STAT 656 | Applied Analytics
Group 13 | Final Project Report

5/3/2018

Vinay Sairam Sashank Bandhakavi

Hari Prasad Anbalagan

Qiongyu Shi

Nishu Kurup

Boyan Zhang

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 PURPOSE

This document summarizes the problem statement, methodology and results for the final project for the course STAT 656 (Applied Analytics) . A data file containing consumer complaints submitted to the NTHSA was analyzed to build and validate the best analytics model to predict the probability of crash based upon topic and sentiment analysis. Both SAS EM and Python were used in this project to perform the analysis and the results, observations and conclusions are presented in the sections below.

The report is divided into several sections. The problem statement is shown in Section 2 along with the data dictionary. The SAS and Python results are presented in sections 3 and 4 respectively. For each analysis the overview of solution, data statistics, solution approach, results along with the conclusions and observations were outlined in detail. Finally, a comparative study of the python and SAS EM was done and is presented the section 5.

The Appendices consist of the python code, SAS diagrams, node properties and SAS codes along with some results from the python web scrapping exercise.

1.2 SOFTWARE AND SYSTEMS

The following software were utilized during the implementation of this project.

1. Microsoft Word/Excel 2016: For documentation and tabulations
2. SAS Enterprise Miner Workstation 14.3
3. Python 3.6.3 (Anaconda Spyder IDE 3.2.4)

2 PROBLEM STATEMENT

The problem statement for the analysis is presented in this section.

The problem is to build and validate the best model for predicting the probability of a crash based on the data file *"HondaComplaints.xlsx"*. These data consist of 5,330 consumer complaints submitted to the NTHSA for some Honda makes in years 2001-2003. The analysis is to be performed based upon the topic and sentiment model and upon the other data available in the project file. This involves the following:

1. Build a Topic Model that organizes these complaints into 7 groups.
2. Score the Sentiment for each complaint.
3. Merge the topic group information and sentiments back into the original data file.
4. Build the best decision tree to predict the probability of a crash.

5. Download the latest news on the Japanese airbag manufacturer "Takata" from API. Analyze these using topic and sentiment analysis.

The data dictionary for the data is shown the table below.

Table 2-1 Data dictionary for data file

DATA DICTIONARY: HondaComplaints
These data are consumer complaints submitted to the NTHSA

ATTRIBUTE	Type	DESCRIPTION
NthsaID	Interval	Record ID (Ignore)
Make	Binary	'honda' or 'accura'
Model	Nominal	'TL', 'ODYSSEY', 'CR-V', 'CL', 'CIVIC', or 'ACCORD'
Year	Nominal	2001, 2002, or 2003
State	Nominal	Two-letter State codes (ignore)
abs	Binary	'Y' or 'N' (anti-brake system)
cruise	Binary	'Y' or 'N' (cruise control)
crash	Binary	'Y' or 'N' (target)
mph	Interval	Miles per Hour (speed)
mileage	Interval	0-200,000 (miles on vehicle)

2.1 DELIVERABLES

The project deliverables is to be uploaded to the ecampus website and consists of

1. A report describing the process, analysis, results and conclusions in pdf format.
2. A zipped copy of the SAS EM Project Directory.
3. The python code used to solve the problem organized into a single executable python code file.

3 SAS SOLUTION

The overview, descriptive statistics, solution approach, results and conclusions for the SAS analysis is presented in the subsections below.

3.1 OVERVIEW OF SAS SOLUTION.

The problem involved analyzing the consumer complaints submitted to NTHSA for some Honda makes in 2001-2003, clustering the complaints into 7 topic groups, calculating the sentiment for the complaints and developing the best decision tree involving the newly created topic group and sentiment score as input parameters as well besides the given parameters to predict a crash event. The best decision tree model from SAS was then compared with best decision tree model from python.

3.2 DATA – DESCRIPTIVE STATISTICS

The dataset provided was generally clean with only one outlier for category "mph" and 70 outliers for category "mileage". The outliers were set to missing and the missing data was then imputed using the tree model for both interval and categorical attributes.

Table 3-1 Statistics for missing variables in data file

Variable	Role	Mean	Standard Deviation	Non Missing	Missing	Minimum	Median	Maximum	Skewness	Kurtosis
REP_mileage	INPUT	83926.18	38247.93	5259	71	0	85064	200000	0.180163	0.421665
REP_mph	INPUT	29.29574	17.47841	5329	1	0	30	80	0.251926	0.057142

The data set was then analyzed for data skewness. For the target variable "crash" it was found that the data set has a heavy set of 'N' making the crash 'Y' event an almost rare event. (10.7 %) However, for purposes of this project, random under sampling was not performed based on instructions from the professor. In addition, cross validation for the data set was not performed in SAS as per instructions of the instructor. However, the cross validation was performed in python.

As the data had a low number of 'Y' for crash, and as the problem statement did not clearly define any costs associated with the crash event, the sensitivity, precision and F1 were the metrics chosen to compare models.

Table 3-2 Statistical data for the training variables.

Data Role	Variable Name	Role	Number of Levels	Missing	Mode	Mode Percentage	Mode2	Mode2 Percentage
TRAIN	Make	INPUT	2	0	HONDA	87.17	ACURA	12.83
TRAIN	Model	INPUT	6	0	ACCORD	31.71	CIVIC	29.91
TRAIN	Year	INPUT	3	0	2002	59.17	2001	33.45
TRAIN	abs	INPUT	2	0	N	73.06	Y	26.94
TRAIN	cruise	INPUT	2	0	N	67.88	Y	32.12
TRAIN	crash	TARGET	2	0	N	89.29	Y	10.71

Distribution of Class Target and Segment Variables
(maximum 500 observations printed)

Data Role=TRAIN

Data Role	Variable Name	Role	Level	Frequency Count	Percent
TRAIN	crash	TARGET	N	4759	89.2871
TRAIN	crash	TARGET	Y	571	10.7129

3.3 SOLUTION APPROACH

The solution approach is described in this section. The SAS diagrams, node property and SAS code snapshots are presented in Appendix C and D.

In SAS, the data file was first read with the import node and then variables that are outside the limits shown in the data dictionary were set to missing. The missing values were then imputed using the tree method. Text Parsing was performed with Parts of Speech(POS), stop words and stemming. The text was then filtered using the text filter node using TF-IDF method to develop the term document matrix. The text cluster node was then used to develop the text clusters using the singular value decomposition(SVD). The SVD resolution was set to maximum and the default SVD dimensions of 100 was not changed. The clustering was then saved into another SAS file.

Figure 3-1 SAS Diagram No 1

The SAS file with the cluster data was then read into a new diagram where text parsing was performed without parts of speech(POS), no stemming and with start list (sentiment words). Text filtering was done with no weightings. Sentiment scoring was then performed on this file with the sentiment terms and scores provided by the instructor.

The sentiment scores that were computed for the documents was then merged with the original clustered SAS file (by the document ID). A decision tree analysis was then conducted for the merged data for varying depths, branches, leaf size, categorical sizes to predict the outcome of the crash.

For the decision tree analysis, the dataset was partitioned based on a 70/30 split. The decision tree models were run on the partitioned dataset and the metrics recorded to identify the best performing decision tree. Since random under sampling was not performed on the data set, the data partition was repeated with different random seeds to ensure unbiased metric results.

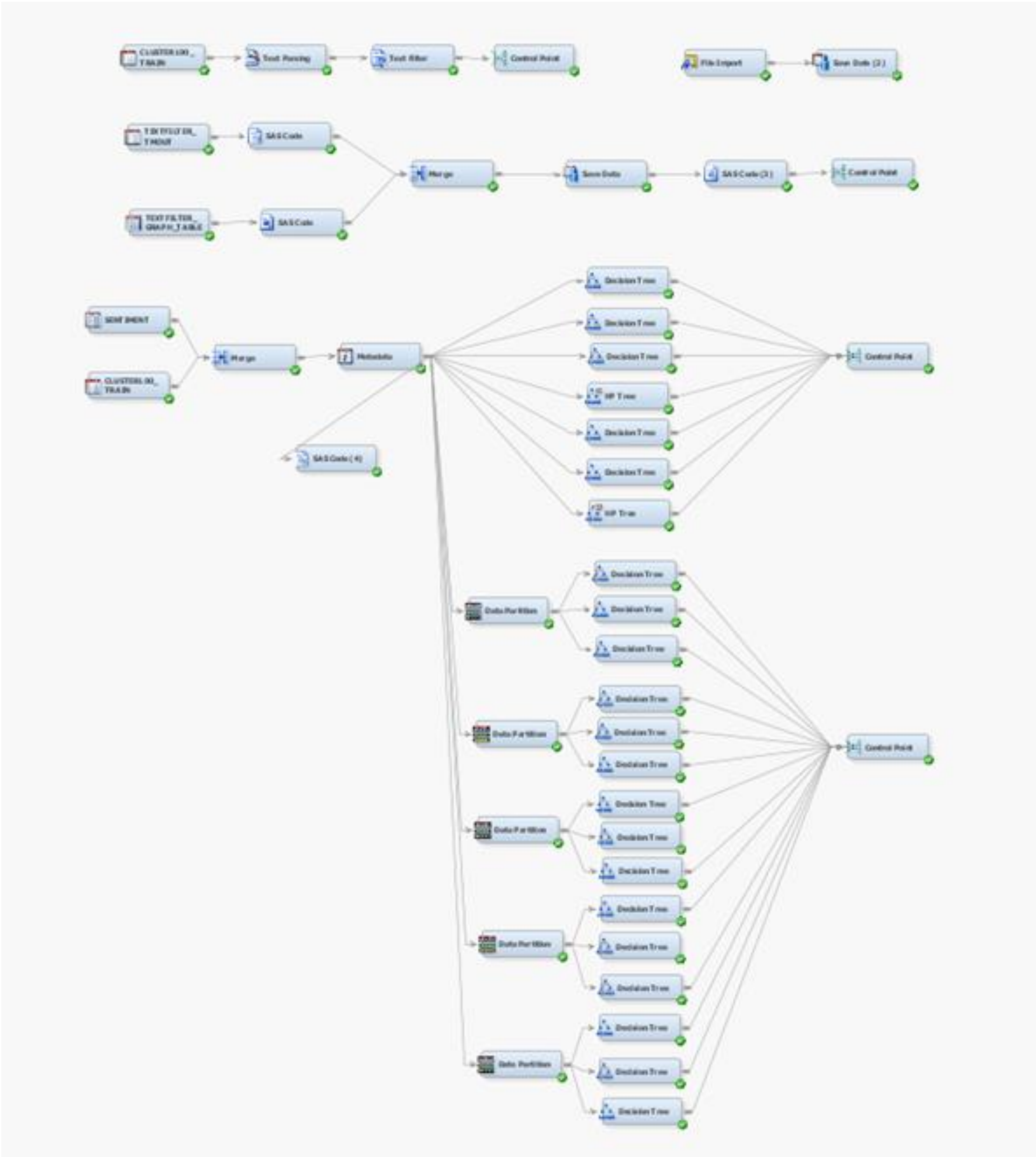


Figure 3-2 SAS Diagram No 2 (Decision tree analysis)

3.4 RESULTS

The text clustering results are shown in Table 3-3 below.

Table 3-3 Descriptive terms and frequencies for the clusters.

Cluster ID	Descriptive Terms	Frequency
1	+tire tread +sidewall flat +rim +stem michelin +blow +purchase rear +damage front +road +month +wheel ...	105
2	+vehicle +dealer +consumer +brake front +noise +steer +wheel +cause +turn +stop driving +control rear +problem ...	697
3	+car +transmission +drive +mile +engine +gear +start +stop +shift +happen check +accelerate +road +slip +problem ...	883
4	+contact +failure +vehicle +mileage honda +repair +'failure mileage' +state +manufacturer +dealer current mph +'current mileage' air +bag ...	820
5	safety +blow +damage +cause +control +notice +find +time rear +turn front +happen +issue +year +month ...	794
6	+transmission +mile +replace +gear +slip +fail acura +odyssey +'transmission failure' +shift +warranty 2nd automatic +cost +recall ...	965
7	+light srs air +bag +airbag +deploy +seat +driver +'srs light' +'air bag' +passenger +belt +accident side safety ...	1066

The average sentiment score for the entire cluster was found to be -1.09 which is expected given that the file is a complaints file. The sentiment scores per model and cluster is presented in Table 3-4.

Table 3-4 Sentiment score per model (above) and cluster (below)

	docScore					
	Min	Mean	Median	Max	N	PctN
Model						
ACCORD	-3.00	-1.08	-1.13	3.00	1690.00	31.71
CIVIC	-3.00	-1.09	-1.17	3.00	1594.00	29.91
CL	-2.75	-0.91	-0.94	0.50	57.00	1.07
CR-V	-3.00	-1.08	-1.20	2.00	464.00	8.71
ODYSSEY	-4.00	-1.10	-1.14	2.00	898.00	16.85
TL	-3.00	-1.16	-1.18	2.00	627.00	11.76

	docScore					
	Min	Mean	Median	Max	N	PctN
TextCluster_cluster_						
1	-3.00	-1.07	-1.07	2.00	105.00	1.97
2	-3.00	-1.15	-1.25	3.00	697.00	13.08
3	-3.00	-0.90	-0.94	3.00	883.00	16.57
4	-2.75	-1.76	-2.00	1.00	820.00	15.38
5	-4.00	-0.87	-0.98	3.00	794.00	14.90
6	-3.00	-1.10	-1.18	2.00	965.00	18.11
7	-3.00	-1.06	-1.09	3.00	1066.00	20.00

Table 3-5 Sentiment score for both model and cluster (part 1)

		docScore					
		Min	Mean	Median	Max	N	PctN
Model	TextCluster_cluster_						
ACCORD	1	-2.50	-0.98	-1.00	1.00	28.00	0.53
	2	-3.00	-1.04	-1.15	3.00	232.00	4.35
	3	-3.00	-0.92	-0.94	3.00	286.00	5.37
	4	-2.50	-1.80	-2.00	0.00	257.00	4.82
	5	-3.00	-0.87	-1.00	3.00	248.00	4.65
	6	-3.00	-1.10	-1.29	2.00	229.00	4.30
	7	-3.00	-1.02	-1.00	3.00	410.00	7.69
CIVIC	1	-2.00	-1.09	-1.39	2.00	31.00	0.58
	2	-3.00	-1.19	-1.25	2.00	217.00	4.07
	3	-3.00	-0.85	-0.92	3.00	354.00	6.64
	4	-2.50	-1.79	-2.00	0.00	232.00	4.35
	5	-3.00	-0.83	-1.00	3.00	238.00	4.47
	6	-3.00	-1.22	-1.33	1.50	120.00	2.25
	7	-3.00	-1.16	-1.20	3.00	402.00	7.54

Table 3-6 Sentiment score for both model and cluster (part 2)

CL	1	-2.75	-1.03	-0.25	0.00	5.00	0.09
	2	-2.00	-0.76	-0.50	0.25	7.00	0.13
	3	-1.82	-0.81	-0.71	0.50	5.00	0.09
	4	-2.33	-1.73	-2.00	0.00	8.00	0.15
	5	-2.00	-0.81	-0.42	-0.33	6.00	0.11
	6	-2.00	-0.79	-0.94	0.29	22.00	0.41
	7	-2.50	-0.89	0.00	0.33	4.00	0.08
CR-V	1	-3.00	-1.27	-1.25	0.00	8.00	0.15
	2	-2.25	-1.10	-1.33	2.00	72.00	1.35
	3	-2.50	-1.01	-0.94	2.00	21.00	0.39
	4	-2.50	-1.79	-2.00	1.00	109.00	2.05
	5	-3.00	-0.85	-0.83	2.00	148.00	2.78
	6	-1.50	-0.59	-0.17	-0.17	3.00	0.06
	7	-3.00	-0.94	-1.09	2.00	103.00	1.93

Table 3-7 Sentiment score for both model and cluster (part 3)

ODYSSEY	1	-2.33	-1.31	-1.25	0.00	15.00	0.28
	2	-3.00	-1.14	-1.17	2.00	97.00	1.82
	3	-3.00	-0.93	-0.95	2.00	139.00	2.61
	4	-2.75	-1.66	-1.70	1.00	115.00	2.16
	5	-4.00	-0.83	-0.87	2.00	93.00	1.74
	6	-3.00	-1.09	-1.00	2.00	329.00	6.17
	7	-3.00	-1.07	-1.07	1.50	110.00	2.06
TL	1	-2.00	-0.94	-1.07	0.00	18.00	0.34
	2	-3.00	-1.53	-1.67	1.00	72.00	1.35
	3	-3.00	-0.98	-1.00	1.00	78.00	1.46
	4	-2.50	-1.70	-1.75	0.00	99.00	1.86
	5	-2.75	-1.08	-1.08	1.50	61.00	1.14
	6	-3.00	-1.08	-1.17	2.00	262.00	4.92
	7	-3.00	-0.79	-0.43	2.00	37.00	0.69

Table 3-8 Decision Tree metrics for various parameters (whole data)

Metrics	Decision Tree Depth 20 Branch 2 Leaf Size 5 Category Size 5 Random Seed 12345	Decision Tree Depth 20 Branch 3 Leaf Size 5 Category Size 5 Random Seed 12345	Decision Tree Depth 20 Branch 2 Leaf Size 3 Category Size 3 Random Seed 12345	Decision Tree Depth 20 Branch 2 Leaf Size 2 Category Size 2 Random Seed 12345	Decision Tree Depth 20 Branch 3 Leaf Size 1 Category Size 1 Random Seed 12345	HP Decision Tree Depth 25 Branch 2 Leaf Size 1 Category Size 1 Random Seed 12345	HP Decision Tree Depth 25 Branch 3 Leaf Size 1 Category Size 1 Random Seed 12345
MISC	0.0402	0.0388	0.0261	0.0176	**0.0000**	0.0604	0.0752
Sensitivity	0.7583	0.7443	0.8722	0.9019	**1.0000**	0.5709	0.4939
Specificity	0.9840	0.9872	0.9861	0.9920	**1.0000**	0.9838	0.9765
FPR	0.0160	0.0128	0.0139	0.0080	**0.0000**	0.0162	0.0235
Precision	0.8507	0.8745	0.8830	0.9313	**1.0000**	0.8089	0.7157
Accuracy	0.9598	0.9612	0.9739	0.9824	**1.0000**	0.9396	0.9248
F1	0.8019	0.8042	0.8775	0.9164	**1.0000**	0.6694	0.5845

Table 3-9 Decision Tree metrics for various seeds.

Data Partition 70/30, DT Depth 20, Branch 2, Leaf Size 2, Category Size 2	**FN**	**TN**	**FP**	**TP**	**METRICS (AVERAGE)**	
Random Seed Value						
1	86	1396	32	86	**MISC**	0.0739
10	94	1411	17	77	**Sensitivity**	0.4609
1000	79	1402	26	93	**Specificity**	0.9819
123	103	1403	25	68	**FPR**	0.0181
12345	99	1400	29	72	**Precision**	0.7538
TOTAL	**461**	**7012**	**129**	**396**	**Accuracy**	0.9261
					F1	0.5720

Table 3-10 Decision Tree metrics for various seeds.

Data Partition 70/30, DT Depth 20, Branch 2, Leaf Size 1, Category Size 1	FN	TN	FP	TP	METRICS	
Random Seed Value						
1	83	1393	35	89	MISC	0.0733
10	91	1410	18	80	Sensitivity	0.4691
1000	77	1401	27	95	Specificity	0.9817
123	107	1409	19	64	FPR	0.0183
12345	97	1397	32	74	Precision	0.7542
TOTAL	455	7010	131	402	Accuracy	0.9267
					F1	0.5784

The decision tree was evaluated for different parameters. On increasing the branches, the metrics were observed to fall. The ideal depth for best metrics is found to be 15-20. The leaf sizes and category sizes were varied until the best results were obtained. It is observed that on reducing the leaf size and category size, the metrics improve. The best performing decision tree was found to have a depth of 20, 2 branches, 1 category size and 1 min leaf size. However as this yielded a perfect result on unpartitioned dataset, there was a chance that the model could be overfitting. Hence on the partitioned data set, decision trees were evaluated for depth of 20, 2 branches, category sizes 1 and 2, leaf sizes 1 and 2.

The best decision tree model is found to be depth of 20, 2 branches, category size 1 and leaf size 1, with the other decision tree not too different.

3.5 OBSERVATIONS AND CONCLUSIONS.

The decision tree did not yield the best results with sensitivities ranging less than 50%. This could be attributed to lack of cross validation and under sampling. Further improvement in the metrics is thus possible if the above techniques are employed.

The importance of clustering of text description is seen below in Table 3-11 with the text cluster probabilities having high importance in determining the final outcome of the crash. The sentiment score however does not have sufficient impact on the decision tree as signified by its low importance score. This is understandable as the complaints file will have almost all negative sentiment scores and the extent of the damage/crash may not always correlate to the sentiment of the comment as it varies depending on the person complaining.

Among the original attributes, MPH has a very high importance in determining the outcome of a crash. This is understandable as well, as higher MPH values are typically expected to result in crashes.

Overall the decision tree results seem reasonable and the decision tree shows the importance of text clustering in determining the outcomes.

Table 3-11 Variable importance in final model

Variable Importance

Variable Name	Label	Number of Splitting Rules	Importance	Validation Importance	Ratio of Validation to Training Importance
TextCluster_prob7		6	1.0000	0.8868	0.8868
TextCluster_prob6		3	0.9438	0.7932	0.8404
IMP_REP_mph	Imputed: Replacement: mph	3	0.9029	1.0000	1.1076
TextCluster_prob4		3	0.6470	0.7317	1.1308
TextCluster_prob1		4	0.6001	0.5886	0.9808
TextCluster_prob3		3	0.4450	0.4583	1.0300
IMP_REP_mileage	Imputed: Replacement: mileage	3	0.3990	0.1099	0.2755
Model	Model	3	0.3841	0.2455	0.6392
ndoc		2	0.3455	0.2563	0.7419
TextCluster_prob2		2	0.2875	0.0554	0.1928
docScore		2	0.2014	0.2200	1.0923
TextCluster_prob5		2	0.1863	0.1106	0.5935

.

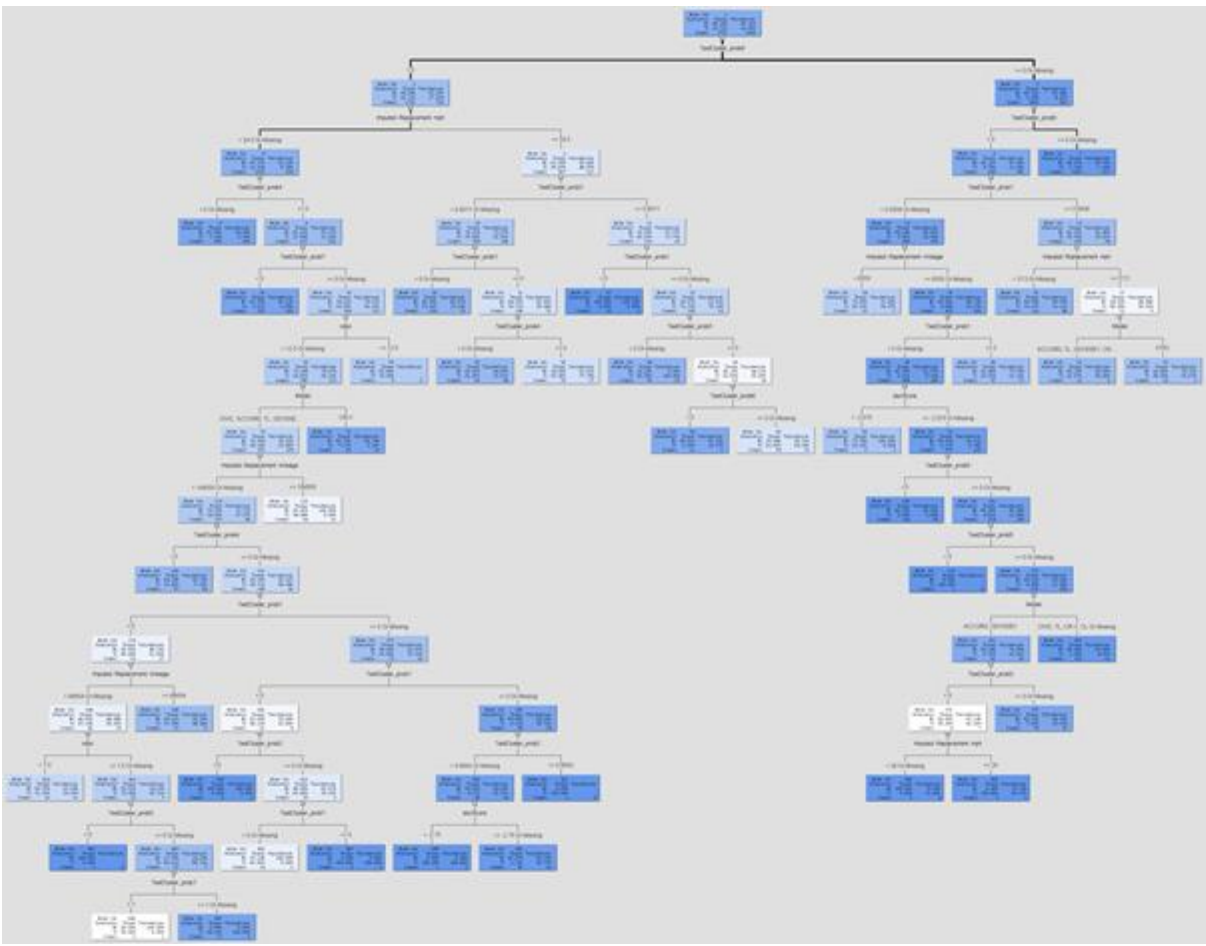


Figure 3-3 Final decision tree model

4 PYTHON SOLUTION

4.1 OVERVIEW OF PYTHON SOLUTION.

The problem statement had two parts.

The business problem was to build and validate the best model for predicting the probability of a crash based upon the topic and sentiment model and upon the other variables available in the given data set. This involved building a topic model, calculating the average sentiment scores for each complaint, merging topic and sentiment scores to the original data set and then building a base decision tree model to predict the crash. Hyper-parameter optimization and 10-fold cross validation is then used to identify the best decision tree model that accurately predicts the crash. A validation set approach was then done to split the entire data set in to train and validation data sets. Finally the best decision Tree model was tested on the validation dataset.

The second part consisted of downloading latest news on the Japanese airbag manufacturer “Takata” using the NewsApiClient package. URLs containing the key topic “Takata” is then searched with the news agency list used by the API news feed package. After that, the content was downloaded from the URLs and cleaned for removing HTML, CSS and JavaScript code to create articles. Finally a summary of how the articles are related to the topic groups generated in the first part was written.

4.2 DATA – DESCRIPTIVE STATISTICS

The HondaComplaint file consist of 5,330 consumer complaints submitted to the NTHSA for some Honda makes in years 2001-2003.

4.2.1 FINDING SYNONYMS

Before doing the preprocessing step, a dictionary of synonyms that need to be replaced by main words was found using the custom function *get_synonyms*. A total of 155 synonyms were found that were then replaced with their main words in the complaints corpus. In the preprocessing step the synonyms dictionary was used to replace synonyms.

4.2.2 FINDING STOP WORDS

Words with highest frequency in a corpus are most of the time in the stop words list. The top 150 words by frequency were found and words such as "us", "v", "ws", "w", "eld", "would", "told", "tc", "sr", "cls", "could", "took", "said", "get", "since", "came", "went", "called", "go", "going", etc. were identified as stop words by skimming through the list of 150 words.

4.2.3 PRE-PROCESSING

Before any real text processing is to be done, text needs to be segmented into linguistic units such as words, punctuations, numbers, alpha-numerics etc. This process is called tokenization. Tokenization was done followed by Parts Of Speech Tagging. POS Tagging is the lowest level of syntactic analysis. It was done to map each of the parts of speech to each token created. After that stop words, which are extremely common words and do not carry any information, were removed. Stemming was used to convert words in to their stems. The commonly used stemming algorithm called Porter Stemmer was used. Finally, a Term/Doc matrix was created with CountVectorizer. The table below shows the terms with highest frequency in the data file.

Table 4-1 Terms with highest frequency

```
Terms with Highest Frequency:
honda           6153
transmission    5305
vehicle         5103
car             4614
problem         2988
contact         2868
dealer          2730
failure         2351
drive           2307
light           2203
```

TF*IDF is an information retrieval technique was used to transform the TF matrix from term counts to term counts weighted by the IDF, Inverse Document Frequency to reduce the weights of common terms used in all documents and increase the weight of terms found in document clusters. The Term/Frequency matrix using TF-IDF technique is presented below.

Table 4-2 Term/frequency matrix

```
Conducting Term/Frequency Matrix using TF-IDF
The Term/Frequency matrix has 5330  rows, and 2865  columns.
The Term list has 2865  terms.

Terms with Highest TF-IDF Scores:
transmission   10209.39
honda           9536.59
car             9524.76
vehicle         9237.59
contact         7036.93
problem         6357.66
dealer          5594.33
failure         5308.85
light           5233.58
drive           4824.68
Number of Reviews.....  5330
Number of Terms.......  2865
```

4.2.4 TOPIC ANALYSIS

Latent Dirichlet Allocation, or LDA was used organize the complaints in to 7 topic groups. A 20 word limit was set for each topic group. A table of topic by complaints information is shown below.

Table 4-3 Topics grouped by complaints

```
Topics Identified using LDA with TF_IDF
Topic #1:
+light         +srs          +problem      +seat         +dealer
+system        +pedal        +airbag       +come         +belt
+sensor        +time         +brake        +unit         +airbags
+safety        +stay         +honda        +control      +replace

Topic #2:
+transmission  +gear         +engine       +shift        +drive
+slips         +car          +mile         +vehicle      +start
+revs          +accelerate   +replace      +check        +problem
+dealer        +noise        +honda        +fail         +automatic

Topic #3:
+honda         +warranty     +tire         +year         +recall
+transmission  +replace      +car          +mile         +service
+part          +problem      +issue        +pay          +cover
+cost          +purchase     +know         +dealerships  +new

Topic #4:
+honda         +car          +problem      +issue        +recall
+complaint     +civic        +mile         +fix          +many
+nothing       +find         +safety       +read         +people
+nearly        +like         +model        +odyssey      +something

Topic #5:
+side          +air          +driver       +bag          +front
+vehicle       +passenger    +deploy       +door         +consumer
+seat          +injury       +rear         +airbag       +crash
+head          +damage       +wheel        +cause        +hit

Topic #6:
+contact       +vehicle      +failure      +mileage      +state
+repair        +own          +manufacturer +recall       +dealer
+current       +nhtsa        +number       +honda        +campaign
+take          +headlight    +bag          +air          +beams

Topic #7:
+car           +acura        +brake        +stop         +tl
+lane          +home         +work         +hit          +road
+get           +traffic      +hour         +minute       +back
+child         +way          +van          +drive        +nearly
```

Table 4-4 Topic by complaint count

```
***Topic by Complaints count***
Out[508]:
topic
0       700
1      1588
2       220
3       366
4       720
5      1495
6       241
```

4.2.5 SENTIMENT ANALYSIS

The average sentiment score for the whole corpus was found to be -1.08, which makes sense since this is a complaint data file. On average, we can confirm that the complaints registered with the NTSHA have negative emotional content. The average sentiment per topic, per make, per model, and per all three was calculated and it was found. The 6 most negative reviews and single most positive review with 4 or more sentiment words scoring -2.75 and 1.89 respectively are shown below.

Table 4-5 Results of sentiment analysis

```
**** Sentiment Analysis ****
Number of Reviews.....  5330
Number of Terms.......132050

Corpus Average Sentiment:  -1.0845359648207924

Most Negative Reviews with 4 or more Sentiment Words:
   Review 878 sentiment is -2.75
   Review 1231 sentiment is -2.75
   Review 2065 sentiment is -2.75
   Review 2778 sentiment is -2.75
   Review 3901 sentiment is -2.75
   Review 4360 sentiment is -2.75

Most Positive Reviews with 4 or more Sentiment Words:
   Review 4588 sentiment is 1.89
```

Table 4-6 Sentiment by topic

```
topic
0   -0.935191
1   -1.014683
2   -0.694637
3   -0.786016
4   -1.066584
5   -1.427362
6   -0.714851
Name: sentiment, dtype: float64
```

Table 4-7 Sentiment by make

```
Make
ACURA   -1.110596
HONDA   -1.080699
Name: sentiment, dtype: float64
```

Table 4-8 Sentiment by model

Model	
ACCORD	-1.042504
CIVIC	-1.118006
CL	-0.939664
CR-V	-1.104395
ODYSSEY	-1.074117
TL	-1.126136

Name: sentiment, dtype: float64

Table 4-9 Sentiment by make, model and topic

Make	topic	Model	
ACURA	0	CL	-0.571429
		TL	-1.172619
	1	CL	-0.753611
		TL	-1.034775
	2	CL	-0.614583
		TL	-0.692975
	3	TL	-1.034776
	4	CL	-0.805556
		TL	-1.091375
	5	CL	-1.528846
		TL	-1.471401
	6	CL	-0.872808
		TL	-0.738665
HONDA	0	ACCORD	-0.883166
		CIVIC	-0.953372
		CR-V	-0.950720
		ODYSSEY	-1.056847
	1	ACCORD	-0.933799
		CIVIC	-1.082320
		CR-V	-1.179388
		ODYSSEY	-1.026601
	2	ACCORD	-0.649903
		CIVIC	-0.786863
		CR-V	-0.653385
		ODYSSEY	-0.697228
	3	ACCORD	-0.662035
		CIVIC	-0.849663
		CR-V	-0.802013
		ODYSSEY	-0.846058
	4	ACCORD	-1.063352
		CIVIC	-1.160605
		CR-V	-0.979314
		ODYSSEY	-0.907704
	5	ACCORD	-1.502385
		CIVIC	-1.387259
		CR-V	-1.319006
		ODYSSEY	-1.430013
	6	ACCORD	-0.670356
		CIVIC	-0.802946
		CR-V	-0.752273
		ODYSSEY	-0.433800

Name: sentiment, dtype: float64

4.2.6 MISSING VALUES AND OUTLIERS

One outlier was found for mph variable. 70 outliers and one missing value were found for mileage variable. Imputing the missing values was done using ReplaceImputeEncode function.

Table 4-10 Sample Table (copy, paste and update field)

```
Attribute Counts
................ Missing  Outliers
NhtsaID....         0         0
Make.......         0         0
Model......         0         0
Year.......         0         0
State......         0         0
abs........         0         0
cruise.....         0         0
crash......         0         0
mph........         0         1
mileage....         1        70
```

4.2.7 WEB SCRAPING

The following dictionary containing the URLs was used for different agencies used by the API news feed package. By default it will only download a maximum of 20 articles from any single request. The search key word in this case is 'Takata'. A total of 63 URLs were found, of which 60 were unique.

Table 4-11 Data dictionary for urls

```
Searching agencies for pages containing: ['Takata']
huffington huffingtonpost.com
reuters www.reuters.com
cbs-news www.cbsnews.com
usa-today usatoday.com
cnn cnn.com
npr www.npr.org
wsj wsj.com
fox www.foxnews.com
abc abc.com
abc-news abcnews.com
abcgonews abcnews.go.com
nyt nytimes.com
washington-post washingtonpost.com
us-news www.usnews.com
msn msn.com
pbs www.pbs.org
nbc-news www.nbcnews.com
enquirer www.nationalenquirer.com
la-times www.latimes.com

Found a total of 63  URLs, of which 60  were unique.
Total Articles: 60
Agency: reuters
Search Word: Takata
URL: https://www.reuters.com/article/us-autos-takata/honda-ford-to-testify-at-u-s-senate-
takata-hearing-aides-idUSKCN1GP30F
```

The content of a total of 60 web pages were downloaded and cleaned for removing CSS, Javascript and HTML code. However, five 404 status codes were also received during this procedure. The list of the webpages and the character count are shown in Appendix B.

4.3 SOLUTION APPROACH

4.3.1 CRASH PREDICTION

Topic analysis is concerned with identifying topics shared among similar documents or reviews. It is performed with the terms found in the corpus, apart from terms found in the stop list. On the other hand, sentiment analysis is about measuring the emotional sentiment in documents so it's only concerned with terms carrying emotional content, either positive or negative.

In topic analysis, the NLTK package was used to customize the function for tokenization, handling synonyms, POS tagging, stop word removal & Stemming. Then a Term/Doc matrix was created with CountVectorizer. This sklearn method returns the matrix where the rows are the documents and the columns are the terms. Next, the TF matrix is transformed from term counts to term counts weighted by the IDF, Inverse Document Frequency. (IDF(i) = log(d/d(i)) where i is the ith term and d is the total number of documents and d(i) is the number for the ith term). Python has several ways to create term groups, or clusters. As similar results were obtained with SVD when compared to LDA, this report highlights topics generated using Latent Dirichlet Allocation.

In sentiment analysis, there's no need to do POS Tagging, Stop Removal & Stemming since we need all terms to calculate sentiment scores. A sentiment dictionary is needed where the keys are the sentiment words and the values are the associated sentiment weight to identify and score the TF matrix to develop an average score that reflects the emotional content of a document.

In the confusion matrix found using the decision tree model the false negatives should be given importance over false positives because, misclassification of no crashes can be accepted but misclassification of crashes cannot be accepted. Hence sensitivity/recall should have high importance in comparing various models over precision. The best model can also be selected based on a combined metric of precision and recall i.e. F1 score. The precision metric answers the following question: out of all the examples the classifier labeled as Crash, what fraction were correct? On the other hand, the recall answers: out of all the Crash examples, what fraction did the classifier pick up? The best model will have relatively a high f1 score and high recall score.

Finally, in predicting the probability of a crash with the data file containing topic and sentiment information, hyperparameter optimization along with 10-fold cross validation was used to configure the best decision tree model. Hyperparameters such as max_depth, min_samples_leaf, & min_samples_split was used in the analysis. Finally, the best model is validated with a 70/30 split.

4.3.2 BEST MODEL

The best model presented below has the highest recall and f1 scores among all the models found using 10 fold cross validation and hyperparameter optimization.

Table 4-12 Metric and parameters for best decision tree model

```
Maximum Tree Depth:  15 Min_samples_leaf 5 Min_samples_split 3
Metric.......  Mean    Std. Dev.
accuracy..... 0.9235    0.0082
recall....... 0.5392    0.1321
precision.... 0.6850    0.0487
f1........... 0.5929    0.0830
```

Table 4-13 Validation od best model

```
Model Metrics..........        Training     Validation
Observations...........            3731           1599
Features...............              22             22
Maximum Tree Depth.....              15             15
Minimum Leaf Size......               5              5
Minimum split Size.....               3              3
Mean Absolute Error....          0.0605         0.0991
Avg Squared Error......          0.0302         0.0676
Accuracy...............          0.9550         0.9199
Precision..............          0.8442         0.6901
Recall (Sensitivity)...          0.6967         0.5385
F1-score...............          0.7634         0.6049
MISC (Misclassification)...        4.5%           8.0%
     class 0...............        1.5%           3.1%
     class 1...............       30.3%          46.2%

Training
Confusion Matrix  Class 0   Class 1
Class 0.....      3292       50
Class 1.....       118      271

Validation
Confusion Matrix  Class 0   Class 1
Class 0.....      1373       44
Class 1.....        84       98
```

Table 4-14 Features ordered by importance

```
FEATURE.... IMPORTANCE
mph........   0.3649
topic4.....   0.2980
mileage....   0.0726
sentiment..   0.0710
Model0.....   0.0621
Model1.....   0.0286
Model4.....   0.0187
Model3.....   0.0168
abs........   0.0167
topic5.....   0.0151
Year0......   0.0095
topic6.....   0.0092
Year1......   0.0087
cruise.....   0.0048
Model5.....   0.0012
topic1.....   0.0006
Year2......   0.0005
topic2.....   0.0004
topic0.....   0.0003
topic3.....   0.0002
Make.......   0.0000
Model2.....   0.0000
```

Table 4-15 Metrics for training and validation for best model

```
***** Train set ******
sensitivity/recall/TPR: 0.6966580976863753
specificity:  0.9850388988629563
accuracy:  0.9549718574108818
precision:  0.8442367601246106
f1_score:  0.7633802816901407
misc:  0.0450281425891182
FPR:  0.014961101137043742


***** Validation set ******
sensitivity/recall/TPR: 0.5384615384615384
specificity:  0.9689484827099506
accuracy:  0.9199499687304565
precision:  0.6901408450704225
f1_score:  0.6049382716049383
misc:  0.08005003126954346
FPR:  0.031051517290049402
```

4.3.3 WEB SCRAPING

Three new packages to do a successful Web Scraping: newspaper, newsapi, and requests. All packages were installed using pip install command. An API key was downloaded from the website https://newsapi.org/docs/get-started to be used while using the function in newsapi package. A function called clean_html was used to clean python string containing raw html and javascript code. The returned file is a string with the html markups removed. The function newsapi*_get_urls* was used to return a dataframe of URLs pointing to news articles drawn from the web. The function *request_pages* was used to return web pages in text format from a list of URLs obtained using *newsapi_get_urls*. No of characters present in each URL are then displayed in Appendix B.

4.4 RESULTS

Hyper-parameter optimization and 10-fold cross validation results are as shown below.

Table 4-16 Metrics for decision trees with various parameters

TITLE: Hyperparameter optimization in decision tree model									
Metrics	**Decision Tree** Depth 5 Leaf Size 3	**Decision Tree** Depth 5 Leaf Size 5	**Decision Tree** Depth 5 Leaf Size 7	**Decision Tree** Depth 6 Leaf Size 3	**Decision Tree** Depth 6 Leaf Size 5	**Decision Tree** Depth 6 Leaf Size 7	**Decision Tree** Depth 8 Leaf Size 3	**Decision Tree** Depth 8 Leaf Size 5	**Decision Tree** Depth 8 Leaf Size 7
Recall	0.3746	0.3764	0.3676	0.4290	0.4343	0.4376	0.5058	0.5076	0.4935
Precision	0.6972	0.6951	0.6951	0.7516	0.7616	0.7553	0.7220	0.7158	0.7135
Accuracy	0.9128	0.9126	0.9124	0.9220	0.9236	0.9229	0.9236	0.9238	0.9212
F1	0.4661	0.4654	0.4608	0.5331	0.5423	0.5397	0.5775	0.5802	0.5619
Metrics	**Decision Tree** Depth 10 Leaf Size 3	**Decision Tree** Depth 10 Leaf Size 5	**Decision Tree** Depth 10 Leaf Size 7	**Decision Tree** Depth 12 Leaf Size 3	**Decision Tree** Depth 12 Leaf Size 5	**Decision Tree** Depth 12 Leaf Size 7	**Decision Tree** Depth 15 Leaf Size 3	**Decision Tree** Depth 15 Leaf Size 5	**Decision Tree** Depth 15 Leaf Size 7
Recall	0.5338	0.5251	0.5111	0.5269	0.5286	0.5111	0.5303	0.5392	0.5111
Precision	0.6638	0.6721	0.6847	0.6305	0.6851	0.6919	0.6183	0.6850	0.6942
Accuracy	0.9169	0.9201	0.9193	0.9141	0.9227	0.9212	0.9124	0.9235	0.9214
F1	0.5737	0.5767	0.5662	0.5625	0.5850	0.5709	0.5589	0.5929	0.5715
Metrics	**Decision Tree** Depth 20 Leaf Size 3	**Decision Tree** Depth 20 Leaf Size 5	**Decision Tree** Depth 20 Leaf Size 7	**Decision Tree** Depth 25 Leaf Size 3	**Decision Tree** Depth 25 Leaf Size 5	**Decision Tree** Depth 25 Leaf Size 7	**Decision Tree** Depth 50 Leaf Size 3	**Decision Tree** Depth 50 Leaf Size 5	**Decision Tree** Depth 50 Leaf Size 7
Recall	0.5285	0.5392	0.5111	0.5285	0.5392	0.5111	0.5285	0.5392	0.5111
Precision	0.6083	0.6825	0.6942	0.6083	0.6825	0.6942	0.6083	0.6825	0.6942
Accuracy	0.9113	0.9231	0.9214	0.9113	0.9231	0.9214	0.9113	0.9231	0.9214
F1	0.5537	0.5917	0.5715	0.5537	0.5917	0.5715	0.5537	0.5917	0.5715

The best model was tested with random seeds for checking its stability. The results of the test is presented in the table below. The ranges are within acceptable limits for prediction. The process for this analysis was simple. The seed was picked randomly and the prediction code was run for each seed value.

Table 4-17 Metrics for best decision trees with varying random seeds

Metrics	TITLE: 70/30 validation of the best decision tree model with different random seeds				**Range**
	Decision Tree Depth 15 min_leaf 5 min_split 3 **Random Seed 12345**	**Decision Tree** Depth 15 min_leaf 5 min_split 3 **Random Seed 123**	**Decision Tree** Depth 15 min_leaf 5 min_split 3 **Random Seed 5**	**Decision Tree** Depth 15 min_leaf 5 min_split 3 **Random Seed 6000**	
MISC	0.0800	0.0787	0.0875	0.0769	0.0769-0.0875
Sensitivity	0.5384	0.6206	0.5519	0.5059	0.5059-0.6206
Precision	0.6901	0.6428	0.6352	0.6800	0.6352-0.6901
Accuracy	0.9199	0.9212	0.9124	0.9230	0.9124-0.9230
F1	0.6040	0.6315	0.5906	0.5802	0.5802-0.6315

4.5 OBSERVATIONS AND CONCLUSIONS.

4.5.1 CRASH PREDICTION

Hyper-parameter max_depth has the highest influence on the metrics score compared to min_sample_leaf and min_samples_split. When the max_depth > 10 the sensitivity/recall score is relatively high compared to other conditions. In addition, the sensitivity/recall score first increases and then decreases when min_sample_leaf value is increased. The precision score just increases when min_sample_leaf value is increased. This indicates that leaf sizes in between 3 and 7 will contain the highest recall and f1 score values. Accuracy is consistently high across all the conditions. Hyper parameters of maximum tree pepth = 15, Min_samples_leaf = 5, and Min_samples_split = 3 were used in the 70/30 validation at the end as they bring the best results when 10-fold cross validation is done. The best decision tree model (with recall = 0.5384 and f1-score = 0.6040) clearly estimates the crashes better than a base model with recall = 0.5 and f1-score = 0.5. Hence it is considered reliable for crash prediction.

4.5.2 WEB SCRAPING

The downloaded content from the URLs related to www.reuters.com mostly talks about issues and safety of Takata airbags and why they were recalled. Hence this content is related to Topic group 4 which has words such as recall, safety, and issue. The content from URLs related to www.usatoday.com highlight about possible injuries consumers faces when using a car with Takata airbags. It also mentions the cause of the injuries and cautions the consumers to stay alert. This content is directly related to Topic 5 which has words such as injury, cause, bag, passenger, driver, etc. Even downloaded content from URLs related to www.money.cnn.com talks about injury and hence is related to Topic 5. Two URLs related to www.abcnews.com and www.washingtonpost.com has content about how Honda profited from its new model called Odyssey. The content in URLs belong to Topic group 4 that has words such as Honda and Odyssey. The single URL related to www.nbc.news.com highlights how Hyundai is under scrutiny for Takata airbag failures after 4 deaths. The content in this link is related to Topics 4 and 6.

Five of the www.washingtonpost.com URLs returned 404 not found error when it was tried to download their content. Most of the ww.abcnews.com URLs had no content left in them after cleaning HTML, CSS, and Javascript code.

5 COMPARISON BETWEEN PYTHON AND SAS RESULTS

While the text clustering procedure is not entirely different between python and SAS, there is a difference in the clustering mechanisms due to probably slightly different approaches between SAS (SVD) and Python (LDA). SVD uses an approach based on finding best reduced matrix that keeps the most information in the original term/frequency matrix. It is basically a deterministic approach using techniques in linear algebra of projecting the matrix into a subspace while maximizing the norm. LDA on the other hand uses probabilistic methods involving conditional probability to connect the topics to the documents and word lists based on the values in the term/document matrix.

Table 5-1 Best SAS Model Metrics

Data Partition 70/30, DT Depth 20, Branch 2, Leaf Size 2, Category Size 2	FN	TN	FP	TP	METRICS (AVERAGE)	
Random Seed Value						
1	86	1396	32	86	**MISC**	0.0739
10	94	1411	17	77	**Sensitivity**	0.4609
1000	79	1402	26	93	**Specificity**	0.9819
123	103	1403	25	68	**FPR**	0.0181
12345	99	1400	29	72	**Precision**	0.7538
TOTAL	**461**	**7012**	**129**	**396**	**Accuracy**	0.9261
					F1	0.5720

Table 5-2 Best Python Model Metrics

TITLE: 70/30 validation of the best decision tree model with different random seeds					
Metrics	**Decision Tree** Depth 15 min_leaf 5 min_split 3 **Random Seed 12345**	**Decision Tree** Depth 15 min_leaf 5 min_split 3 **Random Seed 123**	**Decision Tree** Depth 15 min_leaf 5 min_split 3 **Random Seed 5**	**Decision Tree** Depth 15 min_leaf 5 min_split 3 **Random Seed 6000**	**Range**
MISC	0.0800	0.0787	0.0875	0.0769	0.0769-0.0875
Sensitivity	0.5384	0.6206	0.5519	0.5059	0.5059-0.6206
Precision	0.6901	0.6428	0.6352	0.6800	0.6352-0.6901
Accuracy	0.9199	0.9212	0.9124	0.9230	0.9124-0.9230
F1	0.6040	0.6315	0.5906	0.5802	0.5802-0.6315

Python yielded a decision tree with better sensitivity and F1 score, whereas SAS yielded a model with better precision and slightly better overall misclassification rate. The same model that yielded the best python result was input in SAS and the results are below. The difference in results could be attributed to the difference in clustering between SAS and python.

Table 5-3 SAS vs Python for best Python model

PYTHON - SAS COMPARISON 70/30 validation with different random seeds					
PYTHON Metrics	Decision Tree Depth 15 min_leaf 5 min_split 3 Random Seed 12345	Decision Tree Depth 15 min_leaf 5 min_split 3 Random Seed 123	Decision Tree Depth 15 min_leaf 5 min_split 3 Random Seed 5	Decision Tree Depth 15 min_leaf 5 min_split 3 Random Seed 6000	Range
MISC	0.0800	0.0787	0.0875	0.0769	0.0769-0.0875
Sensitivity	0.5384	0.6206	0.5519	0.5059	0.5059-0.6206
Precision	0.6901	0.6428	0.6352	0.6800	0.6352-0.6901
Accuracy	0.9199	0.9212	0.9124	0.9230	0.9124-0.9230
F1	0.6040	0.6315	0.5906	0.5802	0.5802-0.6315
SAS Results	Random Seed 10	Random Seed 12345	Random Seed 123	Random Seed 1	Range
MISC	0.0682	0.0800	0.0857	0.0688	0.0682-0.0857
Sensitivity	0.4444	0.4152	0.3860	0.5465	0.3860-0.5465
Precision	0.8444	0.7172	0.6735	0.7460	0.6735-0.8444
Accuracy	0.9318	0.9200	0.9143	0.9313	0.9143-0.9318
F1	0.5824	0.5259	0.4907	0.6309	0.4907-0.6309

The sentiment scores for the entire corpus is almost the same in python and SAS which is as expected as the sentiment score file and the data file are the same for python and SAS.

Overall the models from both python and SAS are deemed to be robust.

APPENDIX A: PYTHON CODE

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 28 23:05:53 2018

@author: sasha
"""


# coding: utf-8

# ==============================================================================
# PART 1
# Crash Prediction
# ==============================================================================

import pandas as pd
import numpy as np
import string

# Text topic imports
from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet as wn
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.decomposition import LatentDirichletAllocation
# class for decision tree
from Class_tree import DecisionTree
from sklearn.tree import DecisionTreeClassifier
from Class_replace_impute_encode import ReplaceImputeEncode
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from collections import defaultdict

# ==============================================================================
# Get statistics from confusion matrix
# ==============================================================================
def getClassificationMetrics(tn, fp, fn, tp):

    #sensitivity
    Recall=tp/(tp+fn);
    print("sensitivity/recall/TPR:", Recall)
    #specificity
    Specificity= tn/(tn+fp)
    print("specificity: ", Specificity)
    #accuracy
    print("accuracy: ", (tp+tn)/(tp+fn+tn+fp))
    #precision
    Precision=tp/(tp+fp);
    print("precision: ", Precision)
    #f1 score
    print("f1_score: ",  (2*Recall*Precision)/(Recall + Precision))
    #misclassification
```

```
    print("misc: ", (fp+fn)/(tp+fn+tn+fp))
    #False Positive Rate
    print("FPR: ", 1-Specificity)

    return


# ==============================================================================
# This pre processing is used for finding synonyms
# ==============================================================================
def DoPreProcessing(s):

    # Replace special characters with spaces
    s = s.replace('-', ' ')
    s = s.replace('_', ' ')
    s = s.replace(',', '. ')
    # Replace not contraction with not
    s = s.replace("'nt", " not")
    s = s.replace("n't", " not")
    # Tokenize
    tokens = word_tokenize(s)
    #tokens = [word.replace(',','') for word in tokens ]
    tokens = [word for word in tokens if ('*' not in word) and (" '' "!= word) and ("``"!= word) and              (word!='description') and (word !='dtype') and (word != 'object') and (word!="'s")]


    # Remove stop words
    punctuation = list(string.punctuation)+['..', '...']
    pronouns = ['i', 'he', 'she', 'it', 'him', 'they', 'we', 'us', 'them']
    #Top frequency words ar usually stop words. Top 150 words by frequency are
    #listed and then manually below words were added as stop words
    others   = ["'m","us","v","ws","w","eld","would", "told","tc","sr",
                "cls","could", "took", "said", "get", "since",
                  "came", "went", "called", "go", "going","'d", "co","gm",
                  "ed", "put", "say", "get", "can", "become",
                "los", "sta", "la", "use", "iii", "else", "could", "also",
                "even", "really", "one", "would", "get", "getting", "go", "going",
                "place", "want", "get","take", "end","next", "though","non", "seem"
               ]

    stop = stopwords.words('english') + punctuation + pronouns + others
    filtered_terms = [word for word in tokens if (word not in stop) and (len(word)>1) and (not word.replace('.','',1).isnumeric()) and (not word.replace(" ' " , '',2).isnumeric())]

        # Lemmatization & Stemming - Stemming with WordNet POS
        # Since lemmatization requires POS need to set POS
        tagged_words = pos_tag(filtered_terms, lang='eng')
        # Stemming with for terms without WordNet POS
        stemmer = SnowballStemmer("english")
        wn_tags = {'N':wn.NOUN, 'J':wn.ADJ, 'V':wn.VERB, 'R':wn.ADV}
        wnl = WordNetLemmatizer()
        stemmed_tokens = []
        for tagged_token in tagged_words:
            term = tagged_token[0]
            pos  = tagged_token[1]
            pos  = pos[0]
            try:
                pos   = wn_tags[pos]
                stemmed_tokens.append(wnl.lemmatize(term, pos=pos))
```

```
            except:
                stemmed_tokens.append(stemmer.stem(term))
        #print(stemmed_tokens)
        return stemmed_tokens


    # =============================================================================
    # Get synonyms that need to be replaced
    # =============================================================================
    def get_synonyms(totalList):

        #this dictionary contains words and their synonyms
        #Some of the main words are not in the corpus, but their synonyms are
        d = defaultdict(list)

        for item in totalList:
            syn = wn.synsets(item)
            if len(syn)>0:
                        if syn[0].lemma_names()[0]!=item:
                                d[syn[0].lemma_names()[0]].append(item)

        len(d)

        #This list contains main words and their synonyms
        # if no main word is present its first synonym becomes the main words
        #and the subsequent synonyms become its synonyms
        synonyms = defaultdict(str)

        for item in d:
            if totalList.count(item)==0:
                if len(d[item])>1:
                    # the flag is there to make the first synonym in the list
                    # as main word
                    flag =0
                    for a in d[item]:
                        if flag==0:
                            flag = flag+1
                        else:
                            synonyms[a] = d[item][0]
        return synonyms

    # =============================================================================
    # Used for finding the Term/Document matrix
    # =============================================================================
    def my_analyzer(s):
        # Synonym List
    # =============================================================================
    #     syns = { "n't":'not',  'wont':'would not', 'cant':'can not', 'cannot':'can not',
    #               'couldnt':'could not', 'shouldnt':'should not',
    #               'wouldnt':'would not',   }
    # =============================================================================

        syns = synonymsDict
        # Preprocess String s
```

```
            s = s.lower()
            # Replace special characters with spaces
            s = s.replace('-', ' ')
            s = s.replace('_', ' ')
            s = s.replace(',', '. ')
            # Replace not contraction with not
            s = s.replace("'nt", " not")
            s = s.replace("n't", " not")
            # Tokenize
            tokens = word_tokenize(s)
            #tokens = [word.replace(',','') for word in tokens ]
            tokens = [word for word in tokens if ('*' not in word) and (" '' "!= word) and ("``" != word) and               (word!='description') and (word !='dtype') and (word != 'object') and (word!="'s")]


            # Remove stop words
            punctuation = list(string.punctuation)+['..', '...']
            pronouns = ['i', 'he', 'she', 'it', 'him', 'they', 'we', 'us', 'them']
            #Top frequency words ar usually stop words. Top 150 words by frequency are
            #listed and then manually below words were added as stop words
            others   = ["us","v","ws","w","eld","would", "told","tc","sr",
                        "cls","could", "took", "said", "get", "since",
                          "came", "went", "called", "go", "going","'d", "co","gm",
                          "ed", "put", "say", "get", "can", "become",
                        "los", "sta", "la", "use", "iii", "else", "could", "also",
                        "even", "really", "one", "would", "get", "getting", "go", "going",
                        "place", "want", "get","take", "end","next", "though","non", "seem"
                       ]

            stop = stopwords.words('english') + punctuation + pronouns + others
            filtered_terms = [word for word in tokens if (word not in stop) and (len(word)>1) and (not word.replace('.','',1).isnumeric()) and (not word.replace(" ' ", '',2).isnumeric())]

            # Lemmatization & Stemming - Stemming with WordNet POS
            # Since lemmatization requires POS need to set POS
            tagged_words = pos_tag(filtered_terms, lang='eng')
            # Stemming with for terms without WordNet POS
            stemmer = SnowballStemmer("english")
            wn_tags = {'N':wn.NOUN, 'J':wn.ADJ, 'V':wn.VERB, 'R':wn.ADV}
            wnl = WordNetLemmatizer()
            stemmed_tokens = []
            for tagged_token in tagged_words:
                term = tagged_token[0]
                pos  = tagged_token[1]
                pos  = pos[0]
                try:
                    pos   = wn_tags[pos]
                    stemmed_tokens.append(wnl.lemmatize(term, pos=pos))
                except:
                    stemmed_tokens.append(stemmer.stem(term))


            for i in range(len(stemmed_tokens)):
                if stemmed_tokens[i] in syns:
                    stemmed_tokens[i] = syns[stemmed_tokens[i]]
            #print(stemmed_tokens)
```

```

    return stemmed_tokens


# ==============================================================================
# Used for sentiment analysis
# ==============================================================================
def my_preprocessor(s):
   # Preprocess String s
    s = s.lower()
   # Replace special characters with spaces
    s = s.replace('-', ' ')
    s = s.replace('_', ' ')
    s = s.replace(',', '. ')
   # Replace not contraction with not
    s = s.replace("'nt", " not")
    s = s.replace("n't", " not")
    return s


def display_topics(lda, terms, n_terms=15):
    for topic_idx, topic in enumerate(lda):
        message  = "Topic #%d: " %(topic_idx+1)
        print(message)
        abs_topic = abs(topic)
        topic_terms_sorted =                 [[terms[i], topic[i]]
 for i in abs_topic.argsort()[:-n_terms - 1:-1]]
        k = 5
        n = int(n_terms/k)
        m = n_terms - k*n
        for j in range(n):
            l = k*j
            message = ''
            for i in range(k):
                if topic_terms_sorted[i+l][1]>0:
                    word = "+"+topic_terms_sorted[i+l][0]
                else:
                    word = "-"+topic_terms_sorted[i+l][0]
                message += '{:<15s}'.format(word)
            print(message)
        if m> 0:
            l = k*n
            message = ''
            for i in range(m):
                if topic_terms_sorted[i+l][1]>0:
                    word = "+"+topic_terms_sorted[i+l][0]
                else:
                    word = "-"+topic_terms_sorted[i+l][0]
                message += '{:<15s}'.format(word)
            print(message)
        print("")
    return


#Set Seed
seed = 12345

# topic analysis
```

```
pd.set_option("max_colwidth", 32000)
file_path = "/Users/sasha/Library/Mobile Documents/com~apple~CloudDocs/STAT 656/Final Exam/"
df = pd.read_excel(file_path + "HondaComplaints.xlsx")

df["description"] = df["description"].str.lower()


#Create complaints corpus
description=""
for item in df["description"]:
      description = description+item


#To find stop words in first 150 words with highest frequency
stop = stopwords.words('english')
example = df['description'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
example.head()

series=pd.Series(' '.join(example).split())
pd.Series(' '.join(example).split()).value_counts()[:50]
pd.Series(' '.join(example).split()).value_counts()[50:100]
pd.Series(' '.join(example).split()).value_counts()[100:150]

#Get unique words
StopWordsSeries= set(series)
StopWordslist = list(StopWordsSeries)

#create synonyms dictionary
totallist= DoPreProcessing(description)
# used to remove duplicate items
totalSet= set(totallist)
totallist = list(totalSet)

synonymsDict=get_synonyms(totallist)



# Setup program constants
n_comments  = len(df['description']) # Number of wine reviews
m_features = None                    # Number of SVD Vectors
s_words    = 'english'                # Stop Word Dictionary
comments = df['description']          # place all text reviews in reviews
n_topics =  7                         # number of topic clusters to extract
max_iter = 10                         # maximum number of itertions
learning_offset = 10.                  # learning offset for LDA
learning_method = 'online'             # learning method for LDA



# Create Word Frequency by Review Matrix using Custom Analyzer
cv = CountVectorizer(max_df=0.7, min_df=4, max_features=m_features,analyzer=my_analyzer, ngram_range=(1,2))
tf    = cv.fit_transform(comments)
terms = cv.get_feature_names()
term_sums = tf.sum(axis=0)
term_counts = []
for i in range(len(terms)):
    term_counts.append([terms[i], term_sums[0,i]])
def sortSecond(e):
```

```
            return e[1]
        term_counts.sort(key=sortSecond, reverse=True)
        print("\nTerms with Highest Frequency:")
        for i in range(10):
            print('{:<15s}{:>5d}'.format(term_counts[i][0], term_counts[i][1]))
        print("")




        # Modify tf, term frequencies, to TF/IDF matrix from the data
        print("Conducting Term/Frequency Matrix using TF-IDF")
        tfidf_vect = TfidfTransformer(norm=None, use_idf=True) #set norm=None
        tf          = tfidf_vect.fit_transform(tf)

        term_idf_sums = tf.sum(axis=0)
        term_idf_scores = []
        for i in range(len(terms)):
            term_idf_scores.append([terms[i], term_idf_sums[0,i]])
        print("The Term/Frequency matrix has", tf.shape[0], " rows, and", tf.shape[1], " columns.")
        print("The Term list has", len(terms), " terms.")
        term_idf_scores.sort(key=sortSecond, reverse=True)
        print("\nTerms with Highest TF-IDF Scores:")
        for i in range(10):
            print('{:<15s}{:>8.2f}'.format(term_idf_scores[i][0],  term_idf_scores[i][1]))




        # In sklearn, LDA is synonymous with SVD (according to their doc)
        lda = LatentDirichletAllocation(n_components=n_topics, max_iter=max_iter,learning_method=learning_method, learning_offset=learning_offset, random_state=seed)
        lda.fit_transform(tf)
        print('{:.<22s}{:>6d}'.format("Number of Reviews", n_comments))
        print('{:.<22s}{:>6d}'.format("Number of Terms", len(terms)))
        print("\nTopics Identified using LDA with TF_IDF")
        display_topics(lda.components_, terms, n_terms=20)




        # Review Scores
        # Normalize LDA Weights to probabilities
        lda_norm = lda.components_ / lda.components_.sum(axis=1)[:, np.newaxis]
        # ***** SCORE REVIEWS *****
        rev_scores = [[0]*(n_topics+1)] * n_comments
        # Last topic count is number of reviews without any topic words
        topic_counts = [0] * (n_topics+1)
        for r in range(n_comments):
            idx = n_topics
            max_score = 0
            # Calculate Review Score
            j0 = tf[r].nonzero()
            nwords = len(j0[1])
            rev_score = [0]*(n_topics+1)
            # get scores for rth doc, ith topic
            for i in range(n_topics):
                score = 0
                for j in range(nwords):
```

```
                    j1 = j0[1][j]
                    if tf[r,j1] != 0:
                        score += lda_norm[i][j1] * tf[r,j1]
                rev_score [i+1] = score
                if score>max_score:
                    max_score = score
                    idx = i
        # Save review's highest scores
            rev_score[0] = idx
            rev_scores [r] = rev_score
            topic_counts[idx] += 1

        # Augment Dataframe with topic group information
        cols = ["topic"]
        for i in range(n_topics):
            s = "T"+str(i+1)
            cols.append(s)
        df_topics = pd.DataFrame.from_records(rev_scores, columns=cols)
        df         = df.join(df_topics)



        print("\n**** Sentiment Analysis ****")
        sw = pd.read_excel(file_path + "/Afinn_sentiment_words.xlsx")

        # setup sentiment dictionary
        sentiment_dic = {}
        for i in range(len(sw)):
            sentiment_dic[sw.iloc[i][0]] = sw.iloc[i][1]


        # Create Word Frequency by Review Matrix using Custom Analyzer
        # max_df is a stop limit for terms that have more than this
        # proportion of documents with the term (max_df - don't ignore any terms)
        cv = CountVectorizer(max_df=1.0, min_df=1, max_features=None, preprocessor=my_preprocessor, ngram_range=(1,2))
        tf = cv.fit_transform(df['description'])
        terms = cv.get_feature_names()
        n_terms = tf.shape[1]
        print('{:.<22s}{:>6d}'.format("Number of Reviews", n_comments))
        print('{:.<22s}{:>6d}'.format("Number of Terms", n_terms))



        # calculate average sentiment for every review save in sentiment_score[]
        min_sentiment = +5
        max_sentiment = -5
        avg_sentiment, min, max = 0,0,0
        min_list, max_list = [],[]
        sentiment_score = [0]*n_comments
        for i in range(n_comments):
            # iterate over the terms with nonzero scores
            n_sw = 0
            term_list = tf[i].nonzero()[1]
            if len(term_list) >0:
                for t in np.nditer(term_list):
                    score = sentiment_dic.get(terms[t])
                    if score !=None:
                        sentiment_score[i] += score * tf[i, t]
                        n_sw += tf[i, t]
            if n_sw >0:
```

```
                sentiment_score[i] = sentiment_score[i]/n_sw
            if sentiment_score[i]==max_sentiment and n_sw >3:
                max_list.append(i)
            if sentiment_score[i]>max_sentiment and n_sw>3:
                max_sentiment=sentiment_score[i]
                max=i
                max_list=[i]
            if sentiment_score[i]==min_sentiment and n_sw >3:
                min_list.append(i)
            if sentiment_score[i]<min_sentiment and n_sw>3:
                min_sentiment=sentiment_score[i]
                min=i
                min_list=[i]
            avg_sentiment += sentiment_score[i]
        avg_sentiment = avg_sentiment/n_comments
        print ("\nCorpus Average Sentiment: ", avg_sentiment)
        print ("\nMost Negative Reviews with 4 or more Sentiment Words:")
        for i in range(len(min_list)):
            print("{:<s}{:<d}{:<s}{:<5.2f}".format("    Review ", min_list[i],
                                     " sentiment is ", min_sentiment))
        print("\nMost Positive Reviews with 4 or more Sentiment Words:")
        for i in range(len(max_list)):
             print("{:<s}{:<d}{:<s}{:<5.2f}".format("    Review ", max_list[i],
                                      " sentiment is ", max_sentiment))

        # Augment Dataframe with  sentiment score information
        cols = ["sentiment"]
        df_score = pd.DataFrame(sentiment_score, columns=cols)
        df        = df.join(df_score)

        #Average Sentiment by topic
        df.groupby(['topic'])['sentiment'].mean()

        #Average Sentiment by make
        df.groupby(['Make'])['sentiment'].mean()

        #Average Sentiment by model
        df.groupby(['Model'])['sentiment'].mean()

        #Average Sentiment by make, topic and model
        df.groupby(['Make','topic','Model'])['sentiment'].mean()

        print('***Topic by Complaints count***')
        df.groupby('topic').topic.count()

        print("\n**** Decision tree Analysis ****")
        # create attribute map
        # Attribute Map:  the key is the name in the DataFrame
        # The first number of 0=Interval, 1=binary and 2=nomial
        # The 1st tuple for interval attributes is their lower and upper bounds
        # The 1st tuple for categorical attributes is their allowed categories
        # The 2nd tuple contains the number missing and number of outliers
        attribute_map = {
            "NhtsaID":[3, (560001,10891880), [0,0]],
            "Make": [1, ("HONDA", "ACURA"), [0,0]],
            "Model":[2, ("TL", "ODYSSEY", "CR-V", "CL", "CIVIC", "ACCORD"),[0,0]],
```

```
            "Year":[2, (2001, 2002, 2003), [0,0]],
            "State": [3, (""), [0,0]],
            "abs": [1, ("Y", "N"), [0,0]],
            "cruise":[1, ("Y", "N"), [0,0]],
            "crash": [1, ("Y","N"), [0,0]],
            "mph": [0, (0,80), [0,0]],
            "mileage": [0,(0,200000), [0,0]],
            'topic':[2,(0,1,2,3,4,5,6),[0,0]],
            'T1':[0,(-1e+8,1e+8),[0,0]],
            'T2':[0,(-1e+8,1e+8),[0,0]],
            'T3':[0,(-1e+8,1e+8),[0,0]],
            'T4':[0,(-1e+8,1e+8),[0,0]],
            'T5':[0,(-1e+8,1e+8),[0,0]],
            'T6':[0,(-1e+8,1e+8),[0,0]],
            'T7':[0,(-1e+8,1e+8),[0,0]],
            "sentiment":[0,(-1e+8,1e+8),[0,0]]


        }


        # drop=False - used for Decision tree
        rie = ReplaceImputeEncode(data_map=attribute_map, nominal_encoding='one-hot', interval_scale = 'std',drop = False, display=True)
        encoded_df = rie.fit_transform(df)
        #create X and y
        varlist = ["crash", 'T1', 'T2', 'T3', 'T4', 'T5', 'T6', 'T7']
        y = encoded_df["crash"]
        X = encoded_df.drop(varlist, axis=1)
        np_y = np.ravel(y)
        col  = rie.col
        for i in range(len(varlist)):
            col.remove(varlist[i])




        # Cross Validation for decision tree:
        # best model: Maximum Tree Depth:  15 Min_samples_leaf 3 Min_samples_split 5
        depth_list = [5,6,8,10, 12, 15, 20, 25, 50]
        minSamplesLeaf= [3,5,7]
        minSamplesSplit=[3]

        recall_best = 0
        recall_best_model = ''
        f1score_best = 0
        f1score_best_model = ''
        accuracy_best = 0
        accuracy_best_model = ''
        precision_best = 0
        precision_best_model = ''

        score_list = ['accuracy', 'recall', 'precision', 'f1']
        for d in depth_list:
            for l in minSamplesLeaf:
                for s in minSamplesSplit:
                    print("\nMaximum Tree Depth: ", d, "Min_samples_leaf", l, "Min_samples_split", s)
                    dtc = DecisionTreeClassifier(max_depth=d, min_samples_leaf=l,  min_samples_split=s, random_state=seed)
                    dtc = dtc.fit(X,np_y)
```

```
                scores = cross_validate(dtc, X, np_y, scoring=score_list, return_train_score=False, cv=10)

                print("{:.<13s}{:>6s}{:>13s}".format("Metric", "Mean", "Std. Dev."))
                for sl in score_list:
                    var = "test_"+sl
                    mean = scores[var].mean()
                    std  = scores[var].std()
                    if sl=='recall':
                        if recall_best<mean:
                            recall_best = mean
                            recall_best_model ="d:"+str(d)+" l:"+str(l)+" s:"+str(s)
                    if sl=='precision':
                        if precision_best<mean:
                            precision_best = mean
                            precision_best_model ="d:"+str(d)+" l:"+str(l)+" s:"+str(s)
                    if sl=='f1':
                        if f1score_best<mean:
                            f1score_best = mean
                            f1score_best_model ="d:"+str(d)+" l:"+str(l)+" s:"+str(s)
                    if sl=='accuracy':
                        if accuracy_best<mean:
                            accuracy_best = mean
                            accuracy_best_model ="d:"+str(d)+" l:"+str(l)+" s:"+str(s)

                    print("{:.<13s}{:>7.4f}{:>10.4f}".format(sl, mean, std))

    # =============================================================================
    # d: depth; l: leaf size; s: splits
    # recall_best
    # 0.53917120387174822
    #
    # recall_best_model
    # 'd:15 l:5 s:3'
    #
    # f1score_best
    # 0.59287652022030457
    #
    # f1score_best_model
    # 'd:15 l:5 s:3'
    #
    # accuracy_best
    # 0.92382945659149662
    #
    # accuracy_best_model
    # 'd:8 l:5 s:3'
    #
    # precision_best
    # 0.76163029737558041
    #
    # precision_best_model
    # 'd:6 l:5 s:3'
    # =============================================================================

```

```
        #15 ,3,5
        # 70/30 split
        X_train, X_validate, y_train, y_validate =  train_test_split(X, np_y,test_size = 0.3, random_state=seed)

        # Decison Tree
        dtc = DecisionTreeClassifier(max_depth=15, min_samples_leaf=5, min_samples_split=3, random_state=seed)
        dtc = dtc.fit(X_train,y_train)
        DecisionTree.display_binary_split_metrics(dtc, X_train, y_train,X_validate, y_validate)

        DecisionTree.display_importance(dtc, col)

        # =============================================================================
        # Training
        # Confusion Matrix  Class 0   Class 1
        # Class 0.....      3292        50
        # Class 1.....       118       271
        #
        #
        # Validation
        # Confusion Matrix  Class 0   Class 1
        # Class 0.....      1373        44
        # Class 1.....        84        98
        # =============================================================================

        print('***** Train set ******')
        getClassificationMetrics(3292, 50, 118, 271)


        print('\n')
        print('***** Validation set ******')
        getClassificationMetrics(1373, 44, 84, 98)

        # =============================================================================
        # ***** Train set ******
        # sensitivity/recall/TPR: 0.6966580976863753
        # specificity:  0.9850388988629563
        # accuracy:  0.9549718574108818
        # precision:  0.8442367601246106
        # f1_score:  0.7633802816901407
        # misc:  0.0450281425891182
        # FPR:  0.014961101137043742
        #
        #
        # ***** Validation set ******
        # sensitivity/recall/TPR: 0.5384615384615384
        # specificity:  0.9689484827099506
        # accuracy:  0.9199499687304565
        # precision:  0.6901408450704225
        # f1_score:  0.6049382716049383
        # misc:  0.08005003126954346
        # FPR:  0.031051517290049402
        #
        # =============================================================================

```

```



# ===============================================================================
# PART 2
# Web Scrapping - Search Word 'Takata'
# API Key: 444171d89d544b2da002bb61fe78833a
# ===============================================================================

import re
import requests
import newspaper
from newspaper import Article
from newsapi import NewsApiClient # Needed for using API Feed
from time import time

# News Agencies used by API
agency_urls = {
'huffington': 'http://huffingtonpost.com',
'reuters': 'http://www.reuters.com',
'cbs-news': 'http://www.cbsnews.com',
'usa-today': 'http://usatoday.com',
'cnn': 'http://cnn.com',
'npr': 'http://www.npr.org',
'wsj': 'http://wsj.com',
'fox': 'http://www.foxnews.com',
'abc': 'http://abc.com',
'abc-news': 'http://abcnews.com',
'abcgonews': 'http://abcnews.go.com',
'nyt': 'http://nytimes.com',
'washington-post': 'http://washingtonpost.com',
'us-news': 'http://www.usnews.com',
'msn': 'http://msn.com',
'pbs': 'http://www.pbs.org',
'nbc-news': 'http://www.nbcnews.com',
'enquirer': 'http://www.nationalenquirer.com',
'la-times': 'http://www.latimes.com'
}

# ===============================================================================
# Clean the donloaded content to remove HTML, CSS, and Javascript code.
# ===============================================================================
def clean_html(html):
    # First we remove inline JavaScript/CSS:
    pg = re.sub(r"(?is)<(script|style).*?>.*?(</\1>)", "", html.strip())
    # Then we remove html comments. This has to be done before removing regular

    # tags since comments can contain '>' characters.
    pg = re.sub(r"(?s)<!--(.*?)-->[\n]?", "", pg)
    # Next we can remove the remaining tags:
    pg = re.sub(r"(?s)<.*?>", " ", pg)
    # Finally, we deal with whitespace
    pg = re.sub(r" ", " ", pg)
    pg = re.sub(r"’", "'", pg)
    pg = re.sub(r"“", '"''"', pg)
    pg = re.sub(r"”", '"', pg)
    pg = re.sub(r"\n", " ", pg)
```

```
    pg = re.sub(r"\t", " ", pg)
    pg = re.sub(r" ", " ", pg)
    pg = re.sub(r" ", " ", pg)
    pg = re.sub(r" ", " ", pg)
    return pg.strip()

# =============================================================================
# Get news URLS
# =============================================================================
def newsapi_get_urls(search_words, agency_urls):
    if len(search_words)==0 or agency_urls==None:
        return None
    print("Searching agencies for pages containing:", search_words)
    # This is my API key, each user must request their own
    # API key from https://newsapi.org/account
    api = NewsApiClient(api_key='444171d89d544b2da002bb61fe78833a')
    api_urls = []
    # Iterate over agencies and search words to pull more url's
    # Limited to 1,000 requests/day - Likely to be exceeded
    for agency in agency_urls:
        domain = agency_urls[agency].replace("http://", "")
        print(agency, domain)
        for word in search_words:
    # Get articles with q= in them, Limits to 20 URLs
            try:
                articles = api.get_everything(q=word, language='en',\
                sources=agency, domains=domain)
            except:
                print("--->Unable to pull news from:", agency, "for", word)
                continue
    # Pull the URL from these articles (limited to 20)
            d = articles['articles']
            for i in range(len(d)):
                url = d[i]['url']
                api_urls.append([agency, word, url])
    df_urls = pd.DataFrame(api_urls, columns=['agency', 'word', 'url'])
    n_total = len(df_urls)
    # Remove duplicates
    df_urls = df_urls.drop_duplicates('url')
    n_unique = len(df_urls)
    print("\nFound a total of", n_total, " URLs, of which", n_unique,\
    " were unique.")
    return df_urls

# =============================================================================
# Get Downloaded Content from URLs obtained
# =============================================================================
def request_pages(df_urls):
    web_pages = []
    for i in range(len(df_urls)):
        u = df_urls.iloc[i]
        url = u[2]
        short_url = url[0:50]
        short_url = short_url.replace("https//", "")
        short_url = short_url.replace("http//", "")
        n = 0
        # Allow for a maximum of 5 download failures
```

```
        stop_sec=3 # Initial max wait time in seconds
        while n<3:
            try:
                r = requests.get(url, timeout=(stop_sec))
                if r.status_code == 408:
                    print("-->HTML ERROR 408", short_url)
                    raise ValueError()
                if r.status_code == 200:
                    print("Obtained: "+short_url)
                else:
                    print("-->Web page: "+short_url+" status code:", \
                r.status_code)
                n=99
                continue # Skip this page
            except:
                n += 1
                # Timeout waiting for download
                t0 = time()
                tlapse = 0
                print("Waiting", stop_sec, "sec")
                while tlapse<stop_sec:
                    tlapse = time()-t0
        if n != 99:
        # download failed skip this page
            continue
        # Page obtained successfully

        html_page = r.text
        page_text = clean_html(html_page)
        #print(page_text)
        web_pages.append([url, page_text])
    df_www = pd.DataFrame(web_pages, columns=['url', 'text'])
    n_total = len(df_urls)
    # Remove duplicates
    df_www = df_www.drop_duplicates('url')
    n_unique = len(df_urls)
    print("Found a total of", n_total, " web pages, of which", n_unique,\
    " were unique.")
    return df_www

#Search word
search_words = ['Takata']
df_urls = newsapi_get_urls(search_words, agency_urls)
print("Total Articles:", df_urls.shape[0])


print("Agency:", df_urls.iloc[0]['agency'])
print("Search Word:", df_urls.iloc[0]['word'])
print("URL:", df_urls.iloc[0]['url'])


# Download Discovered Pages
df_www = request_pages(df_urls)
# Store in Excel File
df_www.to_excel('/Users/sasha/Desktop/df_www.xlsx')


for i in range(df_www.shape[0]):
    short_url = df_www.iloc[i]['url']
    short_url = short_url.replace("https://", "")
    short_url = short_url.replace("http://", "")
```

```
            short_url = short_url[0:60]
            page_char = len(df_www.iloc[i]['text'])
            print("{:<60s}{:>10d} Characters".format(short_url, page_char))

```

APPENDIX B : PYTHON WEBSCRAPPING SITELIST

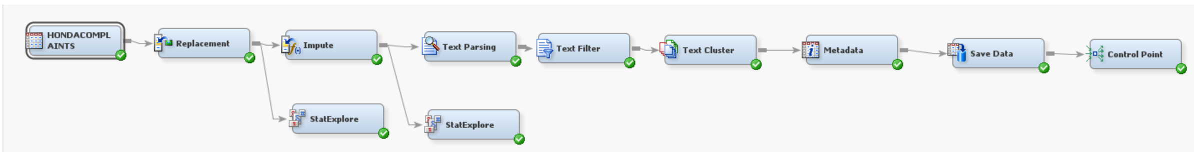
```
Obtained: https://www.reuters.com/article/us-autos-takata/ho
Obtained: https://in.reuters.com/article/autos-takata/honda-
Obtained: https://www.reuters.com/article/us-autos-takata/u-
Obtained: https://www.reuters.com/article/us-takata-whistleb
Obtained: https://uk.reuters.com/article/uk-autos-takata/hon
Obtained: https://www.reuters.com/article/us-takata-pricefix
Obtained: https://www.reuters.com/article/us-autos-takata/se
Obtained: https://ca.reuters.com/article/businessNews/idCAKC
Obtained: https://www.reuters.com/article/us-takata-sale-key
Obtained: https://www.reuters.com/article/us-autos-takata/au
Obtained: https://in.reuters.com/article/autos-takata/automa
Obtained: https://www.reuters.com/article/us-takata-bankrupt
Obtained: https://www.reuters.com/article/us-takata-bankrupt
Obtained: https://in.reuters.com/article/takata-bankruptcy-s
Obtained: https://www.reuters.com/article/us-takata-bankrupt
Obtained: https://ca.reuters.com/article/businessNews/idCAKC
Obtained: https://www.reuters.com/article/us-takata-bankrupt
Obtained: https://www.reuters.com/article/takata-pricefixing
Obtained: https://www.reuters.com/article/takata-bankruptcy-
Obtained: https://www.usatoday.com/story/money/cars/2018/04/
Obtained: https://www.usatoday.com/story/money/cars/2018/02/
Obtained: https://www.usatoday.com/story/money/cars/2018/02/
Obtained: http://money.cnn.com/2018/02/27/news/companies/tak
Obtained: http://money.cnn.com/2018/03/19/news/companies/hyu
Obtained: https://www.wsj.com/articles/takata-whistleblower-
Obtained: https://www.wsj.com/articles/takata-settles-with-d
Obtained: https://www.wsj.com/articles/takata-settles-joint-
Obtained: https://www.wsj.com/articles/regulator-car-executi
Obtained: https://www.wsj.com/articles/more-auto-makers-sued
Obtained: https://www.wsj.com/articles/senators-press-car-ex
Obtained: https://www.wsj.com/articles/u-s-investigates-fail
Obtained: https://www.wsj.com/articles/companies-everywhere-
Obtained: http://abcnews.go.com/Business/wireStory/takata-ac
Obtained: http://abcnews.go.com/International/wireStory/taka
Waiting 3 sec
Obtained: http://abcnews.go.com/International/wireStory/taka
Obtained: http://abcnews.go.com/International/wireStory/judg
Obtained: http://abcnews.go.com/Business/wireStory/states-fo
Obtained: http://abcnews.go.com/International/wireStory/zeal
Obtained: http://abcnews.go.com/Business/wireStory/air-bag-d
Obtained: http://abcnews.go.com/Business/wireStory/lawsuits-
Obtained: http://abcnews.go.com/Business/wireStory/3rd-time-
Obtained: http://abcnews.go.com/International/wireStory/aust
```

```
Obtained: http://abcnews.go.com/Business/wireStory/hondas-pr
Obtained: http://abcnews.go.com/Business/wireStory/business-
Obtained: http://abcnews.go.com/Business/wireStory/business-
Obtained: http://abcnews.go.com/Business/wireStory/business-
Obtained: https://www.nytimes.com/2018/02/22/business/takata
Obtained: https://www.nytimes.com/2018/02/11/business/takata
Obtained: https://www.nytimes.com/2018/02/28/briefing/xi-jin
Obtained: https://www.nytimes.com/2018/02/23/business/dealbo
-->Web page: https://www.washingtonpost.com/world/asia_pacific/ status code: 404
-->Web page: https://www.washingtonpost.com/world/asia_pacific/ status code: 404
-->Web page: https://www.washingtonpost.com/world/new-zealand-r status code: 404
-->Web page: https://www.washingtonpost.com/world/australia-iss status code: 404
-->Web page: https://www.washingtonpost.com/world/australia-iss status code: 404
Obtained: https://www.washingtonpost.com/world/asia_pacific/
Obtained: https://www.washingtonpost.com/news/the-switch/wp/
Obtained: https://www.washingtonpost.com/business/economy/am
Obtained: https://www.washingtonpost.com/news/dr-gridlock/wp
Waiting 3 sec
Obtained: https://www.nbcnews.com/news/us-news/hyundai-kia-u
Found a total of 60  web pages, of which 60  were unique.
```

```
www.reuters.com/article/us-autos-takata/honda-ford-to-testif       4866 Characters
in.reuters.com/article/autos-takata/honda-ford-to-testify-at       4875 Characters
www.reuters.com/article/us-autos-takata/u-s-senators-call-ne       4794 Characters
www.reuters.com/article/us-takata-whistleblowers/takata-whis       5673 Characters
uk.reuters.com/article/uk-autos-takata/honda-ford-to-testify       4987 Characters
www.reuters.com/article/us-takata-pricefixing/south-africa-a       3131 Characters
www.reuters.com/article/us-autos-takata/senators-to-press-au       5733 Characters
ca.reuters.com/article/businessNews/idCAKCN1G10SW-OCABS            3294 Characters
www.reuters.com/article/us-takata-sale-key-safety-systems/ke       5947 Characters
www.reuters.com/article/us-autos-takata/automakers-knew-earl       5083 Characters
in.reuters.com/article/autos-takata/automakers-knew-earlier-       5122 Characters
www.reuters.com/article/us-takata-bankruptcy-hearing/takata-       4538 Characters
www.reuters.com/article/us-takata-bankruptcy-settlement/auto       6416 Characters
in.reuters.com/article/takata-bankruptcy-settlement/automake       6389 Characters
www.reuters.com/article/us-takata-bankruptcy-ruling/judge-ap       3708 Characters
ca.reuters.com/article/businessNews/idCAKCN1FX2VL-OCABS            6078 Characters
www.reuters.com/article/us-takata-bankruptcy-settlement/taka       5010 Characters
www.reuters.com/article/takata-pricefixing/south-africa-anti       2338 Characters
www.reuters.com/article/takata-bankruptcy-settlement/takata-       4762 Characters
www.usatoday.com/story/money/cars/2018/04/12/takata-acquired       7521 Characters
www.usatoday.com/story/money/cars/2018/02/12/takata-settles-       6784 Characters
www.usatoday.com/story/money/cars/2018/02/12/air-bag-danger-       7241 Characters
money.cnn.com/2018/02/27/news/companies/takata-airbags-austr       7893 Characters
money.cnn.com/2018/03/19/news/companies/hyundai-kia-airbag-i       7171 Characters
www.wsj.com/articles/takata-whistleblower-claimants-settle-f      17631 Characters
www.wsj.com/articles/takata-settles-with-drivers-injured-by-      17220 Characters
www.wsj.com/articles/takata-settles-joint-probe-by-u-s-state      17612 Characters
www.wsj.com/articles/regulator-car-executives-to-testify-at-      17595 Characters
www.wsj.com/articles/more-auto-makers-sued-over-exploding-ta      17594 Characters
www.wsj.com/articles/senators-press-car-executives-regulator      17609 Characters
www.wsj.com/articles/u-s-investigates-failing-air-bags-in-hy      17538 Characters
www.wsj.com/articles/companies-everywhere-copied-japanese-ma      18030 Characters
abcnews.go.com/Business/wireStory/takata-acquired-key-safety      78553 Characters
abcnews.go.com/International/wireStory/takata-acquired-key-s      78553 Characters
abcnews.go.com/International/wireStory/takata-corp-maker-def      78553 Characters
abcnews.go.com/International/wireStory/judge-approves-takata      78553 Characters
abcnews.go.com/Business/wireStory/states-forego-650m-legal-s      78553 Characters
abcnews.go.com/International/wireStory/zealand-recalls-50000      78553 Characters
abcnews.go.com/Business/wireStory/air-bag-danger-ford-adds-3      78553 Characters
abcnews.go.com/Business/wireStory/lawsuits-accuse-automakers      78553 Characters
abcnews.go.com/Business/wireStory/3rd-time-general-motors-se      78553 Characters
abcnews.go.com/International/wireStory/australia-issues-comp      78553 Characters
abcnews.go.com/Business/wireStory/hondas-profit-climbs-growi       8568 Characters
```

```
abcnews.go.com/Business/wireStory/business-highlights-537565      78553 Characters
abcnews.go.com/Business/wireStory/business-highlights-532880      78553 Characters
abcnews.go.com/Business/wireStory/business-highlights-538892      78553 Characters
www.nytimes.com/2018/02/22/business/takata-airbags-settlemen      26121 Characters
www.nytimes.com/2018/02/11/business/takata-bankruptcy-airbag      22581 Characters
www.nytimes.com/2018/02/28/briefing/xi-jinping-jared-kushner      11564 Characters
www.nytimes.com/2018/02/23/business/dealbook/business-gun-co      45229 Characters
www.washingtonpost.com/world/asia_pacific/takata-acquired-by       2281 Characters
www.washingtonpost.com/world/asia_pacific/takata-corp-maker-       2281 Characters
www.washingtonpost.com/world/new-zealand-recalls-50000-cars-       2281 Characters
www.washingtonpost.com/world/australia-issues-compulsory-rec       2281 Characters
www.washingtonpost.com/world/australia-issues-compulsory-rec       2281 Characters
www.washingtonpost.com/world/asia_pacific/hondas-profit-clim       2898 Characters
www.washingtonpost.com/news/the-switch/wp/2018/03/14/apple-g       5558 Characters
www.washingtonpost.com/business/economy/amazon-trimming-jobs       5680 Characters
www.washingtonpost.com/news/dr-gridlock/wp/2018/04/27/shed-s       5963 Characters
www.nbcnews.com/news/us-news/hyundai-kia-under-scrutiny-air-       7304 Characters
```

APPENDIX C: SAS DIAGRAM 1 WITH NODE PROPERTIES



Input Data Node Properties

Property	Value
General	
Node ID	Ids
Imported Data	...
Exported Data	...
Notes	...
Train	
Output Type	Data
Role	Raw
Rerun	No
Summarize	No
Drop Map Variables	No
Columns	
Variables	...
Decisions	...
Refresh Metadata	...
Advisor	Basic
Advanced Options	...
Data	
Data Selection	Data Source
Sample	Default
Sample Options	...
Data Source	
Data Source	HONDACOMPLAINTS
Data Source Properties	...
New Table	
Table Name	...
Variable Validation	Strict
New Variable Role	Reject
Metadata	
Table	HONDACOMPLAINTS

Name	Role	Level	Report	Order	Drop	Lower Limit	Upper Limit
Make	Input	Binary	No		No	.	.
Model	Input	Nominal	No		No	.	.
NhtsaID	ID	Interval	No		No	.	.
State	Rejected	Nominal	No		No	.	.
Year	Input	Nominal	No		No	.	.
abs	Input	Binary	No		No	.	.
crash	Target	Binary	No		No	.	.
cruise	Input	Binary	No		No	.	.
description	Text	Nominal	No		No	.	.
mileage	Input	Interval	No		No	.	.
mph	Input	Interval	No		No	.	.

Replacement Node Properties

Property	Value
General	
Node ID	Repl
Imported Data	...
Exported Data	...
Notes	...
Train	
Interval Variables	
Replacement Editor	...
Default Limits Method	User-Specified Limits
Cutoff Values	...
Class Variables	
Replacement Editor	...
Unknown Levels	Ignore
Score	
Replacement Values	User-Specified
Hide	No
Report	
Replacement Report	Yes
Status	
Create Time	30/4/18 9:09 PM
Run ID	97d75d9e-194e-47c7-8f88
Last Error	
Last Status	Complete
Last Run Time	2/5/18 10:34 PM
Run Duration	0 Hr. 0 Min. 4.77 Sec.
Grid Host	
User-Added Node	No

Columns: ☐ Label ☐ Mining ☐ Basic ☐ Statistics

Name	Use	Limit Method	Replacement Lower Limit	Replacement Upper Limit	Replace
mileage	Default	Default	0	200000	Missing
mph	Default	Default	0	80	Missing

Impute Node Properties

General	
Node ID	Impt
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
Nonmissing Variables	No
Missing Cutoff	50.0
Class Variables	
Default Input Method	Tree
Default Target Method	None
Normalize Values	Yes
Interval Variables	
Default Input Method	Tree
Default Target Method	None
Default Constant Value	
Default Character Value	
Default Number Value	.
Method Options	
Random Seed	12345
Tuning Parameters	...
Tree Imputation	...
Score	

Text Parsing Node Properties

.. Property	Value
General	
Node ID	TextParsing
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
Parse	
Parse Variable	description
Language	English
Detect	
Different Parts of Speech	Yes
Noun Groups	Yes
Multi-word Terms	SASHELP.ENG_MULTI
Find Entities	None
Custom Entities	
Ignore	
Ignore Parts of Speech	'Aux' 'Conj' 'Det' 'Interj' 'I
Ignore Types of Entities	...
Ignore Types of Attributes	'Num' 'Punct'
Synonyms	
Stem Terms	Yes
Synonyms	SASHELP.ENGSYNMS
Filter	
Start List	...
Stop List	SASHELP.ENGSTOP
Select Languages	...
Report	
Number of Terms to Display	20000
Status	

Text Filter Node Properties

.. Property	Value
General	
Node ID	TextFilter
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
Spelling	
Check Spelling	No
Dictionary	...
Weightings	
Frequency Weighting	None
Term Weight	Inverse Document Frequency
Term Filters	
Minimum Number of Documer	4
Maximum Number of Terms	.
Import Synonyms	...
Document Filters	
Search Expression	
Subset Documents	...
Results	
Filter Viewer	...
Spell-Checking Results	...
Exported Synonyms	...
Report	
Terms to View	All
Number of Terms to Display	20000

Text Cluster Node Properties

General	
Node ID	TextCluster
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
Transform	
SVD Resolution	High
Max SVD Dimensions	100
Cluster	
Exact or Maximum Number	Exact
Number of Clusters	7
Cluster Algorithm	Expectation-Maximization
Descriptive Terms	15

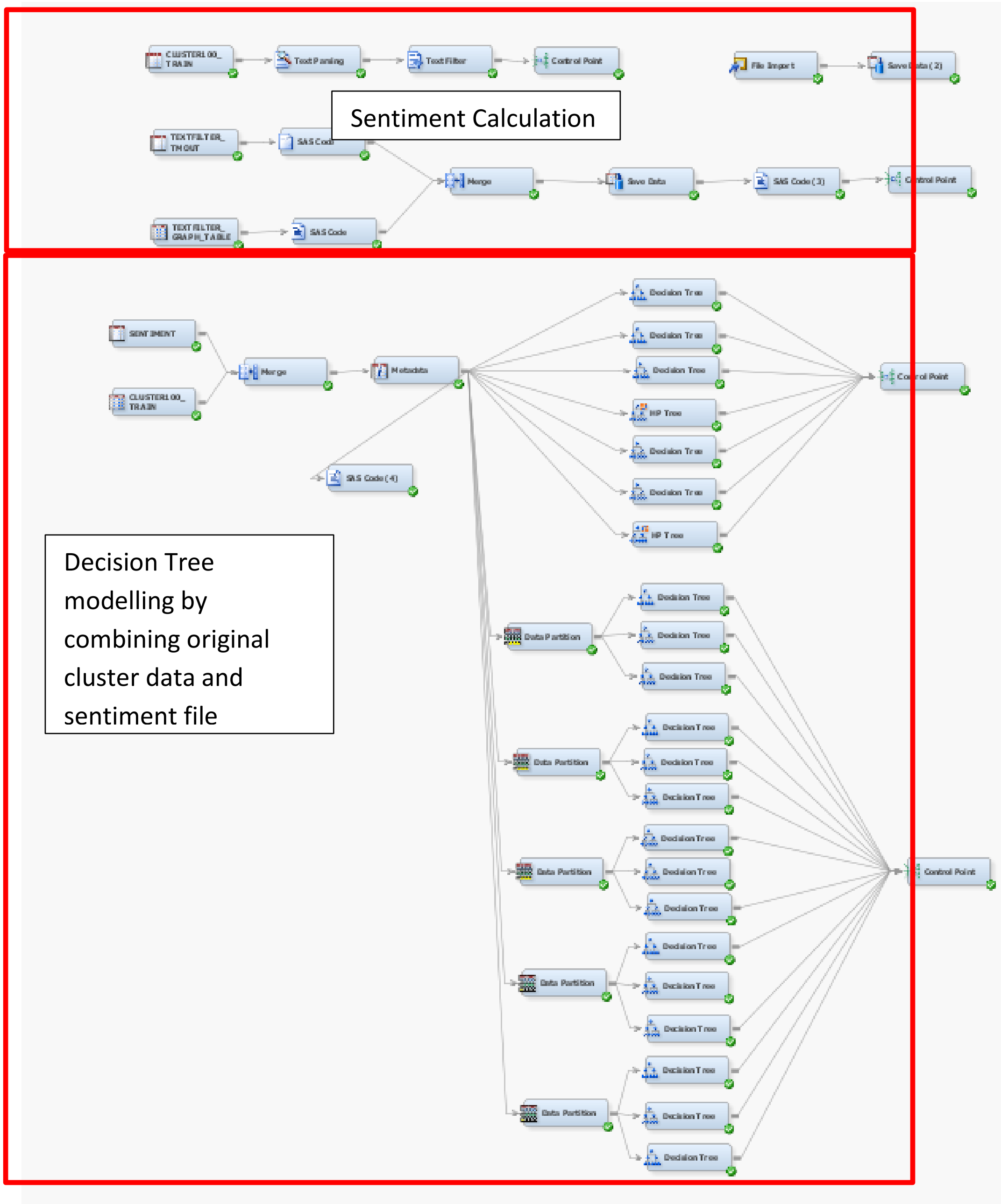
Metadata Properties

Name	Hidden	Hide	Role	New Role	Level
IMP_REP_milea	N	**Default**	Input	**Default**	Interval
IMP_REP_mph	N	**Default**	Input	**Default**	Interval
Make	N	**Default**	Input	**Default**	Binary
Model	N	**Default**	Input	**Default**	Nominal
NhtsaID	N	**Default**	ID	**Default**	Interval
REP_mileage	Y	**Default**	Rejected	**Default**	Interval
REP_mph	Y	**Default**	Rejected	**Default**	Interval
State	N	**Default**	Rejected	**Default**	Nominal
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval
TextCluster_SV	N	**Yes**	Input	**Rejected**	Interval

Save Data Node Properties

.. Property	Value
General	
Node ID	EMSave
Imported Data	...
Exported Data	...
Notes	...
Train	
Output Options	
Variables	...
Filename Prefix	CLUSTER100
Replace Existing Files	Yes
All Observations	Yes
Number of Observations	1000
Output Format	
File Format	SAS (.sas7bdat)
SAS Library Name	MYLIB
Directory	...
Output Data	
All Roles	Yes
Select Roles	...
Status	
Create Time	30/4/18 10:18 PM
Run ID	791a4034-d777-442e-9709-5
Last Error	
Last Status	Complete
Last Run Time	2/5/18 10:35 PM
Run Duration	0 Hr. 0 Min. 5.27 Sec.
Grid Host	
User-Added Node	No

APPENDIX D: SAS DIAGRAM 2 WITH NODE PROPERTIES



Input Data Node Properties

Property	Value
General	
Node ID	Ids
Imported Data	...
Exported Data	...
Notes	...
Train	
Output Type	Data
Role	Raw
Rerun	No
Summarize	No
Drop Map Variables	No
Columns	
Variables	...
Decisions	...
Refresh Metadata	...
Advisor	Basic
Advanced Options	...
Data	
Data Selection	Data Source
Sample	Default
Sample Options	...
Data Source	
Data Source	CLUSTER100_TRAIN ...
Data Source Properties	...

Text Parsing Properties

General	
Node ID	TextParsing
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
Parse	
Parse Variable	description
Language	English ...
Detect	
Different Parts of Speech	No
Noun Groups	Yes
Multi-word Terms	SASHELP.ENG_MULTI ...
Find Entities	None
Custom Entities	
Ignore	
Ignore Parts of Speech	'Aux' 'Conj' 'Det' 'Interj' ' ...
Ignore Types of Entities	...
Ignore Types of Attributes	'Num' 'Punct' ...
Synonyms	
Stem Terms	No
Synonyms	SASHELP.ENGSYNMS ...
Filter	
Start List	MYLIB.AFINN_STARTLIST ...
Stop List	...
Select Languages	...
Report	
Number of Terms to Display	All

Text Filter Properties

General	
Node ID	TextFilter
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
Spelling	
Check Spelling	No
Dictionary	...
Weightings	
Frequency Weighting	None
Term Weight	None
Term Filters	
Minimum Number of Documer	1
Maximum Number of Terms	.
Import Synonyms	...
Document Filters	
Search Expression	
Subset Documents	...
Results	
Filter Viewer	...
Spell-Checking Results	...
Exported Synonyms	...
Report	
Terms to View	Selected
Number of Terms to Display	All
Status	

Input Data Node Properties for TM OUT and graph table

Property	Value
General	
Node ID	Ids2
Imported Data	...
Exported Data	...
Notes	...
Train	
Output Type	Data
Role	Raw
Rerun	Yes
Summarize	No
Drop Map Variables	No
Columns	
Variables	...
Decisions	...
Refresh Metadata	...
Advisor	Basic
Advanced Options	...
Data	
Data Selection	Data Source
Sample	Default
Sample Options	...
Data Source	
Data Source	TEXTFILTER_TMOUT ...
Data Source Properties	...

SAS CODE for TMOUT

```
data &EM_EXPORT_TRAIN;
RENAME _COUNT_ =COUNT _TERMNUM_ =TERMNUMBER;
set &EM_IMPORT_DATA;

proc sort data=&em_export_Train;
by TERMNUMBER;
run;
```

SAS CODE FOR GRAPH TABLE

```
data &EM_EXPORT_TRAIN;
RENAME PARENT_ID = TermNumber NUMDOCS = NDOCS;
Keep Freq Term Parent_ID NUMDOCS;
set &EM_IMPORT_DATA;
if KEEP EQ "Y" then output;

proc sort data=&EM_EXPORT_TRAIN;
BY TERMNUMBER;
RUN;
```

MERGE Node Properties

Property	Value
General	
Node ID	Merge
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
Merging	Match
By Ordering	...
Overwrite Variables	No
Variables Group	
Segment	No
Assess	No
Classification	No
Predicted or Posterior	No
Residual	No

Name	Merge Role	Overwrite Variable	Role	Level
COUNT	none	Default	Input	Interval
TERMNUMBER	By	Default	Input	Interval
DOCUMENT	none	Default	Input	Interval

SAS CODE FOR SENTIMENT CALCULATION

```
proc sort data=MyLib.TermDocMatrix_train;
by Term;

proc sort data=MyLib.afinn_senti_train;
by Term;

Data MyLib.TermDocMatrix2;
merge MyLib.TermDocMatrix_train MyLib.afinn_senti_train;
by Term;
Keep _Document_ termNumber Term Count Score;
if COUNT NE . AND TERM NE ' ' AND _DOCUMENT_ NE . then output;
proc sort data=MyLib.TermDocMatrix2;
by _Document_ Term;


data MyLib.Sentiment;
retain docScore ndoc;
keep _DOCUMENT_ ndoc docScore stars;
set MyLib.TermDocMatrix2;
by _DOCUMENT_ term;
if first._DOCUMENT_ then do;
docscore=count*score;
ndoc=count;
end;
else do;
docscore=docscore + count*score;
ndoc = ndoc + count;
end;
if last._document_ then do;
if ndoc>0 then docscore=docscore/ndoc;
else docscore=0;
if docscore NE . then stars = 3 + (4/6)*docscore;
output;
end;
```

```
DATA MyLib.Sentiment;
RETAIN DOC 0 nsave DocscoreSave StarsSave DocSave;
Keep _DOCUMENT_ ndoc Docscore Stars;
SET MyLib.Sentiment;
Doc = Doc+1;
if Doc LT _DOCUMENT_ THEN DO;
nsave = ndoc; DOCscoreSave = DocScore;
StarsSave = Stars;
DocSave= _Document_;
DO WHILE (Doc LT DocSave);
ndoc = 0; Docscore = 0; Stars = 3; _Document_ = Doc;
OUTPUT;
DOC = DOC  + 1;
END;
ndoc=nsave; Docscore=DocScoreSave; Stars=StarsSave;
_DOCUMENT_=DocSave;
END;
IF DOCSCORE EQ . THEN DO;
ndoc=0; Docscore=0; Stars=3;
END;
OUTPUT;

proc means data=MyLib.sentiment;
var docscore;

run;
```

MERGE Node Properties

Name	Merge Role	Overwrite Variable	Role	Level
DOCUMENT	**By**	**Default**	Input	Interval
docScore	**none**	**Default**	Input	Interval
ndoc	**none**	**Default**	Input	Interval
stars	**none**	**Default**	Input	Interval

METADATA Properties

Name	Hidden	Hide	Role	New Role	Level
IMP_REP_mile	N	**Default**	Input	**Default**	Interval
Year	N	**Default**	Input	**Default**	Interval
TextCluster_pr	N	**Default**	Input	**Default**	Interval
abs	N	**Default**	Input	**Default**	Nominal
TextCluster_pr	N	**Default**	Input	**Default**	Interval
TextCluster_pr	N	**Default**	Input	**Default**	Interval
ndoc	N	**Default**	Input	**Default**	Interval
docScore	N	**Default**	Input	**Default**	Interval
stars	N	**Default**	Input	**Default**	Interval
cruise	N	**Default**	Input	**Default**	Nominal
description	N	**Default**	Text	**Default**	Nominal
IMP_REP_mph	N	**Default**	Input	**Default**	Interval
TextCluster_cl	N	**Default**	Input	**Default**	Interval
TextCluster_pr	N	**Default**	Input	**Default**	Interval
Make	N	**Default**	Input	**Default**	Nominal
Model	N	**Default**	Input	**Default**	Nominal
TextCluster_pr	N	**Default**	Input	**Default**	Interval
TextCluster_pr	N	**Default**	Input	**Default**	Interval
TextCluster_pr	N	**Default**	Input	**Default**	Interval
DOCUMENT	N	**Default**	Input	**ID**	Interval
NhtsaID	N	**Default**	ID	**ID**	Interval
crash	N	**Default**	Input	**Target**	Nominal

DECISION TREE Properties

Property	Value
General	
Node ID	Tree3
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
Interactive	...
Import Tree Model	No
Tree Model Data Set	...
Use Frozen Tree	No
Use Multiple Targets	No
Splitting Rule	
Interval Target Criterion	ProbF
Nominal Target Criterion	Gini
Ordinal Target Criterion	Entropy
Significance Level	0.2
Missing Values	Use in search
Use Input Once	No
Maximum Branch	2
Maximum Depth	20
Minimum Categorical Size	3
Node	
Leaf Size	3
Number of Rules	5
Number of Surrogate Rules	0
Split Size	.