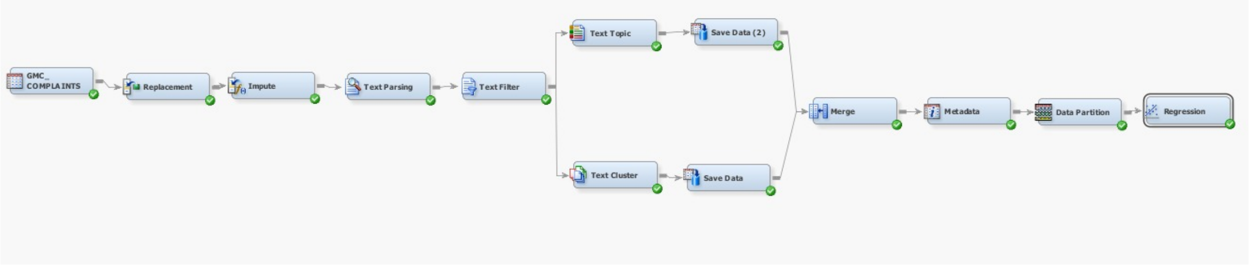
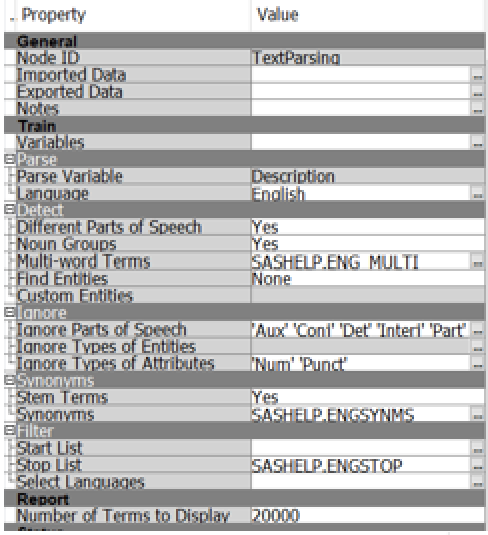
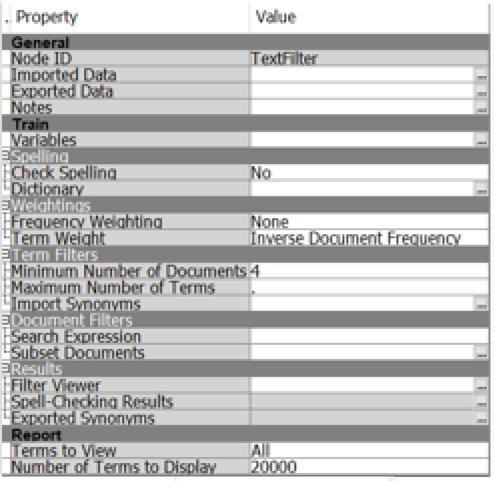
SAS Part

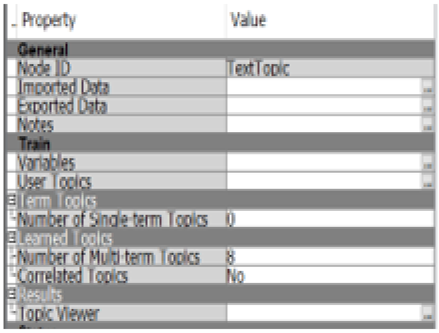




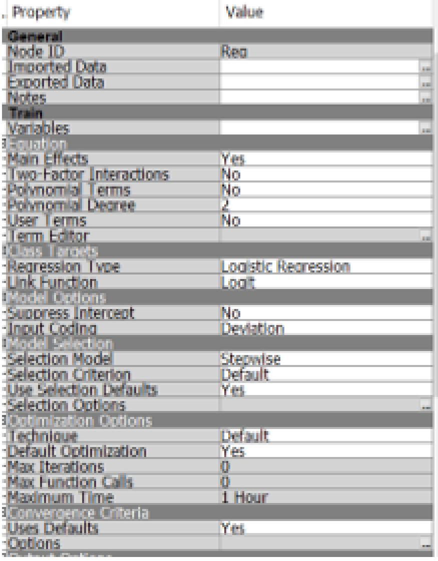
Property	Value
General	
Node ID	TextParsing
Imported Data	
Exported Data	
Notes	
Train	
Variables	
Parse	
Parse Variable	Description
Language	English
Detect	
Different Parts of Speech	Yes
Noun Groups	Yes
Multi-word Terms	SASHELP.ENG_MULTI
Find Entities	None
Custom Entities	
Ignore	
Ignore Parts of Speech	'Aux' 'Conj' 'Det' 'Interj' 'Part'
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



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 Documents	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

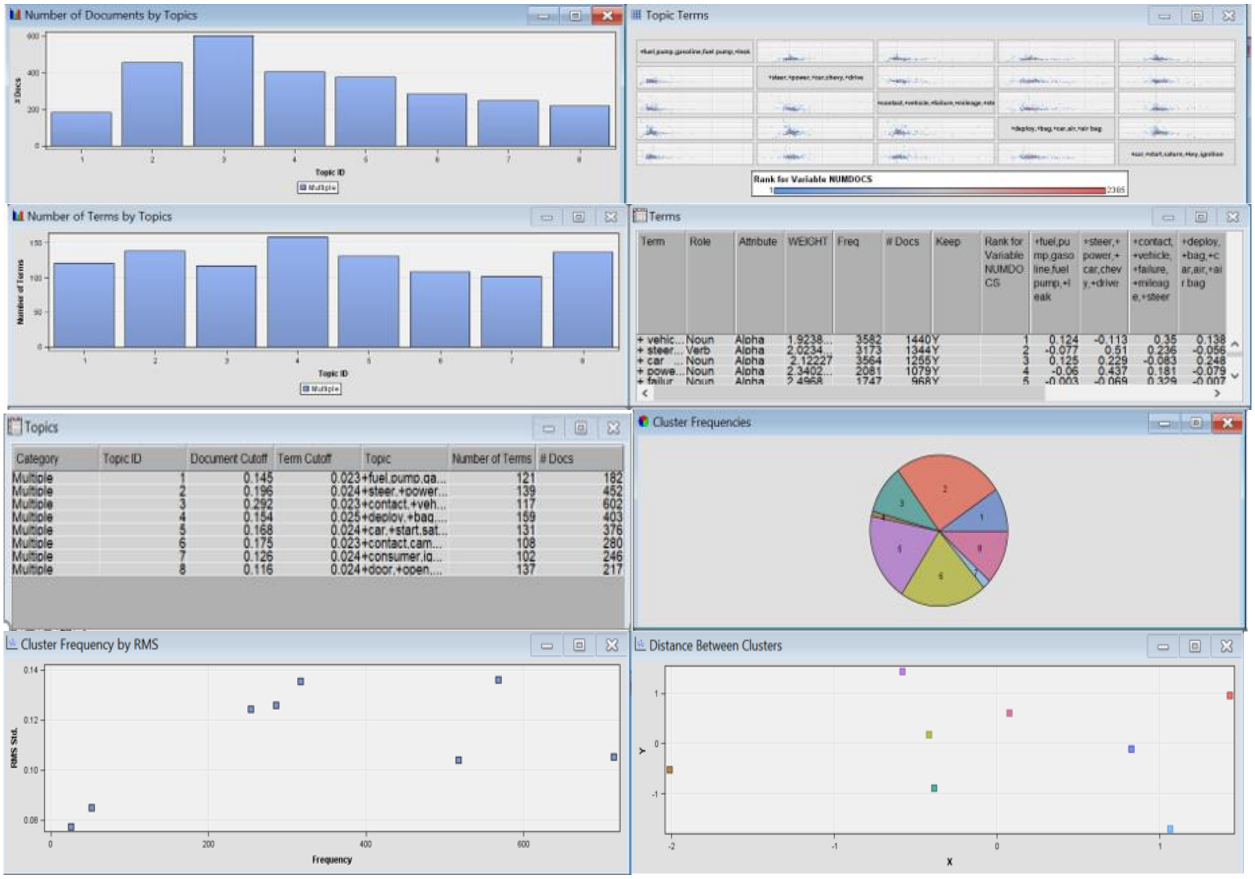


Property	Value
General	
Node ID	TextTopic
Imported Data	
Exported Data	
Notes	
Train	
Variables	
User Topics	
Term Topics	
Number of Single-term Topics	0
Learned Topics	
Number of Multi-term Topics	8
Correlated Topics	No
Results	
Topic Viewer	



Property	Value
General	
Node ID	Reg
Imported Data	
Exported Data	
Notes	
Train	
Variables	
Equation	
Main Effects	Yes
Two-Factor Interactions	No
Polynomial Terms	No
Polynomial Degree	2
User Terms	No
Term Editor	
Class Targets	
Regression Type	Logistic Regression
Link Function	Logit
Model Options	
Suppress Intercept	No
Input Coding	Deviation
Model Selection	
Selection Model	Stepwise
Selection Criterion	Default
Use Selection Defaults	Yes
Selection Options	
Optimization Options	
Technique	Default
Default Optimization	Yes
Max Iterations	0
Max Function Calls	0
Maximum Time	1 Hour
Convergence Criteria	
Uses Defaults	Yes
Options	

Question 2


Number of Documents by Topics
600
400
200
0
# Docs
1
2
3
4
5
6
7
8
Topic ID
Multiple
Topic Terms
+fuel,pump,gasoline,fuel pump,+leak
+steer,+power,+car,chevy,+drive
+contact,+vehicle,+failure,+mileage,+ste
+deploy,+bag,+car,air,+air bag
+car,+start,+stall,+key,ignition
Rank for Variable NUMDOCS
1
2385
Number of Terms by Topics
150
100
50
0
Number of Terms
1
2
3
4
5
6
7
8
Topic ID
Multiple
Cluster Frequencies
1
2
3
4
5
6
7
8
Cluster Frequency by RMS
0.14
0.12
0.10
0.08
RMS Std.
0
200
400
600
Frequency
Distance Between Clusters
1
0
-1
Y
-2
-1
0
1
X

Topics

Category	Topic ID	Document Cutoff	Term Cutoff	Topic	Number of Terms	# Docs
Multiple	1	0.145	0.023	+fuel,pump,ga...	121	182
Multiple	2	0.196	0.024	+steer,+power...	139	452
Multiple	3	0.292	0.023	+contact,+veh...	117	602
Multiple	4	0.154	0.025	+deploy,+bag,...	159	403
Multiple	5	0.168	0.024	+car,+start,sat...	131	376
Multiple	6	0.175	0.023	+contact,cam...	108	280
Multiple	7	0.126	0.024	+consumer,ig...	102	246
Multiple	8	0.116	0.024	+door,+open,...	137	217

Terms

Term	Role	Attribute	WEIGHT	Freq	# Docs	Keep	Rank for Variable NUMDOCS	+fuel,pump,gasoline,fuel pump,+leak	+steer,+power,+car,chevy,+drive	+contact,+vehicle,+failure,+mileage,+steer	+deploy,+bag,+car,air,+air bag
+ vehic...	Noun	Alpha	1.9238...	3582	1440	Y	1	0.124	-0.113	0.35	0.138
+ steer...	Verb	Alpha	2.0234...	3173	1344	Y	2	-0.077	0.51	0.236	-0.056
+ car ...	Noun	Alpha	2.12227	3564	1255	Y	3	0.125	0.229	-0.083	0.248
+ powe...	Noun	Alpha	2.3402...	2081	1079	Y	4	-0.06	0.437	0.181	-0.079
+ failur	Noun	Alpha	2.4968	1747	968	Y	5	-0.003	-0.069	0.329	-0.007

Question 3

		Predicted	
		Negative	Positive
	Negative	564	37
Actual	Positive	65	155

Question 4

Accuracy	87.6
Precision	80.7
Recall	70.5
F1	75.2

Python Part

```
import pandas as pd
import string
import nltk
import numpy as np

from AdvancedAnalytics import ReplaceImputeEncode
from AdvancedAnalytics import logreg
from sklearn.model_selection import cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

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 TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.decomposition import LatentDirichletAllocation

#Analyzer function
def fn_analyzer(s):
    syns = {'veh': 'vehicle', 'car': 'vehicle', 'chev':'cheverolet', \
              'chevy':'cheverolet', 'air bag': 'airbag', \
              'seat belt':'seatbelt', "n't":'not', 'to30':'to 30', \
              'wont':'would not', 'cant':'can not', 'cannot':'can not', \
              'couldnt':'could not', 'shouldnt':'should not', \
              'wouldnt':'would not', 'air':'airbag', 'bag':'airbag'}

    s = s.lower()
    s = s.replace(',', '. ')

#Tokenize_word
    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")]

#Mapping the synonyms
    for i in range(len(tokens)):
        if tokens[i] in syns:
            tokens[i] = syns[tokens[i]]
```

```
#stop words removal
    punctuation = list(string.punctuation)+['..', '...']
    pronouns = ['i', 'he', 'she', 'it', 'him', 'they', 'we', 'us', 'them']
#To add extra stop words
    other =
['own','go','get','seem','say','would','regard','report','involve'\
             ,'do','anoth','consumer',"'ve",'happen','try','either','come',]
    stop = stopwords.words('english') + punctuation + pronouns + other
    filter_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())]

# Stemming
    tag_words  = pos_tag(filter_terms, lang='eng')
    stemmer = SnowballStemmer("english")
    wn_tags = {'N':wn.NOUN, 'J':wn.ADJ, 'V':wn.VERB, 'R':wn.ADV}
    wnl = WordNetLemmatizer()
    stemm_tokens = []
    for tagged_token in tag_words :
        term = tagged_token[0]
        pos  = tagged_token[1]
        pos  = pos[0]
        try:
            pos   = wn_tags[pos]
            stemm_tokens.append(wnl.lemmatize(term, pos=pos))
        except:
            stemm_tokens.append(stemmer.stem(term))
    return stemm_tokens

#NLTK for Stop and Stem
def fn_preprocessor(s):
 #Vectorizer
    s = s.lower()
    s = s.replace(',', '. ')
    print("preprocessor")
    return(s)

def fn_tokenizer(s):
    # Tokenize
    print("Tokenizer")
    tokens = word_tokenize(s)
    tokens = [word.replace(',','') for word in tokens ]
    tokens = [word for word in tokens if word.find('*')!=True and \
              word != "''" and word !="``" and word!='description' \
              and word !='dtype']
    return tokens

#column width increased
pd.set_option('max_colwidth', 32000)
```

```
df = pd.read_excel("GMC_Complaints.xlsx")

num_docs     = len(df['description'])
num_samples  = num_docs
max_features = 1000
s_words    = 'english'
ngram = (1,2)
max_df=0.8

discussions = []
for i in range(num_samples):
    discussions.append(("%s" %df['description'].iloc[i]))


#Creating Word Frequency

cv = CountVectorizer(max_df=max_df, min_df=2, max_features=max_features,\
                     analyzer=fn_analyzer, ngram_range=ngram)
tf = cv.fit_transform(discussions)

num_topics        = 8
max_iteration        =  5
learning_offset = 20.
learning_method = 'online'

tf_idf = TfidfTransformer()
print("\nTF-IDF Parameters\n", tf_idf.get_params(),"\n")
tf_idf = tf_idf.fit_transform(tf)

tf_idf_vect = TfidfVectorizer(max_df=max_df, min_df=2,
max_features=max_features, analyzer=fn_analyzer, ngram_range=ngram)
tf_idf = tf_idf_vect.fit_transform(discussions)
print("\nTF_IDF Vectorizer Parameters\n", tf_idf_vect, "\n")

lda = LatentDirichletAllocation(n_components=num_topics,
max_iteration=max_iteration,\
                                learning_method=learning_method, \
                                learning_offset=learning_offset, \
                                random_state=12345)
lda.fit_transform(tf_idf)
print('{:.<22s}{:>6d}'.format("Number of Reviews", tf.shape[0]))
print('{:.<22s}{:>6d}'.format("Number of Terms",     tf.shape[1]))
print("\nTopics Identified using LDA with TF_IDF")
tf_features = cv.get_feature_names()
max_words = 15
desc = []
for topic_idx, topic in enumerate(lda.components_):
        message = "Topic #%d: " % topic_idx
        message += " ".join([tf_features[i]
                            for i in topic.argsort()[:-max_words - 1:-1]])
        print(message)
        print()
        desc.append([tf_features[i] for i in topic.argsort()[:-max_words -
1:-1]])

#Extracting probabilities of topic
topics = pd.DataFrame(lda.fit_transform(tf_idf))
preds = ['Year', 'make','model','crashed','abs','mileage']
df2 = pd.concat([df[preds],topics], axis=1, ignore_index=True)
df2.columns = ['Year',
'make','model','crashed','abs','mileage','0','1','2','3','4','5','6','7']

#Logistic Regression model

attribute_map = {
        'Year'    :['I',(2003,2011),[0,0]],
        'make'    :['N',('CHEVROLET','PONTIAC','SATURN'),[0,0]],
        'model'   :['N',('COBALT','G5','HHR','ION','SKY','SOLSTICE'),[0,0]],
        'crashed' :['B',('N','Y'),[0,0]],
        'abs'     :['B',('N','Y'),[0,0]],
        'mileage' :['I',(0,200000),[0,0]],
        '0'       :['I',(0,1),[0,0]],
        '1'       :['I',(0,1),[0,0]],
        '2'       :['I',(0,1),[0,0]],
        '3'       :['I',(0,1),[0,0]],
        '4'       :['I',(0,1),[0,0]],
        '5'       :['I',(0,1),[0,0]],
        '6'       :['I',(0,1),[0,0]],
        '7'       :['I',(0,1),[0,0]],
}
varlist = ['crashed']
rie = ReplaceImputeEncode(data_map=attribute_map, \
                               nominal_encoding='one-hot',
                          interval_scale = None, drop=False, display=False)
encoded_df = rie.fit_transform(df2)
X = encoded_df.drop(varlist, axis=1)
y = encoded_df[varlist]
np_y=np.ravel(y)

max_f1 = 0
List=[.1,1,10,100]
score_list = ['accuracy', 'recall', 'precision', 'f1']
for c in List:
    print("\nRegularization Parameter: ", c)
    lgr = LogisticRegression(C=c, tol=1e-8, max_iteration=1000)
    lgr.fit(X, np_y)
    scores = cross_validate(lgr, X, np_y,\
                            scoring=score_list, return_train_score=False, \
                            cv=10)
    print("{:.<13s}{:>6s}{:>13s}".format("Metric", "Mean", "Std. Dev."))
    for s in score_list:
        var = "test_"+s
        mean = scores[var].mean()
        std  = scores[var].std()
        print("{:.<13s}{:>7.4f}{:>10.4f}".format(s, mean, std))
        if mean > max_f1:
            max_f1 = mean
            best_predictor  = c
print("\nBest based on F1-Score")
print("Best Regularization Parameter = ", best_predictor)

X_train, X_valid, y_train, y_valid= \
train_test_split(X,y,test_size = 0.3, random_state=7)

np_y_train = np.ravel(y_train)
np_y_valid = np.ravel(y_valid)

lr = LogisticRegression(C=best_predictor, tol=1e-8, max_iteration=1000)
lr.fit(X_train,np_y_train)

logreg.display_coef(reg,21,2,X_train.columns)

logreg.display_binary_split_metrics(reg,X_train,np_y_train,X_valid,np_y_valid
)
```

Outputs

Metrics

```
Accuracy...............          0.8191       0.7881
Precision.............           0.8818       0.8554
Recall (Sensitivity)...          0.3573       0.3047
F1-score...............          0.5085       0.4494
MISC (Misclassification)...      18.1%        21.2%
    class 0..............        1.7%         2.0%
    class 1..............        64.3%        69.5%
```

Confusion Matrix

```
Validation
Confusion Matrix Class 0   Class 1
Class 0.....        576       12
Class 1.....        162       71
```