In [1]:
```
import pandas as pd
import numpy as np
import csv

import scipy.stats as scs
import statsmodels.api as sm
import statsmodels.formula.api as sms
import scipy.stats as stats

from math import sqrt

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, confusion_
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, Grad
from sklearn.naive_bayes import BernoulliNB, CategoricalNB, GaussianNB, Multi
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import classification_report, confusion_matrix, plot_cor
from sklearn.base import BaseEstimator
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.pipeline import Pipeline
from applesauce import model_opt, transform_df, model_scoring, cost_benefit_a

import matplotlib.pyplot as plt
import seaborn as sns
pd.options.display.float_format = '{:.2f}'.format
```

Purpose: Create a model that be able to take in data for a c predict whether or not there was a fatality in the accident.

In [2]:
```
df = pd.read_csv(r'data/ChicagoCrashes.csv')
```

In [3]:
```
df.describe()
```

	Unnamed: 0	CRASH_DATE_x	OCCUPANT_CNT	POSTED_SPEED_LIMIT	BEAT_OF_
count	567454.00	567454.00	567454.00	567454.00	567454.00
mean	990211.98	2018.06	1.41	28.89	1233.98
std	594777.08	1.28	1.41	5.92	699.58
min	1.00	2015.00	0.00	0.00	111.00
25%	475494.25	2017.00	1.00	30.00	725.00
50%	965871.50	2018.00	1.00	30.00	1212.00
75%	1494691.25	2019.00	2.00	30.00	1821.00
max	2115933.00	2020.00	60.00	99.00	2535.00

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 567454 entries, 0 to 567453
Data columns (total 49 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   Unnamed: 0                      567454 non-null  int64  
 1   CRASH_DATE_x                    567454 non-null  int64  
 2   UNIT_TYPE                       567454 non-null  object 
 3   MAKE                            567454 non-null  object 
 4   MODEL                           567454 non-null  object 
 5   VEHICLE_DEFECT                  567454 non-null  object 
 6   VEHICLE_TYPE                    567454 non-null  object 
 7   VEHICLE_USE                     567454 non-null  object 
 8   MANEUVER                        567454 non-null  object 
 9   OCCUPANT_CNT                    567454 non-null  float64
 10  CRASH_DATE_y                    567454 non-null  object 
 11  POSTED_SPEED_LIMIT              567454 non-null  int64  
 12  TRAFFIC_CONTROL_DEVICE          567454 non-null  object 
 13  DEVICE_CONDITION                567454 non-null  object 
 14  WEATHER_CONDITION               567454 non-null  object 
 15  LIGHTING_CONDITION              567454 non-null  object 
 16  FIRST_CRASH_TYPE                567454 non-null  object 
 17  TRAFFICWAY_TYPE                 567454 non-null  object 
 18  ALIGNMENT                       567454 non-null  object 
 19  ROADWAY_SURFACE_COND            567454 non-null  object 
 20  ROAD_DEFECT                     567454 non-null  object 
 21  REPORT_TYPE                     567454 non-null  object 
 22  CRASH_TYPE                      567454 non-null  object 
 23  DAMAGE                          567454 non-null  object 
 24  PRIM_CONTRIBUTORY_CAUSE         567454 non-null  object 
 25  SEC_CONTRIBUTORY_CAUSE          567454 non-null  object 
 26  BEAT_OF_OCCURRENCE              567454 non-null  float64
 27  NUM_UNITS                       567454 non-null  int64  
 28  MOST_SEVERE_INJURY              567454 non-null  object 
 29  INJURIES_TOTAL                  567454 non-null  float64
 30  INJURIES_FATAL                  567454 non-null  float64
 31  INJURIES_INCAPACITATING         567454 non-null  float64
 32  INJURIES_NON_INCAPACITATING     567454 non-null  float64
 33  INJURIES_REPORTED_NOT_EVIDENT   567454 non-null  float64
 34  INJURIES_NO_INDICATION          567454 non-null  float64
 35  INJURIES_UNKNOWN                567454 non-null  float64
 36  CRASH_HOUR                      567454 non-null  int64  
 37  CRASH_DAY_OF_WEEK               567454 non-null  int64  
 38  CRASH_MONTH                     567454 non-null  int64  
 39  LATITUDE                        567454 non-null  float64
 40  LONGITUDE                       567454 non-null  float64
 41  PERSON_ID                       567454 non-null  object 
 42  PERSON_TYPE                     567454 non-null  object 
 43  CRASH_DATE                      567454 non-null  object 
 44  SEX                             567454 non-null  object 
 45  SAFETY_EQUIPMENT                567454 non-null  object 
 46  AIRBAG_DEPLOYED                 567454 non-null  object 
 47  EJECTION                        567454 non-null  object 
```

```
 48  INJURY_CLASSIFICATION          567454 non-null  object
dtypes: float64(11), int64(7), object(31)
memory usage: 212.1+ MB
```

In [5]:
```
yes_no_converter = lambda x: 1 if x>=1 else 0
```

In [6]:
```
def transform_df(df): # this will create a binary encoding for fatalities in
    # 1 for a fatality was present
    # and 0 for no fatality present
    df['INJURIES_FATAL'] = df['INJURIES_FATAL'].apply(yes_no_converter)
    # df['y'] = df['y'].apply(yes_no_converter)
    return df

df = transform_df(df)
```

KNN is not good with large or wide datasets, let's choose a method.

In [7]:
```
df.head()
```

	Unnamed: 0	CRASH_DATE_x	UNIT_TYPE	MAKE	MODEL	VEHICLE_DEFECT
0	577317	2016	DRIVER	TOYOTA MOTOR COMPANY, LTD.	CAMRY	NONE
1	1612677	2019	DRIVER	BUICK	ENCLAVE	NONE
2	547332	2018	DRIVER	CHEVROLET	MALIBU (CHEVELLE)	NONE
3	756129	2018	DRIVER	HYUNDAI	Accent	NONE
4	95047	2017	DRIVER	CHEVROLET	MONTE CARLO	NONE

5 rows × 49 columns

Drop object models and objects like date that have too mar utilize get_dummies on the data set.

In [8]:
```
df = df.drop(columns=['Unnamed: 0', 'MAKE', 'MODEL', 'LATITUDE', 'LONGITUDE',
```

Dummify the data

In [9]:
```
df_dummies = pd.get_dummies(df)
```

Perform a train test split

In [10]:
```
df_train, df_valid = train_test_split(df_dummies, test_size=0.60)
```

In [11]:
```
display(df_train.info())
display(df_valid.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 226981 entries, 215211 to 79012
Columns: 319 entries, CRASH_DATE_x to EJECTION_UNKNOWN
dtypes: float64(1), int64(7), uint8(311)
memory usage: 82.9 MB
```

```
None
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 340473 entries, 293302 to 151384
Columns: 319 entries, CRASH_DATE_x to EJECTION_UNKNOWN
dtypes: float64(1), int64(7), uint8(311)
memory usage: 124.4 MB
```

```
None
```

Create training variables for the training data and the test d

In [12]:

```
df_train_X = df_train.drop(columns='INJURIES_FATAL')
df_train_y = df_train.loc[:,'INJURIES_FATAL']

X = df_train_X
y = df_train_y

data_full = df_train.groupby(by='INJURIES_FATAL').sum()
data_full.to_csv('crash_data.csv')
display(X.head())
display(y.head())
```

	CRASH_DATE_x	OCCUPANT_CNT	POSTED_SPEED_LIMIT	NUM_UNITS	CRASH_
215211	2019	2.00	30	3	11
53657	2019	1.00	35	3	21
43090	2017	1.00	30	2	7
22369	2019	2.00	30	2	10
258984	2018	5.00	35	3	8

5 rows × 318 columns

```
215211    0
53657     0
43090     0
22369     0
258984    0
Name: INJURIES_FATAL, dtype: int64
```

In [13]:
```
df_valid_X = df_valid.drop(columns='INJURIES_FATAL')
df_valid_y = df_valid.loc[:,'INJURIES_FATAL']

X_valid = df_valid_X
y_valid = df_valid_y

display(df_valid_X.head())
display(df_valid_y.head())
```

	CRASH_DATE_x	OCCUPANT_CNT	POSTED_SPEED_LIMIT	NUM_UNITS	CRASH_
293302	2017	2.00	45	3	9
159985	2018	1.00	30	2	18
357479	2016	3.00	30	2	7
376180	2017	0.00	30	2	21
54523	2019	3.00	30	2	18

5 rows × 318 columns

```
293302    0
159985    0
357479    0
376180    0
54523     0
Name: INJURIES_FATAL, dtype: int64
```

In [14]:
```
X.shape, y.shape, X_valid.shape, y_valid.shape
```

```
((226981, 318), (226981,), (340473, 318), (340473,))
```

In [15]:
```
df_train['INJURIES_FATAL'].value_counts(normalize=True)
```

```
0   1.00
1   0.00
Name: INJURIES_FATAL, dtype: float64
```

In [16]:
```
df_train.head()
```

	CRASH_DATE_x	OCCUPANT_CNT	POSTED_SPEED_LIMIT	NUM_UNITS	INJURIE
215211	2019	2.00	30	3	0
53657	2019	1.00	35	3	0
43090	2017	1.00	30	2	0
22369	2019	2.00	30	2	0
258984	2018	5.00	35	3	0

5 rows × 319 columns

Manually sample data based on target class in order to dea imbalance (~270 fatal accidents to over 560,000 non fatal a

In [17]:
```
from sklearn.utils import resample
```

In [18]:
```
df_majority = df_train.loc[df['INJURIES_FATAL']==0]
df_minority = df_train.loc[df['INJURIES_FATAL']==1]
```

In [19]:
```
df_minority.shape
```

```
(108, 319)
```

In [20]:
```
df_majority.shape
```

```
(226873, 319)
```

In [21]:
```
df_min_sample = resample(df_minority, replace=True, n_samples=1000, random_st
```

In [22]:
```
df_maj_sample = resample(df_majority, replace=True, n_samples=5000, random_st
```

In [23]:
```
num_cols = df_train.drop(columns=['INJURIES_FATAL']).columns
num_cols
```

```
Index(['CRASH_DATE_x', 'OCCUPANT_CNT', 'POSTED_SPEED_LIMIT', 'NUM_UNITS',
       'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
       'UNIT_TYPE_DISABLED VEHICLE', 'UNIT_TYPE_DRIVER',
       'UNIT_TYPE_DRIVERLESS',
       ...
       'AIRBAG_DEPLOYED_DEPLOYED, FRONT', 'AIRBAG_DEPLOYED_DEPLOYED, SIDE',
       'AIRBAG_DEPLOYED_DEPLOYMENT UNKNOWN', 'AIRBAG_DEPLOYED_DID NOT DEPLOY',
       'AIRBAG_DEPLOYED_NOT APPLICABLE', 'EJECTION_NONE',
       'EJECTION_PARTIALLY EJECTED', 'EJECTION_TOTALLY EJECTED',
       'EJECTION_TRAPPED/EXTRICATED', 'EJECTION_UNKNOWN'],
      dtype='object', length=318)
```

Create upsampled data and smote for full feature set

In [24]:
```
df_train_upsampled = pd.concat([df_min_sample, df_maj_sample], axis=0)
df_train_upsampled.shape
```

```
(6000, 319)
```

In [25]:
```
from imblearn.over_sampling import SMOTE
```

In [26]:
```
X_train, y_train = df_train_upsampled[num_cols], df_train_upsampled['INJURIES
```

In [27]:
```
smote=SMOTE()
```

In [28]:
```
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
X_train_smote.head()
```

	CRASH_DATE_x	OCCUPANT_CNT	POSTED_SPEED_LIMIT	NUM_UNITS	CRASH_HOU
0	2017	2.00	30	2	2
1	2020	1.00	30	7	22
2	2017	1.00	30	2	9
3	2019	3.00	15	1	6
4	2018	1.00	35	2	3

5 rows × 318 columns

Run a Random Forest model, fit train and test on SMOTE'd

In [29]:
```
clf = RandomForestClassifier(max_depth=5, min_samples_leaf=0.1, n_estimators=
```

In [30]:
```
clf.fit(X_train_smote, y_train_smote)
```

```
RandomForestClassifier(max_depth=5, min_samples_leaf=0.1, n_estimators=30)
```

In [31]:
```
clf.score(X_train, y_train)
```

```
0.8121666666666667
```

In [32]:
```
y_valid_pred = clf.predict(X_valid)
```

In [33]:
```
X_all, y_all = df_train[num_cols], df_train['INJURIES_FATAL']
```

In [34]:
```
clf.score(X_all, y_all)
```

```
0.7655486582577397
```

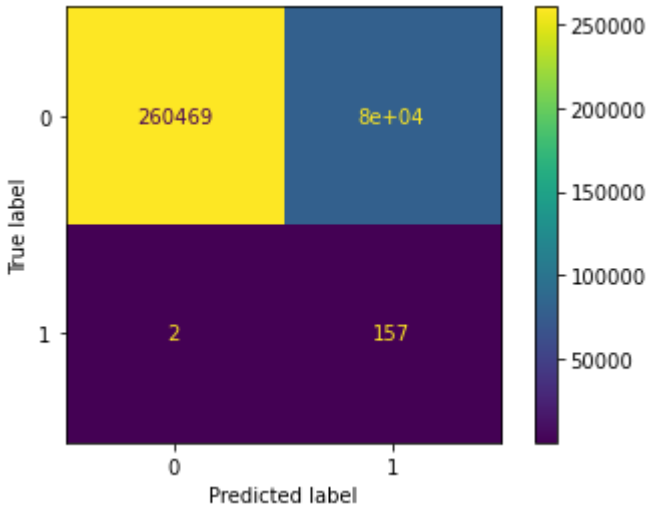
In [35]:
```
print(classification_report(y_valid, clf.predict(X_valid[num_cols])))
```

```
              precision    recall  f1-score   support

           0       1.00      0.77      0.87    340314
           1       0.00      0.99      0.00       159

    accuracy                           0.77    340473
   macro avg       0.50      0.88      0.44    340473
weighted avg       1.00      0.77      0.87    340473
```

In [36]:

```
score_report(y_valid, y_valid_pred)
plot_confusion_matrix(clf, X_valid[num_cols], y_valid)
plt.show()
```

```
Accuracy Score:  0.7654821380843708
Precision Score:  0.001962450938726532
Recall Score:  0.9874213836477987
F1 Score:  0.003917116802435099
```



Confusion matrix shows that the model is giving over 79000 false positive ir classifying a majority of actual deaths correctly.

In [37]:
```
print('''
      This is the total number of accidents with a fatality
      divided by the total number of accidents, expressed as
      a percent: ''',(df['INJURIES_FATAL'].sum()/df['INJURIES_FATAL'].count(
print('''
      This is the total number of accidents with a fatality: ''',df['INJURIES
print('''
      This is the total number of accidents: ''', df['INJURIES_FATAL'].count(
```

```
This is the total number of accidents with a fatality
divided by the total number of accidents, expressed as
a percent:  0.0470522720784416

This is the total number of accidents with a fatality:  267

This is the total number of accidents:  567454
```

Use select k best to limit number of features

In [38]:
```
selector = SelectKBest(score_func=chi2, k=10)
```

In [39]:
```
sel = selector.fit(X_train_smote, y_train_smote)
```

In [40]:
```
dfscores = sel.scores_
dfcols = X.columns

featscore = {}
for num in list(range(len(dfcols))):
    featscore[dfcols[num]] = round(dfscores[num], 2)

top_75 = {}
for item in sorted(featscore.items(), key=lambda x: x[1], reverse=True)[:75]
    top_75[item[0]] = item[1]
```

In [41]:
```
top_75 # most impactful features indicating a fatality is likely
```

```
{'NUM_UNITS': 1348.61,
 'OCCUPANT_CNT': 356.12,
 'CRASH_MONTH': 218.6,
 'POSTED_SPEED_LIMIT': 118.67,
 'CRASH_DAY_OF_WEEK': 81.38,
 'CRASH_DATE_x': 0.43,
 'UNIT_TYPE_DISABLED VEHICLE': nan,
 'UNIT_TYPE_DRIVERLESS': 171.37,
 'CRASH_HOUR': 112.3,
 'UNIT_TYPE_NON-CONTACT VEHICLE': nan,
 'UNIT_TYPE_PARKED': 38.56,
 'VEHICLE_DEFECT_BRAKES': 11.13,
 'VEHICLE_DEFECT_CARGO': nan,
 'VEHICLE_DEFECT_ENGINE/MOTOR': nan,
 'VEHICLE_DEFECT_EXHAUST': nan,
 'VEHICLE_DEFECT_FUEL SYSTEM': nan,
 'VEHICLE_DEFECT_LIGHTS': nan,
 'VEHICLE_DEFECT_RESTRAINT SYSTEM': nan,
 'VEHICLE_DEFECT_SIGNALS': nan,
 'VEHICLE_DEFECT_SUSPENSION': nan,
 'VEHICLE_DEFECT_TRAILER COUPLING': nan,
 'VEHICLE_TYPE_3-WHEELED MOTORCYCLE (2 REAR WHEELS)': nan,
 'VEHICLE_TYPE_BUS OVER 15 PASS.': 21.35,
 'VEHICLE_TYPE_FARM EQUIPMENT': nan,
 'VEHICLE_TYPE_MOPED OR MOTORIZED BICYCLE': nan,
 'VEHICLE_TYPE_MOTORCYCLE (OVER 150CC)': 44.8,
 'UNIT_TYPE_DRIVER': 21.1,
 'VEHICLE_TYPE_BUS UP TO 15 PASS.': 16.0,
 'VEHICLE_DEFECT_STEERING': 5.0,
 'VEHICLE_TYPE_OTHER': 4.05,
 'VEHICLE_TYPE_OTHER VEHICLE WITH TRAILER': 4.0,
 'VEHICLE_DEFECT_WHEELS': 3.0,
 'VEHICLE_DEFECT_TIRES': 2.0,
 'VEHICLE_TYPE_MOTOR DRIVEN CYCLE': 2.0,
 'VEHICLE_DEFECT_WINDOWS': 1.0,
 'VEHICLE_TYPE_ALL-TERRAIN VEHICLE (ATV)': 1.0,
 'VEHICLE_TYPE_AUTOCYCLE': 1.0,
 'VEHICLE_DEFECT_NONE': 0.04,
 'VEHICLE_TYPE_RECREATIONAL OFF-HIGHWAY VEHICLE (ROV)': nan,
 'VEHICLE_TYPE_SINGLE UNIT TRUCK WITH TRAILER': nan,
 'VEHICLE_TYPE_TRUCK - SINGLE UNIT': 88.0,
 'VEHICLE_USE_AGRICULTURE': nan,
 'VEHICLE_USE_CAMPER/RV - SINGLE UNIT': nan,
 'VEHICLE_USE_CAMPER/RV - TOWED/MULTI-UNIT': nan,
 'VEHICLE_USE_HOUSE TRAILER': nan,
 'VEHICLE_USE_NOT IN USE': 289.69,
 'MANEUVER_CHANGING LANES': 114.0,
 'VEHICLE_USE_TAXI/FOR HIRE': 106.0,
 'MANEUVER_BACKING': 75.72,
 'VEHICLE_USE_OTHER': 52.63,
 'VEHICLE_USE_CTA': 52.08,
 'VEHICLE_TYPE_SPORT UTILITY VEHICLE (SUV)': 48.93,
 'VEHICLE_USE_PERSONAL': 40.95,
```

```
 'VEHICLE_USE_COMMERCIAL - MULTI-UNIT': 31.02,
 'VEHICLE_USE_COMMERCIAL - SINGLE UNIT': 28.82,
 'VEHICLE_USE_SCHOOL BUS': 27.0,
 'VEHICLE_USE_OTHER TRANSIT': 25.0,
 'VEHICLE_TYPE_TRACTOR W/ SEMI-TRAILER': 24.64,
 'MANEUVER_AVOIDING VEHICLES/OBJECTS': 23.0,
 'VEHICLE_USE_CONSTRUCTION/MAINTENANCE': 22.0,
 'VEHICLE_USE_UNKNOWN/NA': 16.2,
 'VEHICLE_USE_TOW TRUCK': 13.56,
 'VEHICLE_USE_MASS TRANSIT': 11.0,
 'VEHICLE_TYPE_VAN/MINI-VAN': 10.2,
 'VEHICLE_USE_FIRE': 6.0,
 'VEHICLE_USE_AMBULANCE': 5.0,
 'VEHICLE_USE_STATE OWNED': 4.0,
 'VEHICLE_TYPE_PICKUP': 2.26,
 'VEHICLE_TYPE_TRACTOR W/O SEMI-TRAILER': 2.0,
 'VEHICLE_USE_RIDESHARE SERVICE': 1.71,
 'VEHICLE_TYPE_PASSENGER': 1.5,
 'VEHICLE_USE_DRIVER EDUCATION': 1.0,
 'VEHICLE_USE_LAWN CARE/LANDSCAPING': 1.0,
 'MANEUVER_DISABLED': nan,
 'MANEUVER_DIVERGING': 2.0}
```

In [42]:

```
# def getList(dict):
#     return dict.keys()

new_features = getList(top_75)
```

In [43]:

```
top75_features = list(new_features)
print(top75_features)
```

```
['NUM_UNITS', 'OCCUPANT_CNT', 'CRASH_MONTH', 'POSTED_SPEED_LIMIT', 'CRASH_DAY_OF_WEEK', 'CRASH_DA
E', 'UNIT_TYPE_DRIVERLESS', 'CRASH_HOUR', 'UNIT_TYPE_NON-CONTACT VEHICLE', 'UNIT_TYPE_PARKED', 'VE
ECT_CARGO', 'VEHICLE_DEFECT_ENGINE/MOTOR', 'VEHICLE_DEFECT_EXHAUST', 'VEHICLE_DEFECT_FUEL SYSTEM',
_DEFECT_RESTRAINT SYSTEM', 'VEHICLE_DEFECT_SIGNALS', 'VEHICLE_DEFECT_SUSPENSION', 'VEHICLE_DEFECT_
-WHEELED MOTORCYCLE (2 REAR WHEELS)', 'VEHICLE_TYPE_BUS OVER 15 PASS.', 'VEHICLE_TYPE_FARM EQUIPME
ZED BICYCLE', 'VEHICLE_TYPE_MOTORCYCLE (OVER 150CC)', 'UNIT_TYPE_DRIVER', 'VEHICLE_TYPE_BUS UP TO
G', 'VEHICLE_TYPE_OTHER', 'VEHICLE_TYPE_OTHER VEHICLE WITH TRAILER', 'VEHICLE_DEFECT_WHEELS', 'VEH
MOTOR DRIVEN CYCLE', 'VEHICLE_DEFECT_WINDOWS', 'VEHICLE_TYPE_ALL-TERRAIN VEHICLE (ATV)', 'VEHICLE_
ONE', 'VEHICLE_TYPE_RECREATIONAL OFF-HIGHWAY VEHICLE (ROV)', 'VEHICLE_TYPE_SINGLE UNIT TRUCK WITH
NGLE UNIT', 'VEHICLE_USE_AGRICULTURE', 'VEHICLE_USE_CAMPER/RV - SINGLE UNIT', 'VEHICLE_USE_CAMPER/
SE_HOUSE TRAILER', 'VEHICLE_USE_NOT IN USE', 'MANEUVER_CHANGING LANES', 'VEHICLE_USE_TAXI/FOR HIRE
E_OTHER', 'VEHICLE_USE_CTA', 'VEHICLE_TYPE_SPORT UTILITY VEHICLE (SUV)', 'VEHICLE_USE_PERSONAL',
T', 'VEHICLE_USE_COMMERCIAL - SINGLE UNIT', 'VEHICLE_USE_SCHOOL BUS', 'VEHICLE_USE_OTHER TRANSIT',
ILER', 'MANEUVER_AVOIDING VEHICLES/OBJECTS', 'VEHICLE_USE_CONSTRUCTION/MAINTENANCE', 'VEHICLE_USE_
K', 'VEHICLE_USE_MASS TRANSIT', 'VEHICLE_TYPE_VAN/MINI-VAN', 'VEHICLE_USE_FIRE', 'VEHICLE_USE_AMBU
'VEHICLE_TYPE_PICKUP', 'VEHICLE_TYPE_TRACTOR W/O SEMI-TRAILER', 'VEHICLE_USE_RIDESHARE SERVICE',
USE_DRIVER EDUCATION', 'VEHICLE_USE_LAWN CARE/LANDSCAPING', 'MANEUVER_DISABLED', 'MANEUVER_DIVERGI
```

Create SelectKBest features variable

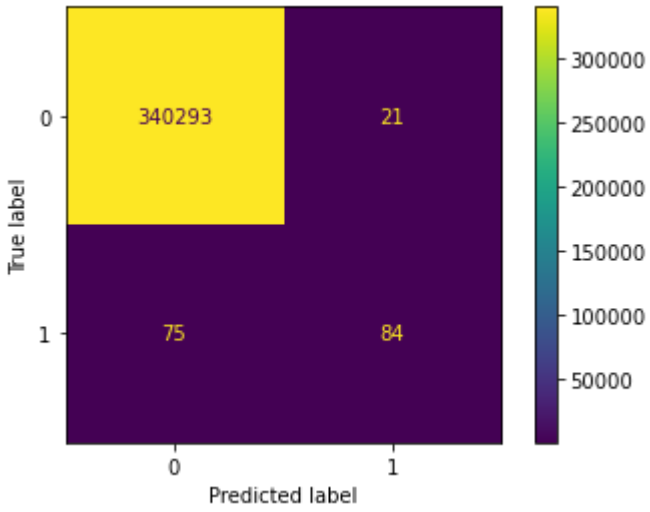
In [44]:
```
feature_list = top75_features
X2 = X_train_smote.loc[:,feature_list]
y2 = y_train_smote
# df_valid_X2 = df_valid[feature_list]
df_valid_X2 = df_valid_X.loc[:,feature_list]
```

Optimize the model using a for loop and model type

In [45]:
```
ran_for = RandomForestClassifier()
ada_clf = AdaBoostClassifier()
gb_clf = GradientBoostingClassifier()
gau_NB = GaussianNB()

models = [ran_for, ada_clf, gb_clf, gau_NB]
```

In [46]:
```
model_opt(models, X_train_smote, y_train_smote, X_valid, y_valid) # full data
# of all models run, the base RFC tree is the best most accurate and
# precise model for predicting whether or not a death occurred
```

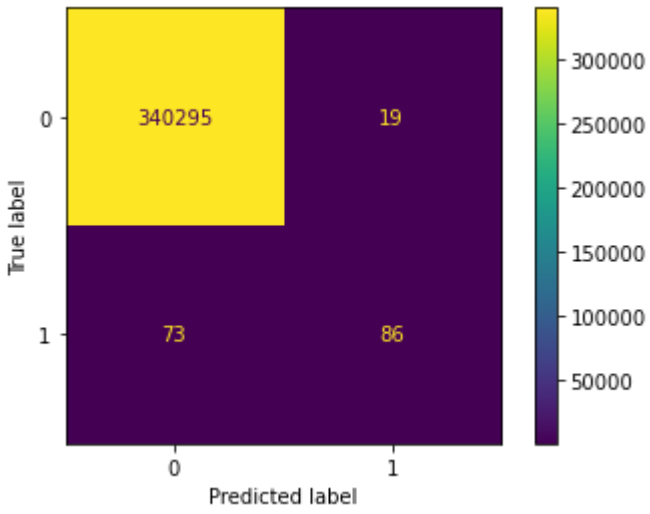
```
RandomForestClassifier()   0.9997180393158929
```



In [47]:
```
def single_model_opt(model, x, y, xtest, ytest):
    pipe = Pipeline(steps=[('model', model)])
    fit = pipe.fit(x, y)
    ypred = model.predict(xtest)
    score_report(ytest, ypred)
    print(model," ", fit.score(xtest, ytest))
    plot_confusion_matrix(model, xtest, ytest, values_format='1')
    plt.show()
    pass
```

In [48]:
```
single_model_opt(ran_for, X_train_smote, y_train_smote, X_valid, y_valid)
```

```
Accuracy Score:  0.9997297876777307
Precision Score:  0.819047619047619
Recall Score:  0.5408805031446541
F1 Score:  0.6515151515151515
RandomForestClassifier()   0.9997297876777307
```



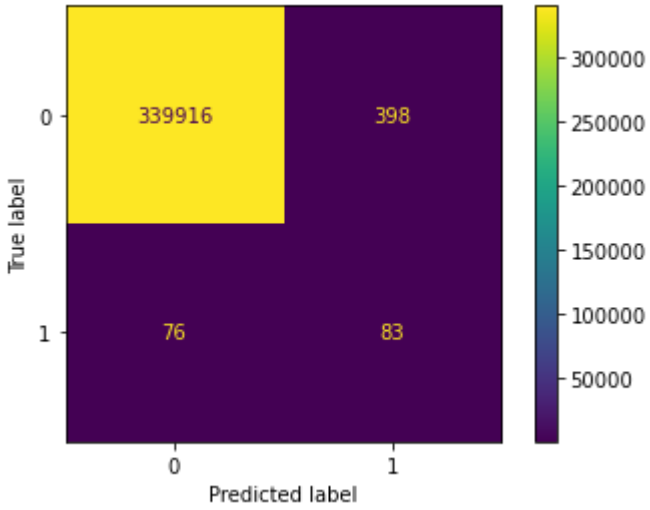
This model is the the most accurate and has a reasonable r
are less than 120 false positives out of over 340,000 negati

Additionally, recall of over 60% while is excellent in combina
accuracy.

Out of the above models, Random Forest performs the bes
applying feature selection if it continues to perform better th
models.

In [49]:
```
model_opt(models, X2, y2, df_valid_X2, y_valid) # this is the reduced feature
```

```
Accuracy Score:  0.9986078191222212
Precision Score:  0.17255717255717257
Recall Score:  0.5220125786163522
F1 Score:  0.259375
RandomForestClassifier()   0.9986078191222212
```



Random Forest performed worse with a r features.

In [50]:
```
# GridsearchCV and improving the full set Random Forest classifier with full
```

Tuning Hyper-parameters for a random forest model

In [51]:
```
# create single item lists for input to model_opt and also gridsearchCV.
selected_model = [ran_for]
```

In [52]:
```
for model in selected_model:
    print(model.get_params().keys())
```

```
dict_keys(['bootstrap', 'ccp_alpha', 'class_weight', 'criterion', 'max_depth', 'max_features', 'ma
_impurity_decrease', 'min_impurity_split', 'min_samples_leaf', 'min_samples_split', 'min_weight_fr
obs', 'oob_score', 'random_state', 'verbose', 'warm_start'])
```

In [53]:
```
# pipe_random = Pipeline([
#     ('select', SelectKBest()),
#     ('model', ran_for)])
# pipe_bayes = Pipeline([
#     ('select', SelectKBest()),
#     ('model', cat_bayes)])
```

In [54]:
```
ran_for = RandomForestClassifier()
cat_bayes = CategoricalNB()
param_random = {
    "max_depth": [3,5,7,11,15],
    "n_estimators": range(5,50,5),
    "max_leaf_nodes": range(3,10,2),
    "max_features": range(5,50,5)
    }
```

In [55]:
```
# need to improve on overall precision, so scoring for both models will be pr
gsforest = GridSearchCV(estimator=ran_for, param_grid=param_random, cv=5, sco
                        verbose=1, n_jobs=6)
```

In [56]:
```
# initial SMOTED training set
gsforest.fit(X_train_smote, y_train_smote)
```

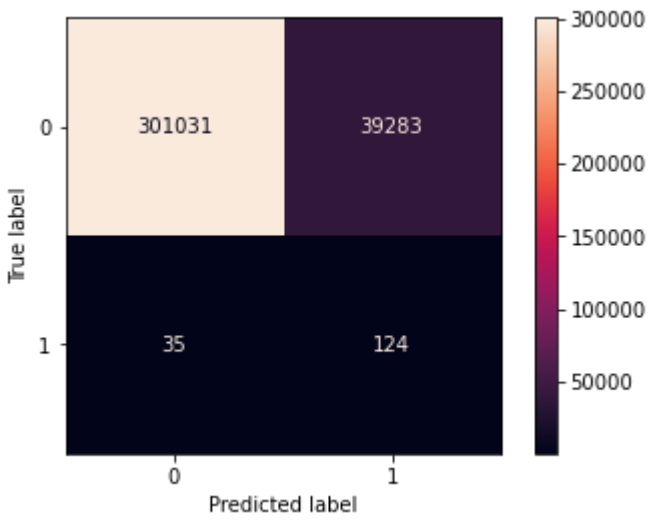
```
Fitting 5 folds for each of 1620 candidates, totalling 8100 fits


[Parallel(n_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.
[Parallel(n_jobs=6)]: Done  38 tasks      | elapsed:    3.1s
[Parallel(n_jobs=6)]: Done 188 tasks      | elapsed:    6.8s
[Parallel(n_jobs=6)]: Done 438 tasks      | elapsed:   14.3s
[Parallel(n_jobs=6)]: Done 788 tasks      | elapsed:   27.0s
[Parallel(n_jobs=6)]: Done 1238 tasks      | elapsed:   46.2s
[Parallel(n_jobs=6)]: Done 1788 tasks      | elapsed:  1.2min
[Parallel(n_jobs=6)]: Done 2438 tasks      | elapsed:  1.6min
[Parallel(n_jobs=6)]: Done 3188 tasks      | elapsed:  2.3min
[Parallel(n_jobs=6)]: Done 4038 tasks      | elapsed:  2.9min
[Parallel(n_jobs=6)]: Done 4988 tasks      | elapsed:  3.9min
[Parallel(n_jobs=6)]: Done 6038 tasks      | elapsed:  4.8min
[Parallel(n_jobs=6)]: Done 7188 tasks      | elapsed:  5.7min
[Parallel(n_jobs=6)]: Done 8100 out of 8100 | elapsed:  6.7min finished


GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=6,
             param_grid={'max_depth': [3, 5, 7, 11, 15],
                         'max_features': range(5, 50, 5),
                         'max_leaf_nodes': range(3, 10, 2),
                         'n_estimators': range(5, 50, 5)},
             scoring='precision', verbose=1)
```

In [57]:
```
y_preds = gsforest.predict(X_valid)
```

In [58]:
```
sum(y_preds)
```

```
39407
```

In [59]:
```
plot_confusion_matrix(gsforest, X_valid, y_valid, cmap='rocket', values_forma
plt.show()
```



In [60]:
```
gsforest_f1 = gsforest.best_estimator_
gsforest_f1
```

```
RandomForestClassifier(max_depth=7, max_features=5, max_leaf_nodes=9,
                       n_estimators=35)
```

In [61]:
```
print(gsforest_f1.score(X_train_smote, y_train_smote))
print(gsforest_f1.score(X_valid, y_valid)) # model is not overfit
```

```
0.8969
0.8845194773153818
```

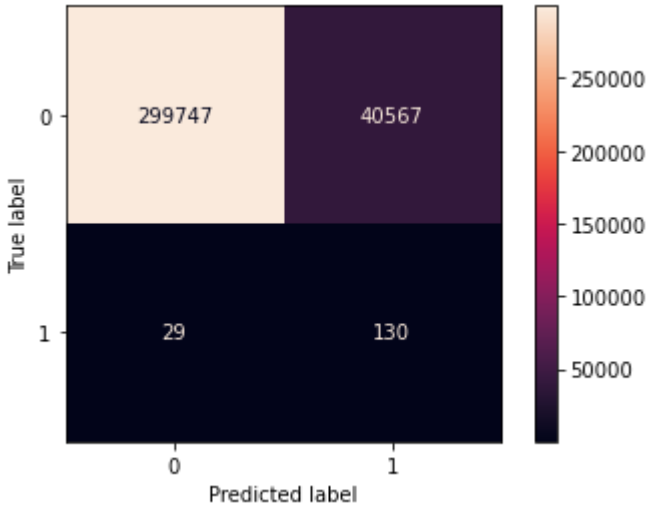
In [62]:
```
gsforest_f1.fit(X_train_smote, y_train_smote)
```

```
RandomForestClassifier(max_depth=7, max_features=5, max_leaf_nodes=9,
                       n_estimators=35)
```

In [63]:
```
y_preds = gsforest_f1.predict(X_valid)
sum(y_preds)
```

```
40697
```

In [64]:
```
plot_confusion_matrix(gsforest_f1, X_valid, y_valid, cmap='rocket', values_fo
plt.show()
```



In [65]:
```
gsforest.fit(X2, y2)
```

```
Fitting 5 folds for each of 1620 candidates, totalling 8100 fits


[Parallel(n_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.
[Parallel(n_jobs=6)]: Done  64 tasks      | elapsed:    2.0s
[Parallel(n_jobs=6)]: Done 364 tasks      | elapsed:   11.8s
[Parallel(n_jobs=6)]: Done 864 tasks      | elapsed:   31.6s
[Parallel(n_jobs=6)]: Done 1564 tasks      | elapsed:  1.1min
[Parallel(n_jobs=6)]: Done 2464 tasks      | elapsed:  1.7min
[Parallel(n_jobs=6)]: Done 3564 tasks      | elapsed:  2.6min
[Parallel(n_jobs=6)]: Done 4856 tasks      | elapsed:  3.8min
[Parallel(n_jobs=6)]: Done 5606 tasks      | elapsed:  4.2min
[Parallel(n_jobs=6)]: Done 6456 tasks      | elapsed:  5.1min
[Parallel(n_jobs=6)]: Done 7406 tasks      | elapsed:  5.8min
[Parallel(n_jobs=6)]: Done 8100 out of 8100 | elapsed:  6.6min finished


GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=6,
             param_grid={'max_depth': [3, 5, 7, 11, 15],
                         'max_features': range(5, 50, 5),
                         'max_leaf_nodes': range(3, 10, 2),
                         'n_estimators': range(5, 50, 5)},
             scoring='precision', verbose=1)
```

In [66]:
```
gsforest.best_estimator_
```

```
RandomForestClassifier(max_depth=5, max_features=15, max_leaf_nodes=9,
                       n_estimators=45)
```

In [67]:
```
gsforest_high_precision_partial = gsforest.best_estimator_
gsforest_high_precision_partial
```

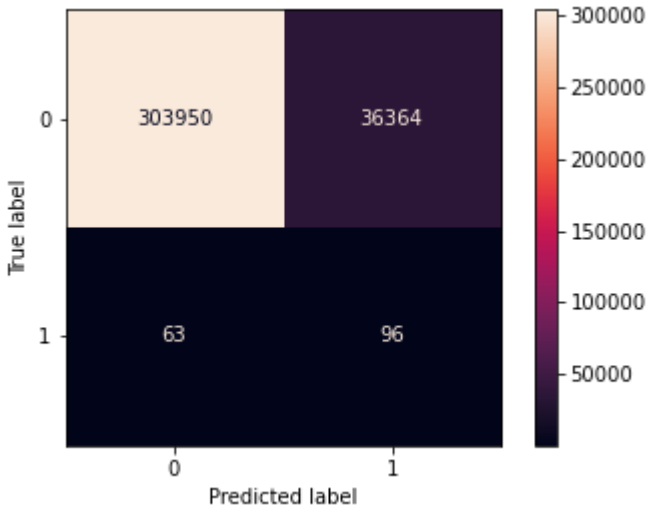
```
RandomForestClassifier(max_depth=5, max_features=15, max_leaf_nodes=9,
                       n_estimators=45)
```

In [68]:
```
gsforest_high_precision_partial.predict(X2)
```

```
array([1, 1, 0, ..., 1, 1, 0], dtype=int64)
```

In [69]:
```
y_preds2 = gsforest_high_precision_partial.predict(X2)
```

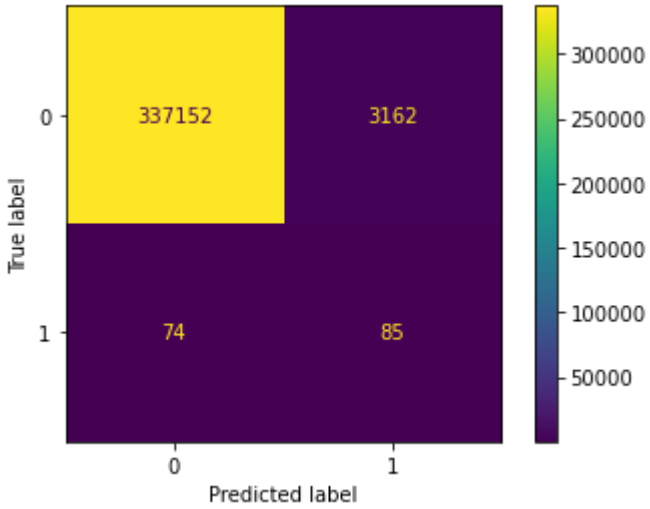
In [70]:
```
plot_confusion_matrix(gsforest_high_precision_partial, df_valid_X2, y_valid,
plt.show()
```



In [71]:
```
clf = RandomForestClassifier(n_estimators=50, random_state=42, max_features=4
clf.fit(X_train_smote, y_train_smote)
y_preds_clf = clf.predict(X_valid)
```

In [72]:
```
model_opt(clf, X_train_smote, y_train_smote, X_valid, y_valid)
plt.show()
```

```
Accuracy Score:  0.9904955752732229
Precision Score:  0.02617801047120419
Recall Score:  0.5345911949685535
F1 Score:  0.049911920140927775
DecisionTreeClassifier(max_features=40, random_state=1608637542)   0.9904955752732229
```



In [73]:
```
ran_for.fit(X_train_smote, y_train_smote)
```

```
RandomForestClassifier()
```

In [74]:
```
cost_benefit_analysis(ran_for, X_valid, y_valid)
```

```
-0.0024671559859372107
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: