal distribution [33, 34, 35, 36]. However, the computational power required for this technique has prevented a few fields from benefiting from Multiple Imputation [37]. As a result, there have been several software solutions developed to provide access to multiple imputation techniques [38].

Various techniques utilize multiple imputation, one of which is *Multiple Imputation using Chained Equations (MICE)* [39]. MICE, also known as Sequential Regression Imputation and Fully Conditional Imputation, utilizes regression models to build its multiply-imputed datasets. During each cycle of imputed-dataset creation, a feature, x_j, containing missing values is modeled with regression trained on all available features in the dataset and only records with observed values for x_j. The regression model chosen depends on the data type of the feature being imputed. Each missing value in x_j is then imputed using the generated regression model. In the following iterations, another feature containing missing variables is modeled using all features in the dataset where any previously imputed feature is used in place of its original null version. Once all the missing values have been imputed, the cycle is complete, and a new cycle begins to re-impute each of these missing values. Each subsequent time these values are imputed, the features used include the updated imputation of the given column [39]. MICE ends this iterative process once the regression coefficients converge [40]. In general, MICE performs optimally when all the feature variables of the given dataset are provided, including the target variable. If one of these variables is not included, missing value estimates are biased and the correlation between an imputed feature and a held-out feature are not preserved and are often reduced to zero [40]. This approach to multiple imputation is incredibly accurate in comparison to various other imputation methods [41]. It is also flexible due to its ability to handle various variable types, such as continuous numerical, binary, unordered categorical, and ordered categorical, using a variety of regression models, such as linear regression, ordinal regression, and logistic regression [41]. Furthermore, MICE is not time-intensive since it performs well on a small number of iterations.

2.3 Related Technology

Data cleaning, although important, is often a time-intensive process. Thus, recent efforts have focused on developing software solutions to provide easy access to data cleaning tools.

2.3.1 Sci-Kit Learn Cleaning Modules

One such solution is provided through Sci-Kit Learn's extensive data science library [14]. Sci-Kit Learn is a vast library that contains many modules widely used in data science. One of these modules is focused on clustering. Our implementation for DBSCAN outlier identification uses this library. This module offers an assortment of clustering methods, ranging from K-Means, DBSCAN, OPTICS, and more. It also allows for extensive customization, allowing many parameters to be specified in each clustering configuration. A similar module in Sci-Kit Learn's library is the imputation library. It also focuses on allowing as much customization as possible. However, it includes a minimal amount of automation.

Though these packages offer a massive collection of useful methods, it suffers from the ability to specify many options for each configuration. The library itself does not have any tactics to recommend parameters outside of default parameters, which is problematic for non-technical users. Our application offers the automation that this library does not. The automation of our application saves the user time in learning what each of the parameters are used for while also ensuring that they are using the right values for their right dataset.

2.3.2 ActiveClean Framework

Moreover, recent frameworks have been developed to automate the cleaning process. ActiveClean [42] is an automated framework in which a dataset is cleaned based on its current accuracy in a predictive model. Studies have shown that data cleaning can be conducted as an iterative process using Active Learning in which a model is trained and dirty data can be identified based on the accuracy of the model. ActiveClean utilizes this approach by iteratively updating the training model based on the deemed clean data. Once the model converges, the framework returns the dirty data in the provided dataset. This approach is a strong contender for automated data cleaning as it produces accurate predictive models. This framework also provides a user-friendly interface in which users can explore their dirty data and decide how to proceed [43]. However, this approach is parametric and can only be used if the user provides the model for predicting, and a method for cleaning detection. Furthermore, the only way to clean dirty data with this user interface is to either delete the dirty data, write a custom method for cleaning, or choose from a pre-defined list of cleaning techniques. Based on our previous research, data should be treated in a certain way depending on its properties, such as the type of the dataset, and should be reflected in this automation to ensure accurate results.

2.3.3 BoostClean Framework

Another such automated tool is BoostClean [44]. This framework is designed to find the best cleaning process for a given dataset using Boosting. Dirty data is first identified using isolation forest-based outlier detection, and then an ensemble of cleaning techniques is iterated through using Boosting to determine the most accurate ensemble. Unlike ActiveClean, this framework utilizes machine learning to truly automate the cleaning process and provide the user with a clean dataset. Nevertheless, the repair algorithms used by BoostClean are limited to mean/mode/median imputation and listwise deletion. As stated previously, these imputation techniques are inaccurate and weak repair techniques as they introduce bias and destroy the distribution of the population. Along with this, their technique using Boosting brute-forces the cleaning process, rather than using the properties of the dataset to accurately clean the data. Moreover, BoostClean is not open source and does not provide a user interface to allow for exploratory data cleaning and outlier detection. This blind cleaning is not optimal since each domain defined its outliers differently.

Thus, there is a necessity for an automated cleaning suite that provides the user with tools for exploratory dirty data analysis as well as recommendations for accurate, robust data cleaning based on the properties of their dataset and dirty data.

3 Methodology

To ensure the successful implementation of the CODeRS application, we clearly define a set of practices in this section that we employed for the entire development cycle. We describe the plan for front-end and back-end development, focusing on their architecture as well as the libraries used. Additionally, we discuss the various methods and dataset properties we used to build our recommender system for dataset cleaning. We also elaborate on our techniques for evaluating both the front and back end portion of our application.

3.1 Technical Approach

GitHub Workflow

Before beginning development, we established a clear set of GitHub practices to organize our work and focus development towards a series of set goals. Each developer followed the following process during the development of each feature:

- When a developer wishes to implement a new feature or update an existing one, they should first make a GitHub issue.
- Once the GitHub issue has been created, the developer should make a new branch to develop and implement their features.
- Branches should only contain development for one focused issue. This ensures that all merges to the main branch can be easily reverted in case of changes or errors.
- While committing to their branch, developers should include the relevant GitHub issue numbers where appropriate in commit messages.
- Commits to a branch should occur early and often.
- Once the development of a feature has been completed, the developer should submit a pull request with a description of the work that was done. The developer should assign another member to review the pull request and ensure everything added to was done properly.
- Once the reviewer has approved the pull request, the developer who initiated the pull request should merge with the main branch, delete the branch they were working on, and resolve the issue.

Test Driven Development

In addition to the above mentioned practices, we utilized Test Driven Development (TDD) throughout development in both the front-end and back-end of our application. Test Driven Development is implemented by writing tests before developing functionality. Whenever new functionality was created on the back-end, datasets and examples were first created to test the effectiveness of the developed feature. For example, when developing the z-score outlier detection, we created a normal dataset with several outliers to ensure that the outlier detection method detects the outliers properly. We chose to implement this practice to ensure the longevity and robustness of our application as well as our focus on implementing features one at a time. This reduced the time searching for bugs and fixing code. Additionally, this provided a set of additional tests to automatically run using Pytest and GitHub Actions during a merge to our main branch to ensure that no new feature negatively affected old features. If any tests fail, the developers were notified to correct anything that may have been broken unintentionally.

Agile Development Method

Our MQP team employed the Agile framework by developing a consistent schedule and set of tasks each week. The Agile framework is implemented by several top software companies to keep developers motivated and focused on specific goals. Each week, we held a scheduled meeting and defined a set of development tasks to complete. For each meeting, we produced the following set of deliverables:

- A meeting agenda including all meeting participants and an outline of everything the group plans to cover. All agendas are available here.
- A presentation covering the progress made during the week using the appropriate terminology and visuals. All presentations are available here.
- A minutes document to document all discussion that takes place during the meetings. All minutes documents are available here.
- A Gantt chart to reflect the work that was completed during the past week. This allowed the team to compare the actual work that was completed against what was planned. All Gantt charts created after the week has been completed are available here.
- A Gantt chart to reflect the work planned for the upcoming week. All Gantt charts planning the next week are available here.

For each of these meetings, one team member led the meeting, while the other took minutes. During the following week, team members completed the tasks they were assigned to the best of their abilities.

3.2 User Interface

As previously mentioned, the goal of this project is to generate an application for cleaning data that is easy to use as a non-technical user. Thus, we built a web application with a Graphical User Interface (GUI) for users to process their data quicker and easier. We chose to utilize a web application due to its easy accessibility for a wide range of users.

During the development of the user interface, we followed a series of steps. We started the development process by generating a mock-up the entire web application, focusing on both the user interface and user experience of the application. For the user interface, we focused on the appearance of the application, including the color scheme and designs of the various web elements. For the user experience, we focused on the user's interaction with the application, including the placement of web elements, the page layout, and the sequence of web pages. We discussed these mock-ups during our meetings with our advisor and refined our designs accordingly. In cases with severe design flaws, we scrapped our old mock-ups and designed a new one to be evaluated. Once we finalized our mock-ups, we developed the front-end of our application accordingly. From here, we utilized a series of user studies to gather further feedback regarding the user interface and experience of our application as described in subsequent sections. Based on the feedback received, several rounds of adjustments were performed on our application before another user study. This process helped us to make our application as user-friendly as possible.

In addition to the user-friendly interface, we ensured usability by developing a user manual to help users understand further how to use the application. This provided insights into the libraries needed, installation instructions, and descriptions of the functionality offered. Not only did this further simplify the user experience, but it also furthers the longevity of the project.

3.3 System Architecture

In addition to the above mentioned methods, we implemented a set of best practices to develop the back-end of CODeRS. The first practice is to develop each step in our cleaning pipeline in a modular fashion. Following this practice allows developers to replace, add to, or subtract from any portion of the data cleaning process with minimal time and effort without affecting other portions of the pipeline. This increased efficiency during each development cycle as well as followed the same organization our user encounters when using CODeRS. Our intent is for the end-user to be able to run any set of data cleaning operations on a dataset, which requires modularity. This also improved the longevity of the project. Modularity allowed each method to be reasonably separated and new methods to be created following the same template. One example is described with outlier detection. If the z-score method is updated to allow for extra functionality, it should not interfere with the functionality of other methods. At the same time, the addition or removal of an existing method should not interfere with the operation of the system when using other outlier detection methods.

Another practice we implemented is the use of abstraction. This involves having the project use the same process to employ different cleaning techniques. For example, if abstraction is applied to the outlier detection portion of the project, the developer only calls one set method to perform outlier detection. They do not have access to the other methods used internally in the project and only need to call a general identify_outliers() method. The greatest benefit of this is that it keeps the structure of the program the same when completing similar tasks. If a developer wants to detect outliers using z-score and the Boxplot methods, they only have to change a minimal amount of code.

The last design method we incorporated in our project is a clear and strict folder structure. We want to group all related files in the necessary folders and sub-folders to organize the project so both a developer and user could understand its organization. This involves putting all callable methods in one folder and possibly an outlier detection sub-folder in this main folder. This is especially important for the longevity of the project as it results in a fraction of the time being needed to onboard a developer to the project structure.

3.4 Libraries Utilized

Throughout the development of our application, we heavily relied on existing python libraries. We created the entire pipeline of our application but utilized these libraries to implement the various modular processes on the back-end. An example of this could be using a library to automate the documentation process or using a series of libraries for different outlier detection methods. In addition, we utilized libraries that implement the specific cleaning techniques themselves. Since the cleaning techniques implemented are widely defined, using python libraries ensures that the implementation of the methods in our application is correct. Below is a list the libraries we used as well as the motivation for using them.

- pandas - The pandas library was used for storing and transforming data. One common pandas data structure that is commonly used in the project is the pandas DataFrame. This data structure stores index columns of data with extra information about each column and the object itself. It is commonly used to store the entire dataset we are reading in.
- SciPy - The SciPy library was used for several statistics elements in the project, such as finding the Interquartile Range of a list of points for the Boxplot outlier detection method.
- NumPy - The NumPy library was used for example dataset generation and other data list operations, such as finding the percentile cutoff value of a list of data points. NumPy is especially useful as only a few parameters are required to generate an entire dataset. For example, the dataset used in Figure 11 was created with only 4 different numpy datasets.
- sklearn - The sklearn library was used for the implementation of several algorithms, such as DBSCAN, MICE, and k-NN. Using these libraries allows development to go faster and ensures the algorithm itself is functional. With complex methods such as DBSCAN, this is especially useful.
- pytest - The pytest library was used to run automated tests on the existing code base. All testing methods are structured in a fashion where pytest automatically runs the methods to see if all pass or fail.
- Matplotlib - The Matplotlib library was used to produce all visualizations made by the project. Matplotlib was chosen due to its highly customizable plotting area, easy-to-use function calls, immense popularity, and compatibility with other libraries.

- Sphinx - The Sphinx library was used to generate all auto-documentation created for this project. The theme for the current auto-documentation sections is from the sphinx_rtd_theme library.
- Missingno - Missingno is a library that was used to create various visualizations to represent the presence of null values in a dataset. This library is used to generate the images that are viewed on the Null Identification page of our application.

3.5 Automated Recommendation System

Our major contribution to this cleaning suite is a recommendation system that automated the cleaning process using state-of-the-art techniques found in our literature research. Currently, popular cleaning techniques utilized are single imputation that provides more harm to data analysis than benefits. Furthermore, past attempts to prevent the overuse of these techniques and automate the cleaning of data performed a brute force analysis to determine the proper cleaning techniques. In contrast, our cleaning suite utilizes the properties of the dataset to derive the proper cleaning and detection techniques. For null correction, the mechanism of the missing values determines the appropriate correction method. However, Multiple Imputation by Chained Equations has been proven to perform extremely well on ignorable missing data (i.e. MCAR and MAR). Thus, our suite recommends the usage of MICE for imputation for all datasets. For values that are MNAR, we cannot provide recommendations for imputation. Since MNAR values are dependent on values that are not present in the dataset, they cannot be properly replaced and should be recollected in the data-gathering phase altogether. Furthermore, there is no way to determine whether missing values are MAR or MNAR since the difference is based on values that are not observed. Thus, we suggest that any dataset provided to this cleaning suite should not contain MNAR data as we cannot remove them for cleaning. For outlier detection, the type of dataset plays a role in the proper recommendation for cleaning. In our literature research, we found that DBSCAN outperforms all other outlier detection methods for point-wise data. However, for time series data, we found that isolation forests work better than DBSCAN. Thus, our application identifies outliers in using DBSCAN for point-wise datasets and isolation forests for time series datasets. In addition to this automated system, our application allows the user to ignore or remove dirty data rather than correct it. Although our techniques are state-of-the-art, the domain from which a dataset is generated must be considered before performing any cleaning. For example, in the Materials Science field, a domain expert may choose to ignore the outliers because they know that some powders in their datasets are much larger than the other available powders. Even though these powders appear to be outliers, they are correct and are important to the data analysis. As a result, we provide the functionality to allow domain experts to choose how to handle their null values and outlier based on what is best for their dataset.

3.6 Evaluation Techniques

In order to properly evaluate the performance of our cleaning techniques in the back-end of our suite, we utilized our cleaning suite on artificially dirty datasets. In the case of outlier detection, we injected outliers into several datasets with differing distributions. We used our outlier detection techniques to identify the outliers and utilize the original data to measure how accurate our detection was. For missing value imputation, we used several datasets and generated multiple versions of these datasets, each with a differing percentage of missing values and a different missing data mechanism (i.e. MCAR, MAR, and MNAR). Since MNAR values cannot be accurately imputed, we do not generate such datasets. Such evaluation techniques are commonly used to compare cleaning techniques in literature from various fields [45, 46, 47, 48, 41, 30]

The user-friendliness and usability of the GUI constructed was evaluated through user studies. In these user studies, we provided each participant with a list of tasks pertaining to the use of our platform in order to evaluate the user experience. We also provided each participant with a form to evaluate the user interface with the Nielsen Heuristic. This heuristic provided ten metrics to determine user-friendliness and usability. To measure users' analysis of the MQP team's GUI according to this heuristic, the team provided them with a numerical scale to rate the application as having achieved according to each metric. For example, the first metric as listed at Nielsen Norman Group's website is

"#1: Visibility of system status. The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time. To measure this and other metrics, they had a scale underneath each section between 1 and 4 to rate how well the current design follows this metric. A comment section also was provided to allow for specific reasons for high or low ratings. We conducted these user studies with various materials science members of the Data-Driven Materials Science research team at Worcester Polytechnic Institute as this cleaning suite directly benefits their work. We also chose this group since it allowed us to evaluate the user-friendliness of this suite on researchers in a non-computing domain.

4 Implementation

Utilizing the knowledge we gained through our background research as well as the methodology we outlined in the previous section, we implemented our web application, CODeRS, to help non-technical users clean their data effectively. CODeRS can be accessed by the link located in Appendix 7.1 In the following section, we discuss how we structured our web application as well as the server we chose to host CODeRS on. Additionally, we provide a demo of the application through a series of images from start to end of the cleaning process.

4.1 System Architecture

4.1.1 Python

Python is an ideal language for data processing, outlier detection, and data manipulation due to the large number of libraries available for these purposes. Due to this, any action that manipulates or processes the data in some way is handled in Python using the libraries discussed in the background section. In the back-end, helper methods were created to run the cleaning methods described in the background, such as null identification, outlier identification, and outlier correction. To fit our Python back-end into the structure of our Ruby on Rails web application, we made a series of Python pipelines to accomplish the necessary tasks. Described below are the Python pipelines that were implemented along with a brief description of their functionality.

- **Null Identification Executable** - This executable reads in the initial data file and identify all the null values. This is completed through reading a data file in the file structure and producing the necessary visuals to highlight these nulls. This step is designed to provide the user with valuable information on the nulls within their dataset.
- **Null Handler Executable** - After nulls have been identified, this executable handles the nulls and produces an updated CSV file. The options for handling nulls are below.
  - **Ignore Nulls** - This option leaves all null values in the data file. In this case, a new CSV file is not created. This step does not allow the web application to move onto outlier identification and handling since DBSCAN cannot run properly if null values are present in a dataset.
  - **Remove Nulls** - This option removes all null values from the dataset. If any tuple has a null value in one of its columns, the entire tuple is removed. The resulting data file is used for the following outlier identification step.
  - **Correct Nulls** - This option corrects all null values using the recommended MICE null correction technique. The new dataset with the corrected values is placed in a new datra file to be used for the following outlier identification step.
- **Outlier Identification Executable** - This executable reads in the data file generated after the Null Handler Executable is run to identify outliers. This process has several different outlier detection methods that can be run as described below. Once this has been done, the non-outlier and outliers are separated into two different Pickle files, and visuals to show the outliers and non-outliers in the dataset are produced.
  - **DBSCAN Outlier Identification** - If the data is not a time series dataset, DBSCAN Outlier Identification is used.
  - **Isolation Forest Outlier Identification** - If the data is a time series set, Isolation Forest Outlier Identification is used.
- **Outlier Handler Executable** - After outliers have been identified, the outliers are handled, and a CSV file with the updated dataset is produced. The options to handle outliers are below. The CSV file generated by this executable represents the final cleaned dataset.

  - **Ignore Outliers -** This option leaves all outlier values in the data file. In this case, a new CSV file is not created.
  - **Remove Outliers -** This option removes all outlier values from the dataset. The resulting data is moved into a new CSV file.

4.1.2 Ruby on Rails

CODeRS was created as a web application to allow availability and accessibility to users. As such, we chose to utilize the Ruby on Rails framework to develop our application simply and effectively. Rails is a full-stack, server-side framework that provides back-end and front-end template structures to build web applications. This framework provides all the necessary tools for building a web application, which allows the developer to focus on their specific application implementation. Several popular websites utilize Ruby on Rails, including GitHub, AirBNB, SoundCloud, and Hulu. In our architecture, the web application itself runs on Ruby on Rails. As such, Ruby controls the HTTP actions, underlying logic, and page directions, PostgreSQL and Ruby combined handle the database, and HTML and CSS generate the front-end and views.

Ruby on Rails is most suitable for web applications but struggles to handle machine learning applications. Thus, the logic utilized to derive the outlier identification and null correction tasks is handled by a Python script that is called from within the Ruby on Rails framework. As a result of this interaction between Python and Ruby on Rails, there was a necessity to develop a system for handling the data files and information produced by the scripts. Each time a user opens the CODeRS application, a new unique ID is created for their session and stored in the database along with information about their dataset. Furthermore, a new folder named with their session ID is created within the storage folder in the Ruby on Rails application. When the outlier identification task or null correction task is executed, the files produced, including any visualizations or altered CSV files, are stored in this folder. Once the user starts the process over, their folder and database record is removed. In the case where users exit out of their browser mid-way through the workflow, their data is removed within 1 hour of creation, keeping our database and file system clean.

4.1.3 Model View Controller

Since we utilized Ruby on Rails to build our application, we chose to follow the Model-View-Controller (MVC) framework. The MVC framework is a popular software architecture in which the functionalities of an application are broken down into three components.

1. **Model**: This component refers to any data that is used throughout the application. Such model classes contain data-related logic regarding a table in the database. Items in the model can be transferred between Controller and View classes. In our application, Sessions for each user are stored in the database and provided a model class to keep track of any logic regarding retrieving, storing, or verifying Session data.
2. **View**: This component contains logic pertaining to the front-end or user interface of the application. All aspects of the application that are shown on the screen when using the application are handled by this component. In our application, our view is full of HTML files.
3. **Controller**: This component handles all interactions between the Model and View components. Logic pertaining to page redirects and HTTP calls are processed by the Controller components. In our application, when the ”Identify Nulls” button is pressed on the first page, the Session Controller Create action creates a new Session with the user's data and redirects to the Identify action within the NullTasks Controller. From here, the Controller calls the Python script to identify nulls within the user's dataset. The user is then redirected to the Show view where the user can view visuals regarding the presence of nulls in their dataset.

The architecture provided by this framework promotes simplicity throughout the entire application. The separation of such components manages complex applications by encouraging the usage of single-responsibility classes. This, in turn, simplifies the test-driven development process and reduces redundant code by allowing each component of the application to be tested once.

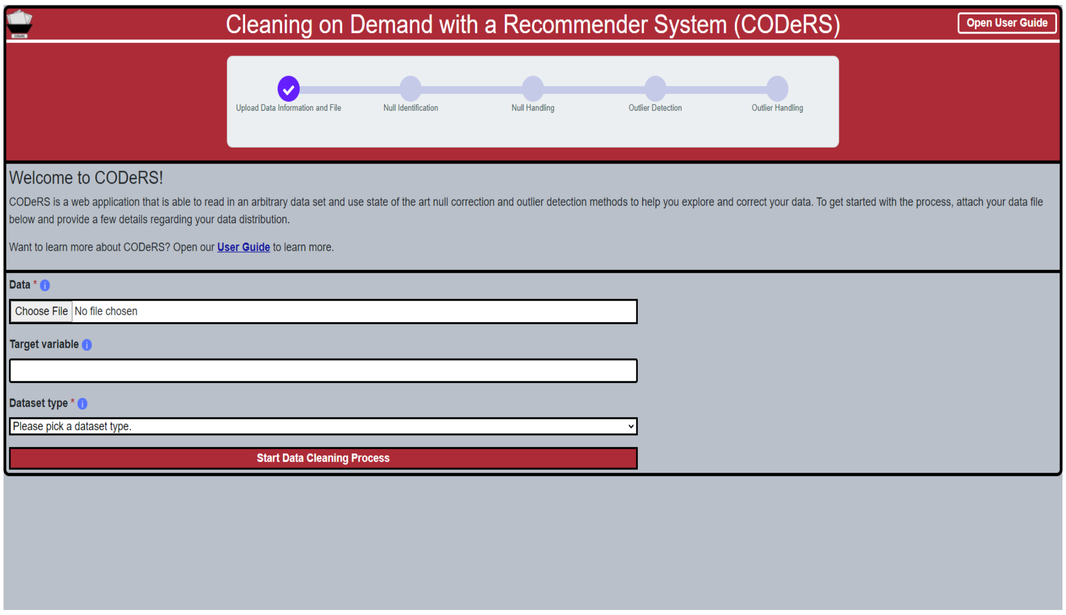
4.1.4 Hosting Software

To host our web application, we used Heroku services. Heroku hosts applications connected to a GitHub branch, allowing for new code development to be quickly and easily updated in an existing web application. Heroku provides quicker page retrieval times at a free level as compared to other web application hosting services such as Glitch. We were able to successfully conduct all of our user studies using Heroku and plan to leave the application on Heroku for our final build. We used Heroku over a custom server due to the simplicity and flexibility it offers. Instead of worrying about a payment plan or setting up the environments necessary to host our web application, Heroku can complete this for free and with the development tools already installed. Ruby was especially difficult to install on our Windows machines during the development of the web application. Thus, utilizing Heroku saved us valuable time that we then allocated to spend time adding functionality to and polishing our final web application. This likely will be an area for future MQP teams to explore when taking over this project.

4.2 User Interface

The user interface and experience of our application is best described by a walk-through of the entire website. Our full application follows a linear process in which users clean their null values before identifying outliers. We chose to use this linear process due to the requirements of our chosen outlier detection technique. Null values cannot be present in a dataset while utilizing DBSCAN or Isolation Forests to detect outliers. Thus, it was important for our application to force users to handle their null values before moving onto outliers. Moreover, the users of our application are not experts in Data Science. Due to this lack of expertise, the linear structure better guides our users through the process by giving them a set of appropriate steps to take to clean their data.

Our application starts on the home page as shown in Figure 17 below.


Cleaning on Demand with a Recommender System (CODeRS)
Open User Guide
Upload Data Information and File
Null Identification
Null Handling
Outlier Detection
Outlier Handling

Welcome to CODeRS!

CODeRS is a web application that is able to read in an arbitrary data set and use state of the art null correction and outlier detection methods to help you explore and correct your data. To get started with the process, attach your data file below and provide a few details regarding your data distribution.

Want to learn more about CODeRS? Open our User Guide to learn more.

Data *

Choose File No file chosen

Target variable

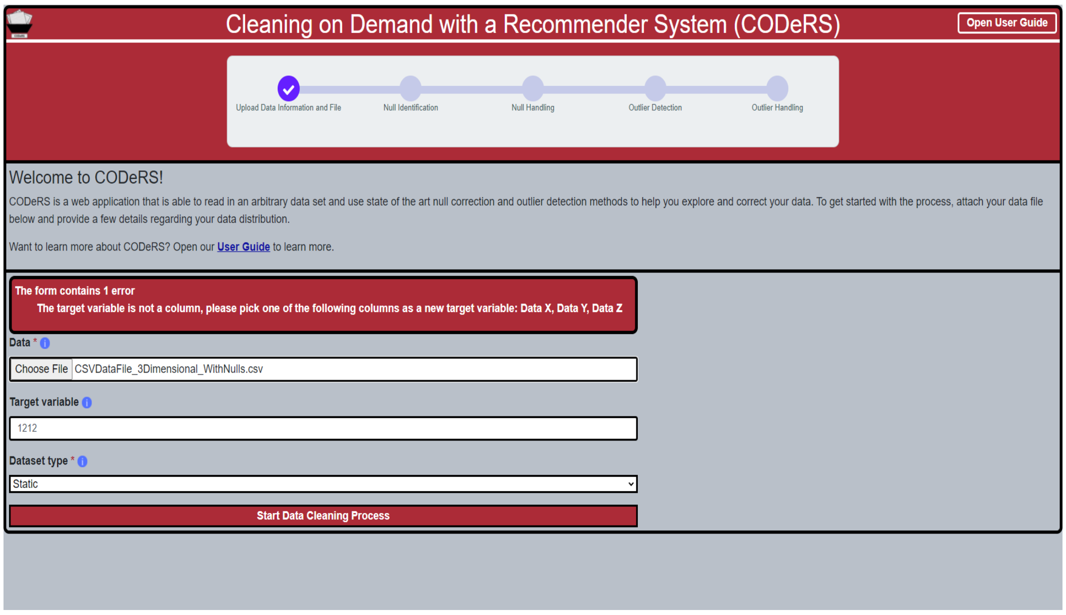
Dataset type *

Please pick a dataset type.

Start Data Cleaning Process

Figure 17: Web Application Home Page

The home page serves as the starting point of the cleaning process where users upload their dataset and answer a few questions pertaining to the characteristics of their data. The dataset upload and dataset type (i.e. Time Series or Static) are required fields because cleaning techniques used in the subsequent pages utilize the responses in these fields. The target variable is an optional field as some users do not know the target variable of their dataset before the cleaning process. We ask for the target variable because outlier detection should not be performed on this column due to the nature of our outlier detection technique. To ensure that the user enters a valid column name, our application checks the column names in the given dataset and returns an error as shown in Figure 18 if the entered column is not in the dataset. To assist the user in choosing a valid column, the error lists the possible columns in the given dataset. Once a user has inputted the appropriate information, they can press the "Submit" button at the bottom of the page to start the cleaning process.


Cleaning on Demand with a Recommender System (CODeRS)
Open User Guide
Upload Data Information and File
Null Identification
Null Handling
Outlier Detection
Outlier Handling

Welcome to CODeRS!

CODeRS is a web application that is able to read in an arbitrary data set and use state of the art null correction and outlier detection methods to help you explore and correct your data. To get started with the process, attach your data file below and provide a few details regarding your data distribution.

Want to learn more about CODeRS? Open our User Guide to learn more.

The form contains 1 error
The target variable is not a column, please pick one of the following columns as a new target variable: Data X, Data Y, Data Z

Data *
Choose File CSVDataFile_3Dimensional_WithNulls.csv

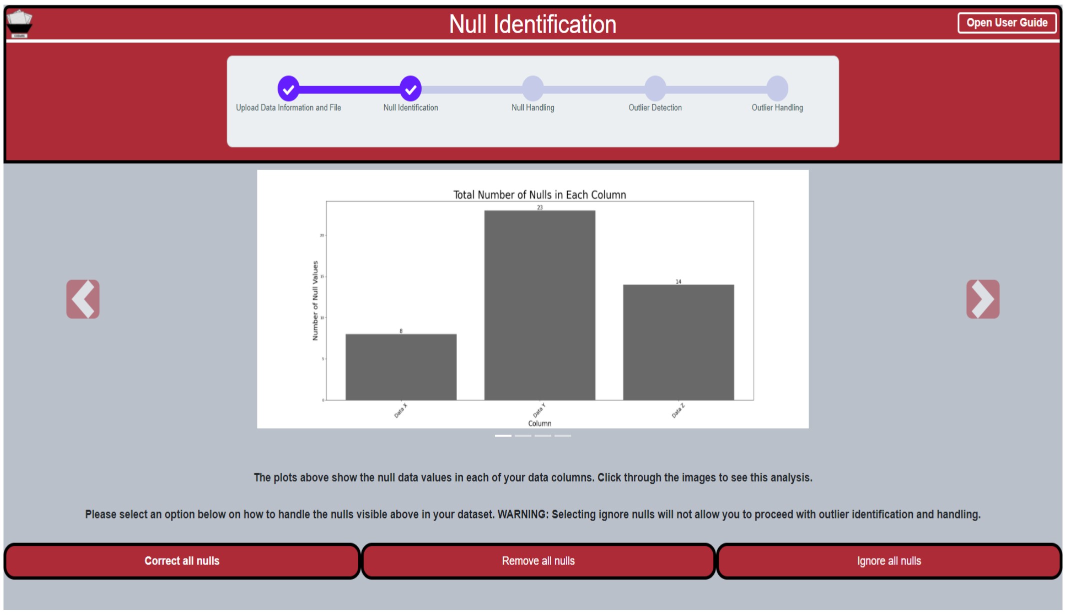
Target variable
1212

Dataset type *
Static

Start Data Cleaning Process

Figure 18: Web Application Home Page - Wrong Column Name

The next page starts the cleaning process by identifying the null values in the dataset. As shown in Figure 19 below, our application represents the null values in the user's dataset through a series of four visuals. Users can scan through the images by clicking on the left and right arrows of the image carousel. An indicator is shown beneath the image view that shows which image in the series the user is viewing.



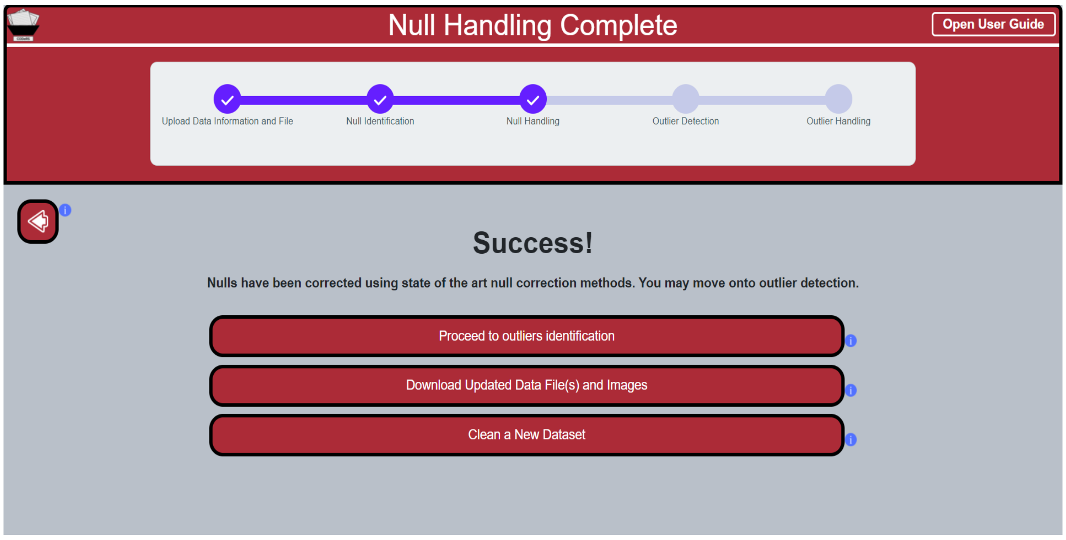
The plots above show the null data values in each of your data columns. Click through the images to see this analysis.

Please select an option below on how to handle the nulls visible above in your dataset. WARNING: Selecting ignore nulls will not allow you to proceed with outlier identification and handling.

Figure 19: Web Application Nulls Identified Page

From here, the user can select to remove, ignore, or correct nulls in the dataset. Clicking on the ”Remove all nulls” button creates a new dataset where tuples containing null values are removed. Furthermore, clicking on the ”Correct all nulls” button creates a new dataset where previously null values are corrected using the MICE algorithm described previously. In contrast, clicking on the ”Ignore all nulls” button results in no change to the user’s dataset. As shown in the warning above the buttons, selecting the ”Ignore all nulls” button ends the cleaning process for the user. Our recommended outlier detection algorithm cannot detect outliers in a dataset containing null values. Thus, ignoring null values forces the cleaning process to end. A user can proceed to the next page by clicking any of these buttons.

During the transition from identifying null values to completing the cleaning step for handling null values, the null values are handled based on the button pressed by the user. As shown in Figure 20 below, the following page confirms the completion of the null handling step.

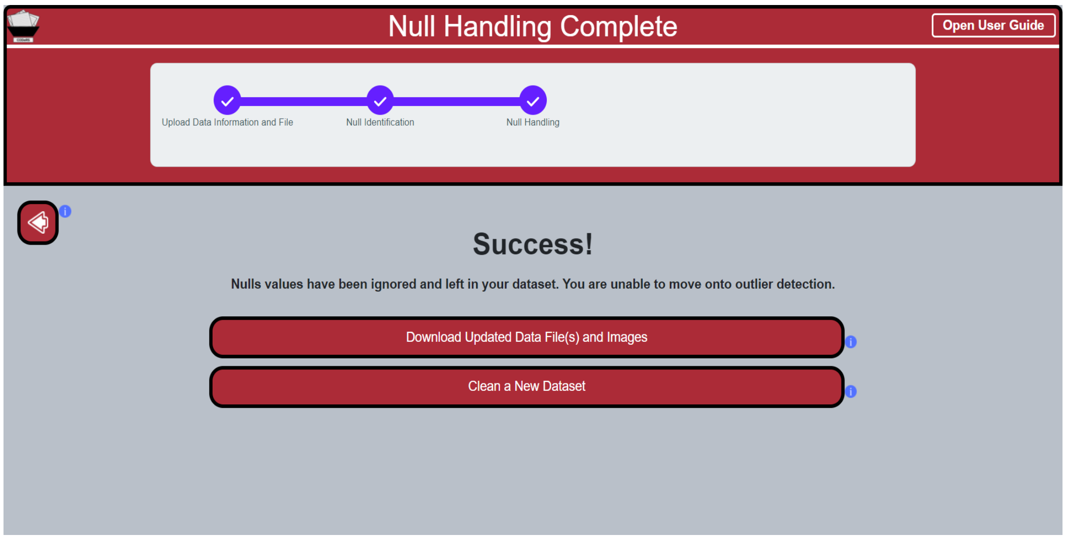


Success!

Nulls have been corrected using state of the art null correction methods. You may move onto outlier detection.

Figure 20: Web Application Nulls Handled Page

From here, the user is given several actions. The ”Download Updated Data File(s) and Images” button allows the user to download their original data file and the new data file created as a result of their chosen null handling technique. This button also allows users to download all of the images shown in the previous step. Clicking the ”Clean a New Dataset” button ends the cleaning process for this dataset and allows the user to start the entire process over. The back arrow provided in the top left corner allows the user to move to the previous step and choose a different technique for handling their null values. If this option is chosen, the updated data file is replaced with the new handling technique they choose. The final option is the click the ”Proceed to outliers identification”. As previously mentioned, the user does not have this option if they chose to ignore null values as shown in Figure 21.

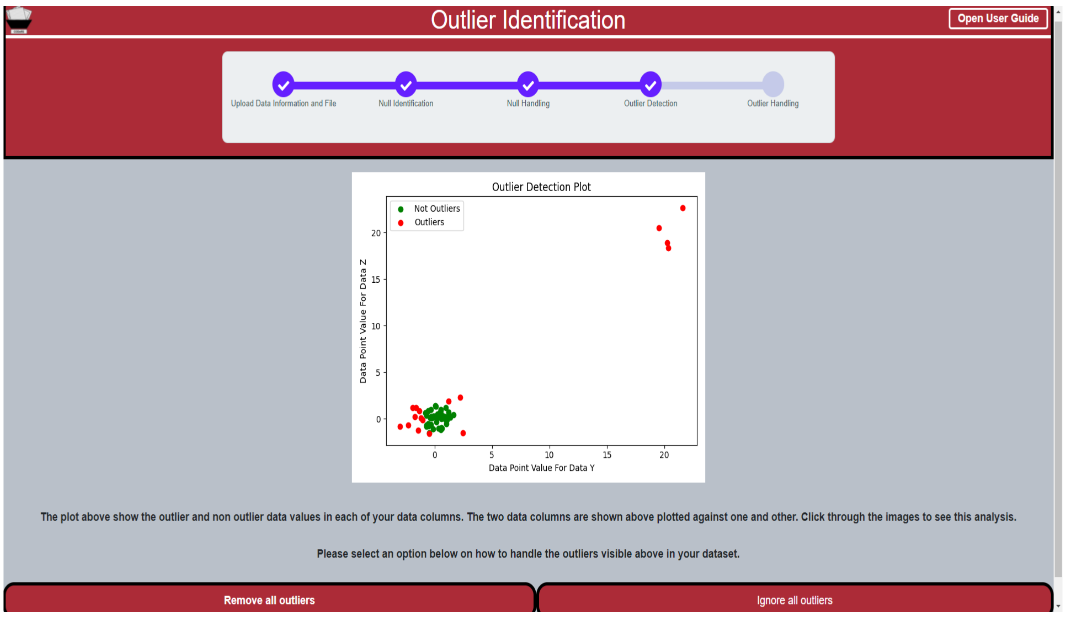


Success!

Nulls values have been ignored and left in your dataset. You are unable to move onto outlier detection.

Figure 21: Web Application Nulls Handled Page When Nulls Are Ignored

Clicking on the ”Proceed to outliers identification” redirects the user to the outlier identification page as shown in Figure 22. Similar to the null identification step, our application generates a series of images to represent the presence of outliers in the user's dataset. Each image shows the user the distribution of values for a different column, highlighting the outliers in red.



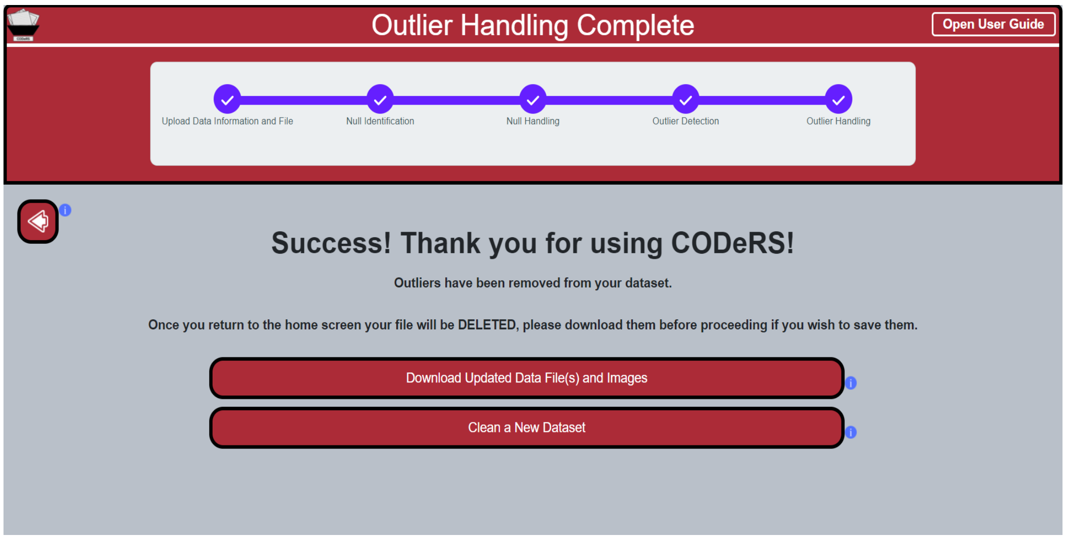
The plot above show the outlier and non outlier data values in each of your data columns. The two data columns are shown above plotted against one and other. Click through the images to see this analysis.

Please select an option below on how to handle the outliers visible above in your dataset.

Figure 22: Web Application Outliers Identified Page

From here, the user can select to remove or ignore outliers in the dataset. Clicking on the ”Remove all nulls” button creates a new dataset where tuples identified as outliers are removed entirely. In contrast, clicking on the ”Ignore all nulls” button results in no change to the user's dataset. A user can proceed to the next page by clicking any of these buttons.

During the transition from identifying null values to completing the cleaning step for handling null values, the outliers are handled based on the button pressed by the user. As shown in Figure 23 below, the following and final page confirms the completion of the outlier handling step and the cleaning process.



Success! Thank you for using CODeRS!

Outliers have been removed from your dataset.

Once you return to the home screen your file will be DELETED, please download them before proceeding if you wish to save them.

Figure 23: Web Application Outliers Handled Page

Similar to the completion page for null handling, the user is given several actions to choose from. The "Download Updated Data File(s) and Images" button allows the user to download their original data file and both new data files created from their null handling technique and outlier handling technique. The data file created following the outlier handling step contains changes made from both cleaning steps. This button also allows users to download all of the images shown in the previous steps from both outlier and null identification. Clicking the "Clean a New Dataset" button ends the cleaning process for this dataset and allows the user to start the entire process over. The back arrow provided in the top left corner allows the user to move to the previous step and choose a different technique for handling their null values. If this option is chosen, the updated data file is replaced with the new handling technique they choose. This page marks the end of the entire cleaning process for the user.

5 Results

After our project was fully implemented, we tested the functionality and design of our application to make improvements and provide recommendations for future MQP teams. We performed two types of tests to gain an understanding of the effectiveness of our application for users. Our first test was to performing data cleaning on a series of artificially generated datasets using our web application. Based on how well the data was cleaned, we could confirm how well the python back-end handles the actual data cleaning portion. Due to CODeRS being a web application, we spent just as much time testing the front-end portion through user studies. These user studies involved non-computer science members of our research team utilizing the application and providing us with their honest opinions on how well the application performs in terms of both design and usability.

5.1 Datasets

To test the effectiveness of our application in terms of data cleaning, we created four datasets and attempted to clean them using CODeRS. Each dataset was created using a python script that injects outliers and null values into the dataset to allow us to test how well our application is able to identify and clean both outliers and null values.

5.1.1 Dataset 1

The first dataset tested using the CODeRS web application is a one-dimensional dataset injected with nulls and outliers. The script used to create this dataset and raw data plots is located at our GitHub repository. The dataset itself is also located in our GitHub repository. This dataset was created using the NumPy library. This dataset was generated by randomly sampling values from a normal distribution with a set mean and standard deviation. We then used a random probability to insert null values and outliers throughout the dataset. This nature of inserting null values produces MCAR null values. A visualization of this raw data is shown below in Figure 24.

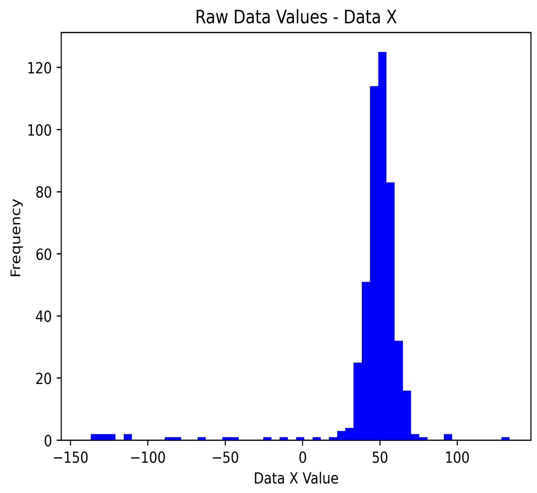


Figure 24: Dataset 1 - Raw Data

During the null identification step, the application was able to correctly identify that the dataset contained 30 null values as shown below in Figure 25.

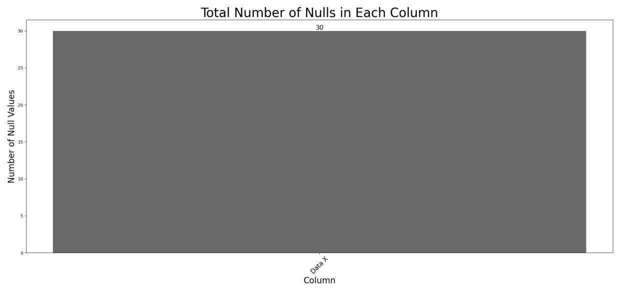


Figure 25: Dataset 1 - Null Count

In the outlier detection portion of the CODeRS application, outliers were identified using the DBSCAN outlier identification method. This resulted in Figure 26 below, identifying the majority of the outlier points as outliers except for one small cluster of points located around -50.

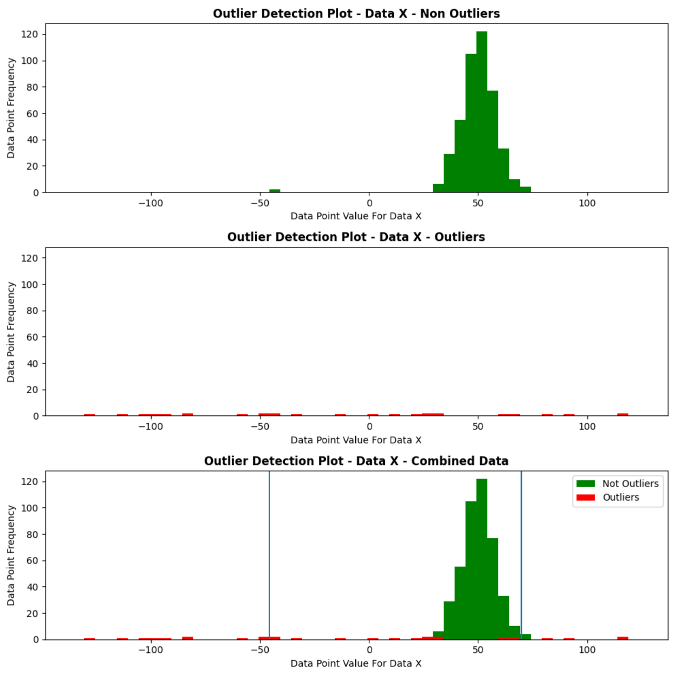


Figure 26: Dataset 1 - Outlier Identification

This test succeeded as all of the null values and a majority of the outliers were correctly identified and handled without needing to input any information other than the dataset type and dataset itself.

5.1.2 Dataset 2

The next dataset used in our testing is five-dimensional. Columns Data X, Data Z, and Data A were generated using a random sample from a normal distribution. In contrast, column Data Y was determined using a random sample from a uniform distribution, and column Data B was determined using a random sample from a Wald distribution. We then used a random probability to insert null values and outliers throughout the dataset, similar to the process used to generate the previous dataset. To make the dataset more interesting, some columns were changed based on changes performed on other columns. This first occurs with the Data X column, where if any value is set to null in another column, the same index of the Data X column is set to be null. This causes Data X to be null if any other value in the tuple is set equal to null. This accurately simulates MAR null values. Similarly, in Data Z, if any tuple is an outlier in the other columns, the item at the same index in Data Z is set to be an outlier. This results in Data Z points being outliers at every position another data point is an outlier in another column. The script to create this dataset and display its raw data distributions is located on our GitHub repository. The dataset itself is also located on our GitHub repository. The most notable part of running this dataset through CODeRS is that the null images correctly show the correlation between the null columns. This is shown below in Figure 27.

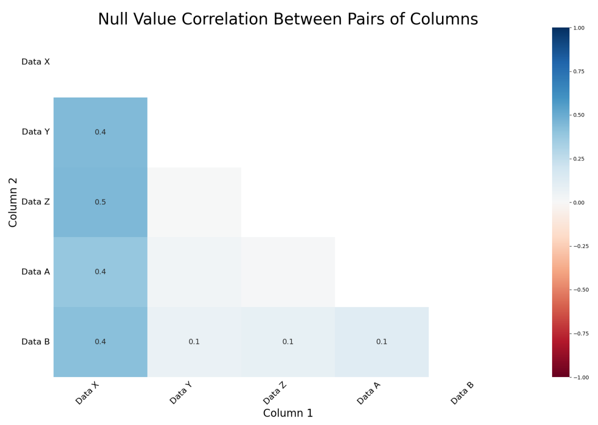


Figure 27: Dataset 2 - Null Correlation Matrix

The correlation matrix figure is able to show the null values in column Data X having a strong correlation with null values in every other column. From this plot, a user is able to determine that their Data X column is most prone to having null values based on other null values in their dataset. After this step, all null values were corrected and all outliers were removed. CODeRS was able to successfully handle all nulls and most outliers in a similar fashion to the previous dataset, showing that CODeRS works on multivariate datasets.

5.1.3 Dataset 3

The next dataset used was a six dimensional dataset that is equivalent to the previous dataset but with an additional column. This extra column was set to be the sum of all other columns after nulls and outliers were inserted into the dataset. This column was then labeled as the target column to test how CODeRS handles datasets with a target column. The script that created this dataset is located on our GitHub repository. The dataset itself is also located on our GitHub repository.

After running this dataset through the CODeRS web application, the process was able to successfully handle the target variable according to our desired functionality. When handling nulls, it removes all null values in the target column regardless of remove nulls or correct nulls being selected on the null identification page. On the outlier identification page, it then does not perform outlier identification on the target column and does not display a plot for the column values. It only removed tuples based solely on non-target variable columns. Thus, the CODeRS web application cleaned the data as we had intended.

5.1.4 Dataset 4

The last dataset included for testing purposes is a four dimensional time-series dataset. Each column represents a different sine or cosine curve with a varying amount of noise and a different frequency. Null values and outliers were then injected at random points. This dataset did not have any columns whose values were reliant off of other columns. The purpose of this dataset was to test CODeRS and its handling of n-dimensional time series datasets. The code to create this dataset is located on our GitHub repository. The dataset itself is also located on our GitHub repository. To highlight the cleaning process, we focus on the Data Z column shown below in Figure 28.

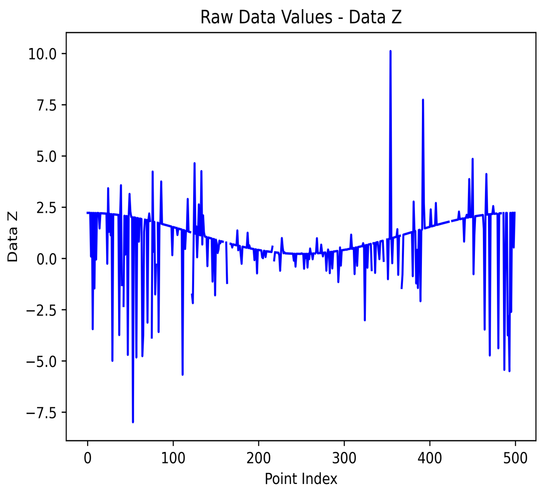


Figure 28: Dataset 4 - Raw Data

This dataset starts off with a few extreme outliers along with 29 null values. CODeRS was able to successfully identify and correct these null values to allow outliers to be identified. The resulting column's values following outlier identification and removal is shown below in Figure 29.

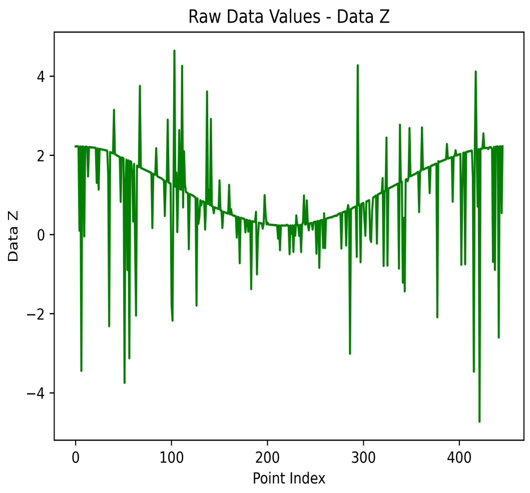


Figure 29: Dataset 4 - Cleaned Data

The transition from the raw data to the cleaned data highlights the effectiveness of CODeRS since it removes outliers to the point where the majority of points that remain have minimal deviation from the main curve. As intended, this example dataset shows that CODeRS successfully handles nulls and outliers in a time series.

5.2 User Study: Materials Science

Throughout the development of our web application, we held two user studies over Zoom to test the usability, complexity, and appearance of our application. Our participants consisted of three top-notch materials scientists from WPI's Data Driven Materials Science research group. In the real world, this application will be used by users who are experts in a field outside of data science but wish to implement data science solutions. Our participants' expertise in a separate domain as well as their past utilization of data analytics in their research made them the perfect candidates for our application. During each user study, the participants shared their screen and attempted to complete the tasks we provided, while we observed and collected notes regarding their interaction with the application. The tasks we developed are provided below:

- Upload a csv file
- Identify nulls
- Remove all nulls
- Download files
- Navigate to downloaded files
- Choose to start the process over with a new csv file
- Upload a new csv file and input a target variable
- Choose to Ignore nulls
- Return to the previous step
- Correct all nulls
- Move on to the next step
- Remove outliers
- Download files
- Start the process over
- Perform any additional tasks

For each task, we asked them to complete the task and provide opinions on how difficult the task was to complete. Additionally, each time a user was redirected to a new page, we asked them to comment on the complexity of the layout and how intuitive the page functionalities were. This allowed us to identify overly technical or unclear aspects of our application that we were blinded to as developers and data scientists. We also asked our participants to provide any ideas for solutions to these flaws in functionality or design to guide our next stage of development. This ensured that these flaws would be fixed for the benefit of the user.

Our first user study involved one participant. The application was fully fleshed out for this user study with full functionality. However, this version of our application had not yet received feedback, so the design and amount of provided information at each step was minimal. The following table shows the opinions presented during our user study as well as the resulting changes made in our web application.

Table 2: User Study 1 Results and Solutions

Comments	**Solutions**
It is unclear which data fields are required on the first page.	We added asterisks to the required fields, similar to other popular web applications.
The fillable data fields on the first page are confusing.	We provided information about each text field at the top of the page to further describe the purposed of each fields and what type of value should be inserted.
The null images are unclear and not easy to understand what they represent.	We added axis labels and a title to each image. We also provided further descriptions in the user guide.
The options for handling outliers and nulls were unclear and not explained.	We added additional information about each option at the top of the page to further explain what each technique does. Additionally, we provided examples of when to use each technique in the user guide.
There is no option to go back to a previous page after selecting a method for handling nulls or outliers.	We added a back button to allow the user to easily move to a previous page without downloading the wrong image and data files.
The user's progress throughout the entire cleaning process was unclear.	We developed a progress bar, similar to the pizza tracker used on Domino's website, to help the user understand which step they are on while using our application.
The terminology used in the application is confusing. An example is the word "null".	We defined these technical terms and processes in the user guide.
The wording on the completion pages (i.e. once outlier and null handling is finished) is confusing.	All the wording was updated to be clearer and more consistent. We also added more text segments to better explain what each option means.
Updated data files are named the same regardless of the handling technique that was used, which is confusing. For example, removing nulls and correcting nulls both lead to a data file named null_corrected.csv.	We changed the naming of the data files to be generated based on the technique used. Additionally, we did not add an extra data file if the dirty data was ignored.
The order of the buttons on the last page do not match the order of the buttons on the null completion page even though they contain the exact same buttons.	We rearranged the order of the buttons to match on both pages.
The gray used on the application is harsh on the eyes.	We changed the color to a lighter, more appealing gray.

One major theme of this user study was the lack of information provided during each step of the process. Our participant often struggled to determine what would occur if a button was pressed. They also struggled to figure out which cleaning technique to use for outliers and nulls due to the lack of explanation on the images and page itself. Due to the knowledge we gained through our development of the application, we did not realize that aspects of our application would not be intuitive to non-data science experts. From our perspective, each page was self-explanatory. However, to a domain expert in an entirely different field, each step of the process was entirely new. These comments from our participant alone were highly valuable to us and proved the importance of these user studies in the development of our application. As a result, we provided an increased amount of text descriptions above each set of buttons and field inputs to further explain each element in our application.

Our second user study involved two participants. The 3-day break between user studies gave us the ability to make many changes to the web application prior to the second user study. Throughout this user study, the participants affirmed the comments made in our first user study. They commented on how easy the application was to understand. Some of the features they found especially useful were the go back buttons, the pizza tracker progress bar, and the custom names of updated data files. The following table shows the opinions expressed in this second user study as well as the changes made in our application as a result.

Table 3: User Study 2 Results and Solutions

Comments	**Solutions**
Throughout the application, the additional information regarding data fields, buttons, and general information about each step was incredibly useful. However, it was bulky and intimidating as text at the top of each page.	We moved the information from the top of the page to information bubbles next to the necessary location to increase simplicity.
The data fields on the home page are too large and prevent the user from seeing everything they see in one glance.	We reduced the width of each data field and button to half the size of the web page.
The format of the column names that appear when the Target Variable error shows on the first page is confusing to non-technical users.	We adjusted the formatting to be clearer as to what the specific column names in the dataset are.
The Target Variable error does not stand out enough.	We put this error into a banner with different colors to grab the user's attention better.
The application itself is super cool!	We are pleased the application is enjoyed by the users.
It would be useful to see all images at once for a side-by-side comparison rather than in a carousel on the null and outlier handling pages.	For time's sake, we chose to add a recommendation to future MQP team's to place these images as thumbnails on the page at once and allow users to click to enlarge the image.
For users who are inexperienced in data science visualizations, the images to represent the state of outliers and nulls in the dataset are confusing.	We included thorough details about what these plots mean as well as how to use them to determine the best handling method for their dataset.
The order of the null handling buttons do not reflect the best order at which a user should choose a technique.	We adjusted the order of the buttons to start with "Correcting Null" since correction is the preferred handling method in a majority of situations. We also bolded this button to emphasize that it is the best choice.
The button for moving to the previous page is not intuitive as a "Go Back" button.	Since icons are powerful, we changed the "Go Back" button to a left arrow and placed it in the top left corner of the completion pages.
The transition from handling dirty data to the process being complete is quick, which causes users to be confused about if the process occurred or not.	We added "Success!" in large text at the top of each completion page to help signify that the process succeeded.
The horizontal orientation of the buttons on the completion pages is not appealing.	We moved these buttons into a vertical alignment for visual appeal.
It is not clear which data point are considered outliers in the outlier plots.	We added a legend to explain that data points colored in red are outliers.
It would helpful to allow users to choose which outliers they want to delete.	For time's sake, we added this additional functionality as a recommendation for future MQP teams.
The pizza track progress bar is super helpful in tracking progress throughout the application.	We chose to keep this feature due to this positive feedback.

Overall, the users in this study appreciated the additional information provided at the top of each page, but they found it cumbersome and intimidating. They mentioned that first time users may have a difficult time digesting the information in this paragraph format. In order to alleviate this intimidation, we moved this information into a series of information bubbles next to the appropriate elements. For example, on the first page of our application, an information bubble was added to each text field to further explain the field itself as well as the values that are accepted. This allowed us to provide our users with strong descriptions in a simple way. Another major talking point during this user study was adding more clarity and validation throughout our application. For example, our users expressed interest in adding a notification to the completion pages to highlight that the cleaning process finished. We saw this same need for more clarity in our first user study when our user expressed interest in having a progress bar to show which step the user is in the process. As a result, we went through our application and determined pages where we could add a higher sense of clarity for our users. Additionally, we developed a user guide to provide users with a thorough description of each step of our cleaning process. This user guide gave us the opportunity to provide more detailed explanations of our process to our users while keeping our application simple. This user guide is accessible on every page and pops up in a new window to allow users to read the user guide while using our application.

These user studies provided invaluable feedback that was used to improve the web application immensely. In addition to the user study itself, we wanted to gather more information regarding the design and usability of our application. Thus, our participants completed a Nielsen Heuristic survey [49], as shown in Appendix 7.4. This survey was created with questions based on the heuristic presented on the Nielsen Norman Group website. The results of these surveys are shown in Appendix 7.3. Though we had a small dataset, we saw a significant improvement in every usability standard on our survey.

6 Conclusions and Future Work

Data cleaning is a crucial step that must be taken in order to derive insights from a dataset. However, due to the lack of literature and implementations of high quality cleaning techniques, non-data science experts struggle to create accurate data science solutions. We built CODeRS with the sole purpose of creating an application that automates the cleaning process with robust cleaning techniques for non-technical users. Through our research, we found that Multiple Imputation by Chained Equations is the ideal null correction method, while DBSCAN and isolation forests are the ideal outlier identification methods for static and time series data respectively. Utilizing the previously mentioned state-of-the-art cleaning techniques, we automated the cleaning process and provided direct functionality for cleaning. We utilized the Agile methodology as well as a set of best practices for software development to develop our application from scratch. Through a series of user studies, we determined points of weakness in our application, such as the lack of user instructions, and refined our application in multiple iterations. We tested the functionality of our final application on a series of datasets, and CODeRS accurately identified and corrected a majority of the null values and outliers found.

Our work over the past three terms has laid the foundation for the CODeRS web application. With this foundation, we hope that future MQP teams will expand upon our work and continue to improve CODeRS. To help future teams continue where we left off, we compiled a list of recommendations for future work on CODeRS based on the feedback we received from our user studies. Our recommended additional functionalities for future MQP teams to add listed below.

- Time Series Specific Visualizations
- Raw Data Visualizations Prior to Preprocessing
- A Preprocessing Section to Allow for Techniques such as Normalizing Data
- Page Redirects to Maintain a User's Position in the CODeRS Process
- A Custom Server for CODeRS
- An Image View on the Null Identification Page and Outlier Identification Page that Shows Users All the Images and Enlarges them on Click
- Support for More File Upload Types such as Excel
- Functionality to Support Categorical Data

Each recommendation above serves to further expand the CODeRS application. One recommendation that highlights this is the addition of a preprocessing section that allows for techniques such as normalizing data. This would allow the base functionality of the CODeRS to be utilized while expanding it to include more use cases.

Upon reflection of our work, we are proud to declare that we have accomplished our project goals. We are hopeful that CODeRS will highly benefit data analyses performed in various fields, including the Materials Science and Medical fields, by providing them with a simple, automated tool for cleaning their dataset. CODeRS ability to quickly clean an arbitrarily sized dataset using our recommender system solves the issues previously presented in the Materials Science field. Additionally, the standardized recommender system implemented in CODeRS provides medical professionals with the accurate and automated tools they have needed to perform data analyses. We are excited to see how CODeRS is used in the real world to standardize the data cleaning process and improve the quality of data science solutions for all of our users.

7 Appendices

7.1 Web Application Link

Our web application can be found at `http://coders-alpha.herokuapp.com/`.

7.2 User Manual

CODeRS User Manual can be found on the website at User Manual Link.

7.3 User Study Documents

Listed below are each of our documents related to our User Studies.

Initial task list prior to any user study taking place: User Study Task List V1.

Notes on User Study 1: User Study with Jack.

Second tasks list developed after User Study 1: User Study Task List V2.

Notes on User Study 2: User Study with Matt.

Notes on User Study 3: User Study with Professor Cote:

Nielsen Heuristic Survey: User Survey Results

7.4 User Study Survey

A survey was given to ever individual who participated in our User Study to fill out based on their experience with our web application. They were presented with every question below and given the following prompt.

"Please leave feedback in each category on a scale of 1-4. 1 indicates the current state of the application not properly satisfying the description of the category while a 4 indicates the current state satisfies the description."

Survey Categories:

- Visibility of System Status - The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time.
- Match between system and the real world - The design should speak the users' language. Use words, phrases, and concepts familiar to the user, rather than internal jargon. Follow real-world conventions, making information appear in a natural and logical order.
- User control and freedom - Users often perform actions by mistake. They need a clearly marked "emergency exit" to leave the unwanted action without having to go through an extended process.
- Consistency and standards - Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform and industry conventions.
- Recognition rather than recall - Minimize the user's memory load by making elements, actions, and options visible. The user should not have to remember information from one part of the interface to another. Information required to use the design (e.g. field labels or menu items) should be visible or easily retrievable when needed.

- Flexibility and efficiency of use- Shortcuts (hidden from novice users) may speed up the interaction for the expert user such that the design can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
- Aesthetic and minimalist design - Interfaces should not contain information which is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility.
- Help and documentation - It's best if the system doesn't need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks.

7.5 Heroku Help

Our instructions for hosting our web application through Heroku can be found at Heroku Setup Document. Following these instructions allows any team to host this application.

References

[1] A. Gaur, B. Scotney, G. Parr, and S. McClean, "Smart city architecture and its applications based on iot," Procedia computer science, vol. 52, pp. 1089–1094, 2015.

[2] J. Schobel, R. Pryss, M. Schickler, M. Ruf-Leuschner, T. Elbert, and M. Reichert, "End-user programming of mobile services: Empowering domain experts to implement mobile data collection applications," in 2016 IEEE International Conference on Mobile Services (MS), IEEE, 2016, pp. 1–8.

[3] S. N. Shukla and T. A. Champaneria, "Survey of various data collection ways for smart transportation domain of smart city," in 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2017, pp. 681–685. DOI: 10.1109/I-SMAC.2017.8058265.

[4] A. Shybeko, Use of predictive modeling in healthcare, Last accessed 12 February 2022, 2021. [Online]. Available: https://light-it.net/blog/use-of-predictive-modeling-in-healthcare/.

[5] J. Van den Broeck, S. A. Cunningham, R. Eeckels, and K. & Herbst, "Data cleaning: Detecting, diagnosing, and editing data abnormalities.," PLoS Medicine, vol. 2, no. 10, 2005. [Online]. Available: https://dx.doi.org/10.1371%5C%2Fjournal.pmed.0020267.

[6] R. J. Hyndman and G. Athanasopoulos, Forecasting: Principles and practice, Last accessed 12 February 2022, 2018. [Online]. Available: https://otexts.com/fpp2/components.html.

[7] N. I. of Standards and Technology, Use of predictive modeling in healthcare, Last accessed 12 February 2022, 2012. [Online]. Available: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc443.htm.

[8] M. W. Watson, "Time series: Cycles," International Encyclopedia of the Social & Behavioral Sciences: Second Edition, 2015. [Online]. Available: https://www.princeton.edu/~mwatson/papers/isb201119.pdf.

[9] G. for Geeks, What is a trend in time series? Last accessed 12 February 2022, 2021. [Online]. Available: https://www.geeksforgeeks.org/what-is-a-trend-in-time-series/.

[10] E. Acuna and C. Rodriguez, "A meta analysis study of outlier detection methods in classification," Technical paper, Department of Mathematics, University of Puerto Rico at Mayaguez, vol. 1, p. 25, 2004.

[11] A. Wooditch, N. J. Johnson, R. Solymosi, J. Medina Ariza, and S. Langton, "The normal distribution and single-sample significance tests," A Beginner's Guide to Statistics for Criminology and Criminal Justice Using R, pp. 155–168, 2021.

[12] O. Maimon and L. Rokach, "Data mining and knowledge discovery handbook," 2005.

[13] V. Hodge and J. Austin, "A survey of outlier detection methodologies," Artificial Intelligence Review, vol. 22, pp. 85–126, Oct. 2004. DOI: 10.1023/B:AIRE.0000045502.10941.a9.

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

[15] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," 2008 Eighth IEEE International Conference on Data Mining, pp. 413–422, 2008. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4781136.

[16] J. L. Peugh and C. K. Enders, "Missing data in educational research: A review of reporting practices and suggestions for improvement," Review of educational research, vol. 74, no. 4, pp. 525–556, 2004.

[17] J. H. Ware, Interpreting incomplete data in studies of diet and weight loss, 2003.

[18] N. R. Council et al., "The prevention and treatment of missing data in clinical trials," 2010.

[19] M. J. Lipinski, C. A. Cauthen, G. G. Biondi-Zoccai, A. Abbate, B. Vrtovec, B. V. Khan, and G. W. Vetrovec, "Meta-analysis of randomized controlled trials of statins versus placebo in patients with heart failure," The American journal of cardiology, vol. 104, no. 12, pp. 1708–1716, 2009.

[20] D. B. Rubin, "Inference and missing data," Biometrika, vol. 63, no. 3, pp. 581–592, 1976.

[21] A. B. Pedersen, E. M. Mikkelsen, D. Cronin-Fenton, N. R. Kristensen, T. M. Pham, L. Pedersen, and I. Petersen, "Missing data and multiple imputation in clinical epidemiological research," Clinical epidemiology, vol. 9, p. 157, 2017.

[22] T. A. Myers, "Goodbye, listwise deletion: Presenting hot deck imputation as an easy and effective tool for handling missing data," Communication methods and measures, vol. 5, no. 4, pp. 297–310, 2011.

[23] R. Little, M. Cohen, K. Dickersin, S. Emerson, J. Farrar, J. Neaton, W. Shih, J. Siegel, and H. Stern, "The design and conduct of clinical trials to limit missing data," Statistics in medicine, vol. 31, no. 28, pp. 3433–3443, 2012.

[24] G. J. Van der Heijden, A. R. T. Donders, T. Stijnen, and K. G. Moons, "Imputation of missing values is superior to complete case analysis and the missing-indicator method in multivariable diagnostic research: A clinical example," Journal of clinical epidemiology, vol. 59, no. 10, pp. 1102–1109, 2006.

[25] G. King, J. Honaker, A. Joseph, and K. Scheve, "List-wise deletion is evil: What to do about missing data in political science," in Annual Meeting of the American Political Science Association, Boston, 1998.

[26] A. Donner, "The relative effectiveness of procedures commonly used in multiple regression analysis for dealing with missing values," The American Statistician, vol. 36, no. 4, pp. 378–381, 1982.

[27] A. F. Hayes, M. D. Slater, and L. B. Snyder, The Sage sourcebook of advanced data analysis methods for communication research. Sage, 2008.

[28] W. G. Madow, H. Nisselson, I. Olkin, and D. B. Rubin, Incomplete Data in Sample Surveys: Theory and Bibliographies. Academic Press, 1983, vol. 2.

[29] P. L. Roth, "Missing data: A conceptual review for applied psychologists," Personnel psychology, vol. 47, no. 3, pp. 537–560, 1994.

[30] C. M. Musil, C. B. Warner, P. K. Yobas, and S. L. Jones, "A comparison of imputation techniques for handling missing data," Western Journal of Nursing Research, vol. 24, no. 7, pp. 815–829, 2002.

[31] D. B. Rubin, "Multiple imputations in sample surveys-a phenomenological bayesian approach to nonresponse," in Proceedings of the survey research methods section of the American Statistical Association, American Statistical Association, vol. 1, 1978, pp. 20–34.

[32] J. C. Wayman, "Multiple imputation for missing data: What is it and how can i use it," in Annual Meeting of the American Educational Research Association, Chicago, IL, vol. 2, 2003, p. 16.

[33] J. L. Schafer and J. W. Graham, "Missing data: Our view of the state of the art.," Psychological methods, vol. 7, no. 2, p. 147, 2002.

[34] A. R. T. Donders, G. J. Van Der Heijden, T. Stijnen, and K. G. Moons, "A gentle introduction to imputation of missing values," Journal of clinical epidemiology, vol. 59, no. 10, pp. 1087–1091, 2006.

[35] T. Ezzati-Ricel, W. Johnson, M. Kharel, R. Little, D. Rubin, and J. Schafer, "Imputations in nchs health examination surveys," in Proceedings:... Annual Research Conference, US Department of Commerce, Bureau of the Census, 1995, p. 257.

[36] J. L. Schafer and M. K. Olsen, "Multiple imputation for multivariate missing-data problems: A data analyst's perspective," Multivariate behavioral research, vol. 33, no. 4, pp. 545–571, 1998.

[37] J. A. Sterne, I. R. White, J. B. Carlin, M. Spratt, P. Royston, M. G. Kenward, A. M. Wood, and J. R. Carpenter, "Multiple imputation for missing data in epidemiological and clinical research: Potential and pitfalls," Bmj, vol. 338, 2009.

[38] N. J. Horton and S. R. Lipsitz, "Multiple imputation in practice: Comparison of software packages for regression models with missing variables," The American Statistician, vol. 55, no. 3, pp. 244–254, 2001.

[39] T. E. Raghunathan, J. M. Lepkowski, J. Van Hoewyk, P. Solenberger, et al., "A multivariate technique for multiply imputing missing values using a sequence of regression models," Survey methodology, vol. 27, no. 1, pp. 85–96, 2001.

[40] I. R. White, P. Royston, and A. M. Wood, "Multiple imputation using chained equations: Issues and guidance for practice," Statistics in medicine, vol. 30, no. 4, pp. 377–399, 2011.

[41] G. Ambler, R. Z. Omar, and P. Royston, "A comparison of imputation techniques for handling missing predictor values in a risk model with a binary outcome," Statistical methods in medical research, vol. 16, no. 3, pp. 277–298, 2007.

[42] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, "Activeclean: Interactive data cleaning while learning convex loss models," arXiv preprint arXiv:1601.03797, 2016.

[43] S. Krishnan, M. J. Franklin, K. Goldberg, J. Wang, and E. Wu, "Activeclean: An interactive data cleaning framework for modern machine learning," in Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 2117–2120.

[44] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu, "Boostclean: Automated error detection and repair for machine learning," arXiv preprint arXiv:1711.01299, 2017.

[45] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman, "Missing value estimation methods for dna microarrays," Bioinformatics, vol. 17, no. 6, pp. 520–525, 2001.

[46] A. K. Waljee, A. Mukherjee, A. G. Singal, Y. Zhang, J. Warren, U. Balis, J. Marrero, J. Zhu, and P. D. Higgins, "Comparison of imputation methods for missing laboratory data in medicine," BMJ open, vol. 3, no. 8, e002847, 2013.

[47] S. Ghorbani and M. C. Desmarais, "Performance comparison of recent imputation methods for classification tasks over binary data," Applied Artificial Intelligence, vol. 31, no. 1, pp. 1–22, 2017.

[48] A. Jadhav, D. Pramod, and K. Ramanathan, "Comparison of performance of data imputation methods for numeric dataset," Applied Artificial Intelligence, vol. 33, no. 10, pp. 913–933, 2019.

[49] J. Nielsen, "Enhancing the explanatory power of usability heuristics," in Proceedings of the SIGCHI conference on Human Factors in Computing Systems, 1994, pp. 152–158.