Arman Jordan
6/9/2023

<u>Large dataset analysis</u>

Introduction:

I have been given a dataset that is missing a large amount of information. I have not been given any of the column names, and am being asked to create a model to predict the value of a label(which can be any number 0-3) based on the values of 14 other features. The dataset has around 1500 entries.

The dataset I have been given is lacking a lot of information, but I have been given enough that I can draw certain conclusions. For one, all of the columns are pretty full of data, there aren't any columns filled with missing data, so there aren't too many rows that need to be fixed. This means I can avoid trying to impute data(and maybe accidentally invalidating it), and can just remove the rows with missing data. On the downside, this also means that there aren't any feature columns I can immediately write-off as not having a large effect on the label, so I will have to clean and keep all of them. On top of this, since none of the column names are included, I cannot use common sense to reason through which columns might have a larger effect on the output label. There are some columns where I can assume the context, like the column filled with sports names, the column filled with animal names, the column that uses kg, and the column that uses mph, but even then I cannot assume which column has a higher impact on the label so I must clean all of them.

I feel that KNN would be a good model to use for this data just because without knowing the title of the columns I cannot assume trends, but predicting based on alike data points is sure to be accurate. But because of the sheer size of the dataset, KNN would involve iterating over a lot of data, and I would also consider using a logistic regression classifier. But until I am able to visualize a clean dataset, I will not know for sure which model to use.

Data Cleaning:

There were four main steps that I took with data cleaning. Removing the mph and kg units after values, removing the rows with missing values, assigning data-types to the columns, and converting the strings to lowercase. I implemented a separate function for each of the aforementioned processes, and each one had its own particularities. The function extract_numbers() for removing the mph and kg units iterates through the dataframe, removing units as it goes, but it serves a few smaller functions as well. The function casts all of the values that can be numbers as ints and floats, instead of just strings that look like numbers. This allows the later assign() function to more easily typeset each column. The function also assigns all of the “NaN, nan, n/a, N/A, none, ?, null” invalid strings to a single value, numpy.nan. This allows the next function, remove_nan(), to more easily do its job of removing the rows with invalid values.

When handling rows with null values, I had two options. To remove the row altogether, or to impute the data. I considered imputing the data, but decided not to for a number of reasons. 1) I know very little about the context of the data, so I did not feel confident simply placing the mean of the column in the entry, 2) When printing out the information about the dataframe, it is displayed that there are very few null values, so I wasn’t worried about losing too much data, and 3) there are 1,470 entries and only around 20 of those per column that show as having null values. When considering these things, it did not seem worth risking imputing inaccurate data when I would still have a very large amount of data to work with.

A choice I had to make along the way was whether or not to keep this implementation, because after I finally got it working I realized that there were more null rows than I had previously believed. Displaying the information of the dataframe didn’t capture all of the null rows, as rows with none and n/a were counted as rows with strings, not null rows. But I decided to keep the implementation because even though more rows were removed than I had expected, there were still a very large number of datapoints left.

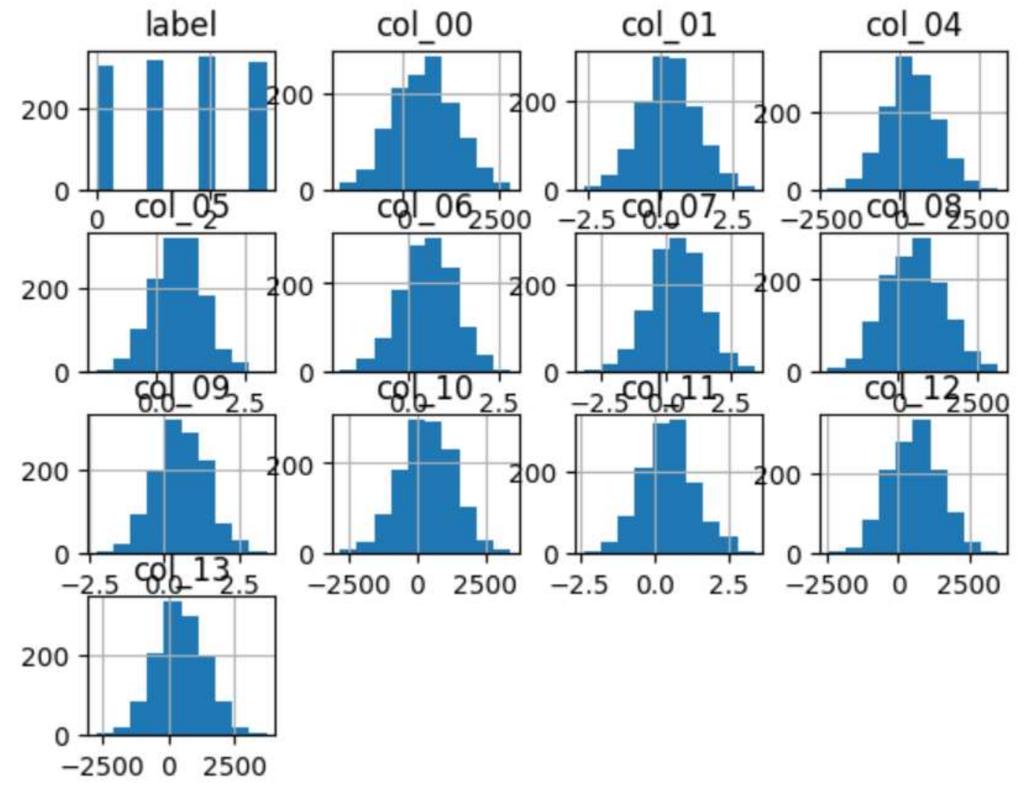
The next step in the process was to assign data types to the columns using assign_types(). This was a simple process, if a column contained anything other than a number at this point, it meant that it was a column of strings, and was assigned as such by the function. Otherwise, the column was assigned as a float, and was parsed for decimal numbers. If no decimal numbers were found, the column was then reassigned as an int. This process made it so that I could more easily use built-in pandas functions to read data, including viewing statistics for each numerical column.

The final step in this process was to convert all of the remaining strings to lowercase using lowercase(). This was so that when I one-hot encode the data later so I can use it with the other numerical values in the model, something like “MOTOR SPORTS” and “motor sports” will add weight in the same direction instead of being counted as two different values.

After cleaning the data, I decided to also use the one-hot function to transform columns 2 and 3 from categorical values into numbers so I can use them later when modeling.

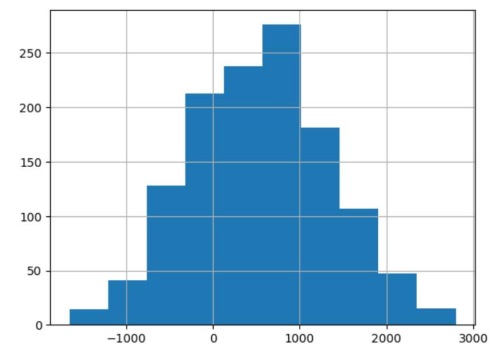
Visualization:

To visualize the data, I first created histograms to represent all of the columns, including the label.

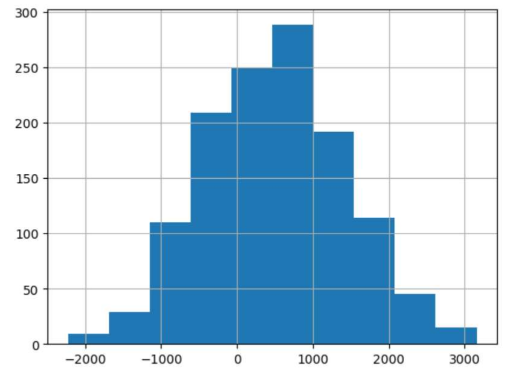


From this I noticed that there is a pretty even distribution of values 0-3 for the output label, and that everything has a pretty decent bell curve distribution of data.

When looking at these models, I noticed that col_00 and col_08, col_01 and col_10, and col_04 and col_13 all have very similar looking distributions. This could suggest a correlation, so I decided to take a closer look. When modeling col_00 and col_08 as larger histograms, their similarities were even more apparent.



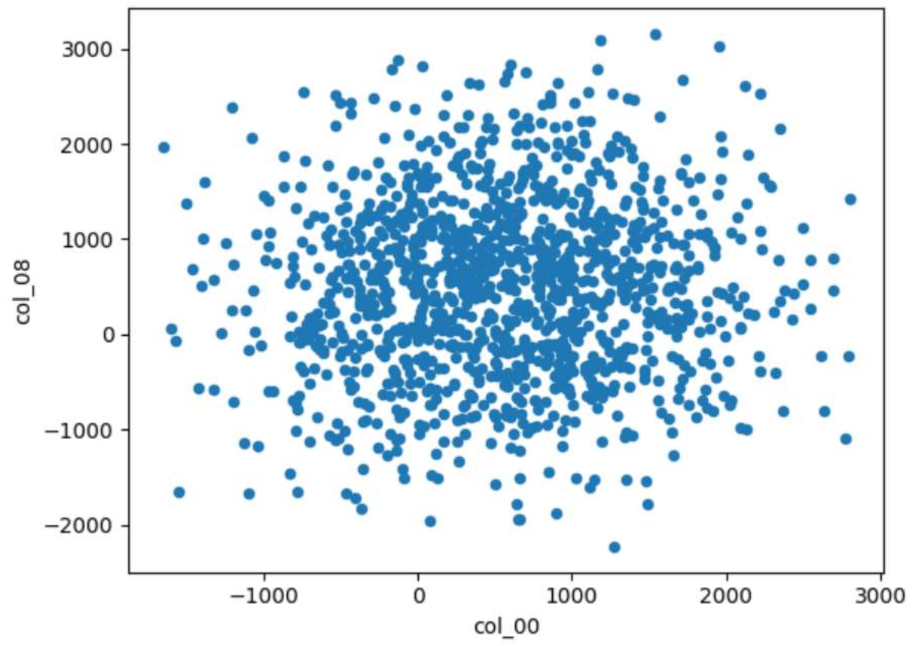
```
[14]:  <Axes: >
```



col_00 on the left, col_08 on the right

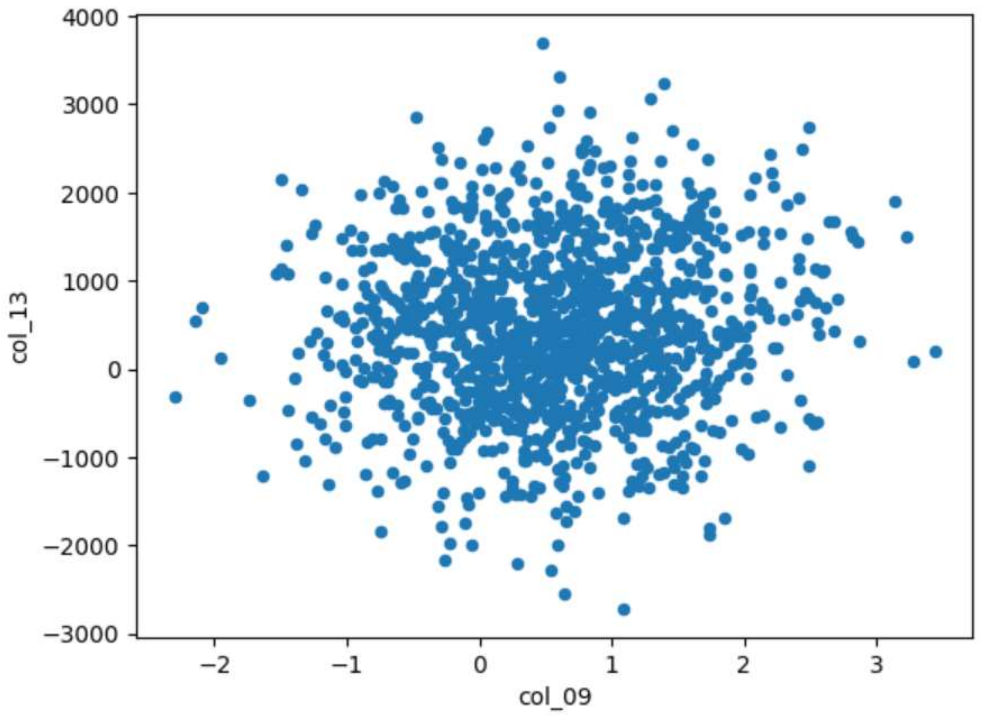
Unfortunately however, when graphing the data in a scatterplot, it wasn't immediately evident whether there was any correlation.

```
[10]:  <Axes: xlabel='col_00', ylabel='col_08'>
```



The only thing that I learned from this scatterplot was that the data for col_00 and col_08 is fairly similar. They both fall within the same ranges of -2000 to 3000, and are more densely located between -1000 and 2000. When treating 0,0 as the origin, you can notice that there is some positive correlation between the two, but it is difficult to definitively claim this. So I went back to the histograms, and noticed that col_09 and col_13 looked like they were both centered around positive values. So I graphed them as a scatterplot as well.

```
[14]:  <Axes: xlabel='col_09', ylabel='col_13'>
```



But again, the data seemed very sparse. There was again a slight positive correlation, but not enough to mean anything relevant. I began going through and graphing the correlation between all of the columns, and I realized that none of the scatterplots revealed any obvious correlation. They did give me information about data ranges, and I now understood that for all of the columns, values could be negative, but were primarily positive and centered around either -1 to 2, or -1000 to 2000. This was very interesting to me because I find it strange that data patterns like this could naturally emerge. I was very interested in what the column names were at this point, but was reassured that because all the data was centered so similarly, they could all pull a model towards a similar answer. I knew that since values all fell within similarly shaped ranges but on a different scale, I could easily preprocess the data and standardize the weights without ruining the data. I pressed on to the next step to see if my hypothesis was correct.

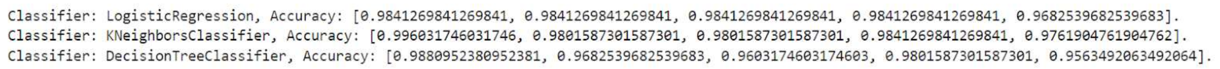
Modeling:

The classifiers I used were the logistic regression model, the K-nearest neighbor model, and the decision tree model. The KNN model had the highest accuracy, followed by the logistic regression curve, and in last place was the decision tree classifier. The standard deviation on the Decision Tree was the highest so there was some variation from the lowest accuracy, but that also means the accuracy could go even lower. KNN has the highest accuracy and 2nd lowest standard deviation, so it had the best performance but because it stores all of the 1000+ data points when doing its prediction, it is not the most efficient. Logistic regression has an extremely similar mean accuracy, only slightly below, and an even smaller standard deviation. Because of the nature of logistic regression, it involves storing a lot less data and can be used to quickly and cost-effectively make predictions, so I believe that that is the best model of the 3 for this dataset.

Model	Mean Accuracy	Standard Dev. of Accuracy
Linear Regression	0.9809	0.00634
KNN	0.9833	0.00682
Decision Tree(floor 10)	0.9698	0.01022

Analysis:

All three models ended up performing well enough on modeling the dataset that the significant difference test returned False for all of the comparisons. But there still were some particularities to note.



```
Classifier: LogisticRegression, Accuracy: [0.9841269841269841, 0.9841269841269841, 0.9841269841269841, 0.9841269841269841, 0.9682539682539683].
Classifier: KNeighborsClassifier, Accuracy: [0.996031746031746, 0.9801587301587301, 0.9801587301587301, 0.9841269841269841, 0.9761904761904762].
Classifier: DecisionTreeClassifier, Accuracy: [0.9880952380952381, 0.9682539682539683, 0.9603174603174603, 0.9801587301587301, 0.9563492063492064].
```

For the Decision Tree classifier, there were a few tests where the prediction accuracy fell short, where it reached 96% and 95% accuracy. I had predicted that the decision tree wouldn't be as accurate as the other models. I know that the decision tree can be extremely accurate, but to avoid overfitting and extremely long trees I set a floor of 10 as the max_depth. And when there are 1000+ data points, I predicted that binary splits wouldn't result in the most accurate results. That being said, 95% is still fairly accurate. But not as accurate as, say, the KNN model, which put out 99.6% accuracy followed by a few 98%s and one 97%. When beginning this task, I believed that KNN would have the highest accuracy, because I knew that there being a lot of data points would increase the performance. What I was surprised by was how accurate the logistic regression model was. It does make sense because the label we are predicting can only be one of 4 values, and in situations where we are basically dealing with classification, logistic regression curves can fit very well.

I believe that accuracy was the best evaluation metric to consider for this dataset. I know that if data is imbalanced it can be advantageous to measure things like precision or recall instead, but when viewing the histograms for the columns of the data, they were all balanced, which was made evident by their bell-curve shape. One thing that I would go back and experiment with changing in the data cleaning is to now see how imputing data instead of removing missing values affects the performance of the model. But I wouldn't say it's a problem with the cleaning, just an opportunity for further analysis. As explained in the previous paragraph, the statistical difference of decision trees having the lowest accuracy while KNN had the highest made sense and lined up with my initial predictions of which models would perform the best. I changed the values of the number of neighbors for the KNN model and the max_depth of the decision trees model multiple times, but found that the differences in results were negligible. With the KNN model, using a higher number of neighbors to consider(like 20) lowers the accuracy of the model by a couple tenths of a percent, so performance actually decreases. With the decision tree model, if I change the max_depth parameter to be much higher(like 100), the accuracy can increase a lot to 99%, but I worry about overfitting when the tree is allowed to get that specific on modeling each training set. My prediction of there being overfitting was also evidenced by the accuracy values returned from the cross-validations set. In the same cross-validation test where the model achieved 99% accuracy, the model also went as low as 96% accuracy. This high variance leads me to believe that overfitting occurred.

Conclusion:

In conclusion, although the K-nearest neighbor model consistently returned the highest accuracy, I believe that the logistic regression model is the best overall model to use. This dataset includes a lot of data points, and if this data is to be used in the future and more points are to be added, it would be most efficient to use a model that doesn't need to store all of the data to make a prediction. The accuracy of the regression model is on par with that of the KNN model, but if accuracy is the priority then the KNN could be considered as well. Regardless, because of the size of the dataset and its propensity to grow, the decision tree model should not be used to draw predictions. In terms of further investigation, it is still very unclear what the context of the data is, and I believe it would be very advantageous to understand what values each column represents. Once the context is understood, it would be much easier to decide which features have a greater impact on the output label, and further improve accuracy.