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

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

from haversine import haversine
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, Grac
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

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

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

In [3]:
```
yes_no_converter = lambda x: 1 if x>=1 else 0
```

In [4]:
```
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 [5]:
```
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 [6]:
```
# target data, what we are trying to predict, should it be death ?
print(df['INJURIES_FATAL'].sum()) # 685 deaths out of this 1.1 million line d
```

```
331.0
```

In [7]:
```
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)
```

In [8]:
```
print(df['MOST_SEVERE_INJURY'].unique()) # what can we learn by attempting to
```

```
['NO INDICATION OF INJURY' 'NONINCAPACITATING INJURY'
 'REPORTED, NOT EVIDENT' 'INCAPACITATING INJURY' 'FATAL']
```

In [9]:
```
# KNN is not good with large or wide datasets, lets choose a different method
```

In [10]:
```
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

In [11]:
```
df = df.drop(columns=['Unnamed: 0', 'MAKE', 'MODEL', 'LATITUDE', 'LONGITUDE',
```

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 567454 entries, 0 to 567453
Data columns (total 31 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   CRASH_DATE_x            567454 non-null  int64  
 1   UNIT_TYPE               567454 non-null  object 
 2   VEHICLE_DEFECT          567454 non-null  object 
 3   VEHICLE_TYPE            567454 non-null  object 
 4   VEHICLE_USE             567454 non-null  object 
 5   MANEUVER                567454 non-null  object 
 6   OCCUPANT_CNT            567454 non-null  float64
 7   POSTED_SPEED_LIMIT      567454 non-null  int64  
 8   TRAFFIC_CONTROL_DEVICE  567454 non-null  object 
 9   DEVICE_CONDITION        567454 non-null  object 
 10  WEATHER_CONDITION       567454 non-null  object 
 11  LIGHTING_CONDITION      567454 non-null  object 
 12  FIRST_CRASH_TYPE        567454 non-null  object 
 13  TRAFFICWAY_TYPE         567454 non-null  object 
 14  ROADWAY_SURFACE_COND    567454 non-null  object 
 15  ROAD_DEFECT             567454 non-null  object 
 16  REPORT_TYPE             567454 non-null  object 
 17  CRASH_TYPE              567454 non-null  object 
 18  DAMAGE                  567454 non-null  object 
 19  PRIM_CONTRIBUTORY_CAUSE 567454 non-null  object 
 20  SEC_CONTRIBUTORY_CAUSE  567454 non-null  object 
 21  NUM_UNITS               567454 non-null  int64  
 22  INJURIES_FATAL          567454 non-null  int64  
 23  CRASH_HOUR              567454 non-null  int64  
 24  CRASH_DAY_OF_WEEK       567454 non-null  int64  
 25  CRASH_MONTH             567454 non-null  int64  
 26  PERSON_TYPE             567454 non-null  object 
 27  SEX                     567454 non-null  object 
 28  SAFETY_EQUIPMENT        567454 non-null  object 
 29  AIRBAG_DEPLOYED         567454 non-null  object 
 30  EJECTION                567454 non-null  object 
dtypes: float64(1), int64(7), object(23)
memory usage: 134.2+ MB
```

In [13]:
```
df_dummies = pd.get_dummies(df)
```

In [124]:
```
df['CRASH_DATE_x'].max()
```

```
2020
```

In [14]:
```
df_train, df_valid = train_test_split(df_dummies, test_size=0.30)
```

In [15]:
```
display(df_train.info())
display(df_valid.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 397217 entries, 196452 to 290726
Columns: 319 entries, CRASH_DATE_x to EJECTION_UNKNOWN
dtypes: float64(1), int64(7), uint8(311)
memory usage: 145.1 MB
```

```
None
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 170237 entries, 31363 to 108389
Columns: 319 entries, CRASH_DATE_x to EJECTION_UNKNOWN
dtypes: float64(1), int64(7), uint8(311)
memory usage: 62.2 MB
```

```
None
```

In [123]:
```
df_train_X = df_train.drop(columns='INJURIES_FATAL')
df_train_y = df_train['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')
```

In [17]:
```
cat_bayes = CategoricalNB()
```

In [18]:
```
cat_bayes.fit(X,y)
```

```
CategoricalNB()
```

In [19]:
```
df_valid_X = df_valid.drop(columns='INJURIES_FATAL')
df_valid_y = df_valid['INJURIES_FATAL']

X_valid = pd.get_dummies(df_valid_X)
y_valid = df_valid_y
```

In [20]:
```
X.shape, y.shape, X_valid.shape, y_valid.shape
```

```
((397217, 318), (397217,), (170237, 318), (170237,))
```

In [21]:
```
train_score1 = cat_bayes.score(X,y)
print(train_score1)
```

```
0.9901993117112309
```

In [22]:
```
test_score1 = cat_bayes.score(X_valid, y_valid)
print(test_score1)
```

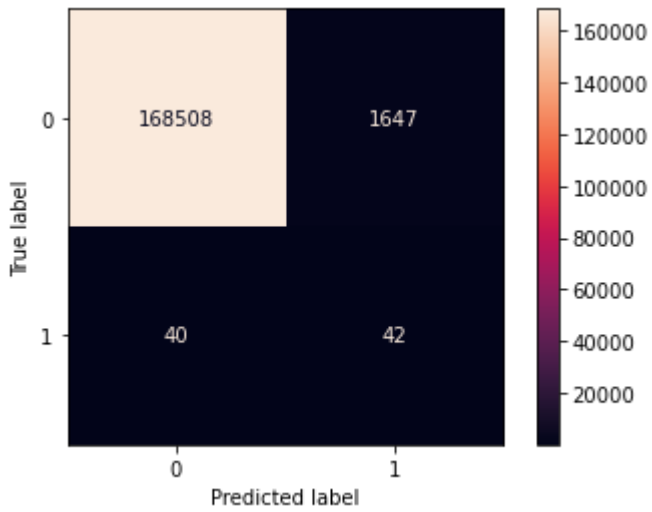
```
0.9900902858955456
```

In [23]:
```
if(train_score1 > test_score1):
    print('Model is overfit')
else:
    print('Model is not overfit')
```

```
Model is overfit
```

In [24]:
```
y_valid_pred = cat_bayes.predict(X_valid)
```

In [25]:
```
plot_confusion_matrix(cat_bayes, X_valid, y_valid, cmap='rocket', values_forr
plt.show()
```



In [26]:
```
def score_report(ytrue, ypred):
    print("Accuracy Score: ", accuracy_score(ytrue, ypred))
    print("Precision Score: ", precision_score(ytrue, ypred)) # a little dif
    print("Recall Score: ", recall_score(ytrue, ypred)) # recall score is he
    print("F1 Score: ", f1_score(ytrue, ypred))
    pass
```

In [27]:
```
score_report(y_valid, y_valid_pred)
```

```
Accuracy Score:  0.9900902858955456
Precision Score:  0.02486678507992895
Recall Score:  0.5121951219512195
F1 Score:  0.047430830039525695
```

In [28]:
```
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
```

In [29]:
```
# based on the above we can see that this model is missing the mark on attemp
# to predict where someone is likely to lose their life, missing on 16/43 act
# fatalities. it additionally misclassifies 1166 accidents where there was no
# but has indicated a false positive. The false positives are a problem in th
```

In [30]:
```
selector = SelectKBest(score_func=chi2, k=15)
```

In [31]:
```
sel = selector.fit(X,y)
```

In [32]:
```
dfscores = sel.scores_
dfcols = X.columns

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

top_18 = {}
for item in sorted(featscore.items(), key=lambda x: x[1], reverse=True)[:18]
    top_18[item[0]] = item[1]
```

In [33]:
```
top_18 # most impactful features indicating a fatality is likely
```

```
{'EJECTION_TRAPPED/EXTRICATED': 1713.88,
 'PRIM_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF DRIVER': 1156.52,
 'FIRST_CRASH_TYPE_PEDESTRIAN': 560.35,
 'AIRBAG_DEPLOYED_DEPLOYED, COMBINATION': 469.23,
 'EJECTION_TOTALLY EJECTED': 426.88,
 'PRIM_CONTRIBUTORY_CAUSE_EXCEEDING AUTHORIZED SPEED LIMIT': 398.23,
 'CRASH_TYPE_INJURY AND / OR TOW DUE TO CRASH': 374.3,
 'SAFETY_EQUIPMENT_HELMET NOT USED': 239.22,
 'VEHICLE_TYPE_MOTORCYCLE (OVER 150CC)': 168.14,
 'REPORT_TYPE_ON SCENE': 146.24,
 'SEC_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF DRIVER': 143.71,
 'CRASH_TYPE_NO INJURY / DRIVE AWAY': 135.81,
 'REPORT_TYPE_NOT ON SCENE (DESK REPORT)': 106.8,
 'NUM_UNITS': 100.46,
 'PRIM_CONTRIBUTORY_CAUSE_DISREGARDING TRAFFIC SIGNALS': 98.52,
 'FIRST_CRASH_TYPE_FIXED OBJECT': 70.11,
 'PRIM_CONTRIBUTORY_CAUSE_OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESSI
 'SEC_CONTRIBUTORY_CAUSE_DISREGARDING TRAFFIC SIGNALS': 68.15}
```

In [34]:
```
def getList(dict):
    return dict.keys()

new_features = getList(top_18)
```

In [71]:
```
top18_features = list(new_features)
print(top18_features)
```

```
['EJECTION_TRAPPED/EXTRICATED', 'PRIM_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF DRIVER', 'FIRST_CRA
YED_DEPLOYED, COMBINATION', 'EJECTION_TOTALLY EJECTED', 'PRIM_CONTRIBUTORY_CAUSE_EXCEEDING AUTHORI
RY AND / OR TOW DUE TO CRASH', 'SAFETY_EQUIPMENT_HELMET NOT USED', 'VEHICLE_TYPE_MOTORCYCLE (OVER
'SEC_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF DRIVER', 'CRASH_TYPE_NO INJURY / DRIVE AWAY', 'REPOR
T)', 'NUM_UNITS', 'PRIM_CONTRIBUTORY_CAUSE_DISREGARDING TRAFFIC SIGNALS', 'FIRST_CRASH_TYPE_FIXED
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESSIVE MANNER', 'SEC_CONTRIBUTO
ALS']
```

In [72]:
```
def feature_occurrence_count(dataframe, lst):
    for feat in lst:
        if
        print(feat)
        print(feat, " ", dataframe[feat].sum())
    return
```

In [119]:
```
feature_list = ['PRIM_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF DRIVER',
 'EJECTION_TRAPPED/EXTRICATED',
 'SAFETY_EQUIPMENT_HELMET NOT USED',
 'AIRBAG_DEPLOYED_DEPLOYED, COMBINATION',
 'FIRST_CRASH_TYPE_PEDESTRIAN',
 'CRASH_TYPE_INJURY AND / OR TOW DUE TO CRASH',
 'EJECTION_TOTALLY EJECTED',
 'PRIM_CONTRIBUTORY_CAUSE_DISREGARDING TRAFFIC SIGNALS',
 'VEHICLE_TYPE_MOTORCYCLE (OVER 150CC)',
 'PRIM_CONTRIBUTORY_CAUSE_EXCEEDING AUTHORIZED SPEED LIMIT',
 'SEC_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF DRIVER',
 'REPORT_TYPE_ON SCENE',
 'CRASH_TYPE_NO INJURY / DRIVE AWAY',
 'TRAFFIC_CONTROL_DEVICE_SCHOOL ZONE',
 'REPORT_TYPE_NOT ON SCENE (DESK REPORT)',
 'SEC_CONTRIBUTORY_CAUSE_EXCEEDING AUTHORIZED SPEED LIMIT',
 'NUM_UNITS',
 'TRAFFICWAY_TYPE_FOUR WAY']

X2 = df_train[feature_list]
y2 = df_train_y

data_investigate = df_train
data_investigate.sort_values(by='PRIM_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION (
data_investigate.info()
di = data_investigate.groupby(by='INJURIES_FATAL').sum()
di[feature_list].to_csv('vehicle_accident_data.csv')

df
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 397217 entries, 196452 to 290726
Columns: 319 entries, CRASH_DATE_x to EJECTION_UNKNOWN
dtypes: float64(1), int64(7), uint8(311)
memory usage: 145.1 MB
```

In [37]:
```
cat_bayes.fit(X2,y2)
```

```
CategoricalNB()
```

In [38]:
```
df_valid_X = df_valid[feature_list]
df_valid_y = df_valid['INJURIES_FATAL']
# top 15 features are being used in X_valid2
X_valid2 = df_valid_X
y_valid2 = df_valid_y
```

In [39]:
```
X2.shape, y2.shape, X_valid2.shape, y_valid2.shape
```

```
((397217, 18), (397217,), (170237, 18), (170237,))
```

In [40]:
```
train_score2 = cat_bayes.score(X2, y2)
print(train_score2)
```

```
0.9929887190125297
```

In [41]:
```
test_score2 = cat_bayes.score(X_valid2, y_valid2)
print(test_score2)
```

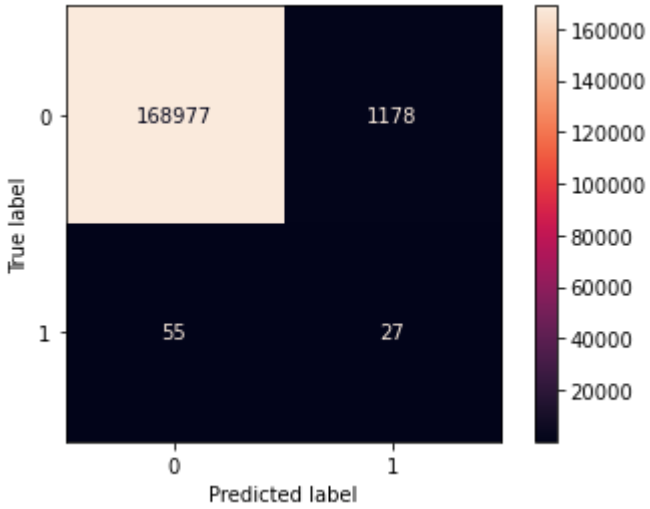
```
0.992757156199886
```

In [42]:
```
if(train_score2 > test_score2):
    print('Model is overfit')
else:
    print('Model is not overfit')
```

```
Model is overfit
```

In [43]:
```
y_valid_pred2 = cat_bayes.predict(X_valid2)
```

In [44]:
```
plot_confusion_matrix(cat_bayes, X_valid2, y_valid2, cmap='rocket', values_fo
plt.show()
```



Model Insights - an improvement from our first model in terr positives, but we also lost true positives and classified them negatives.

In [45]:
```
## optimize the model using a for loop and model type and a gridsearchCV
# BernoulliNB, CategoricalNB, GaussianNB, MultinomialNB
```

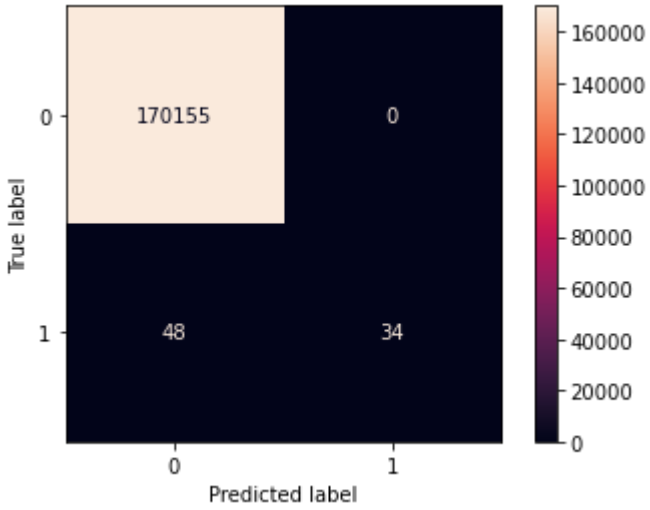
In [46]:
```
ran_for = RandomForestClassifier()
ada_clf = AdaBoostClassifier()
gb_clf = GradientBoostingClassifier()
gau_NB = GaussianNB()

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

In [47]:

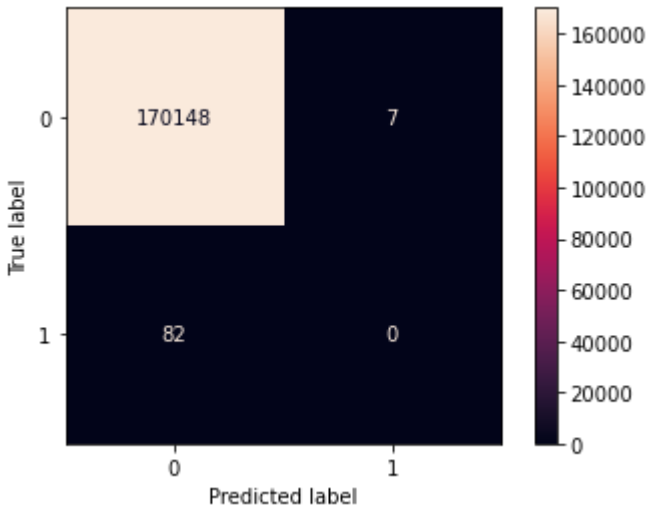
```
def model_opt(models, x, y, xtest, ytest):
    for model in models:
        pipe = Pipeline(steps=[('model', model)])
        fit = pipe.fit(x, y)
        ypred = model.predict(xtest)
        print(model," ", fit.score(x, y))
        plot_confusion_matrix(model, xtest, ytest, cmap='rocket', values_forr
        plt.show()
        score_report(ytest, ypred)
    pass
```

In [48]:
```
model_opt(models, X, y, X_valid, y_valid) # this is the full model dataset
```

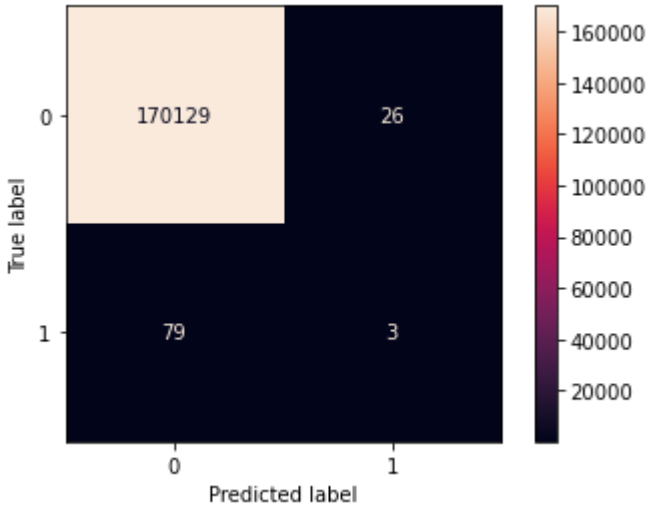
```
RandomForestClassifier()   0.9999974824843851
```



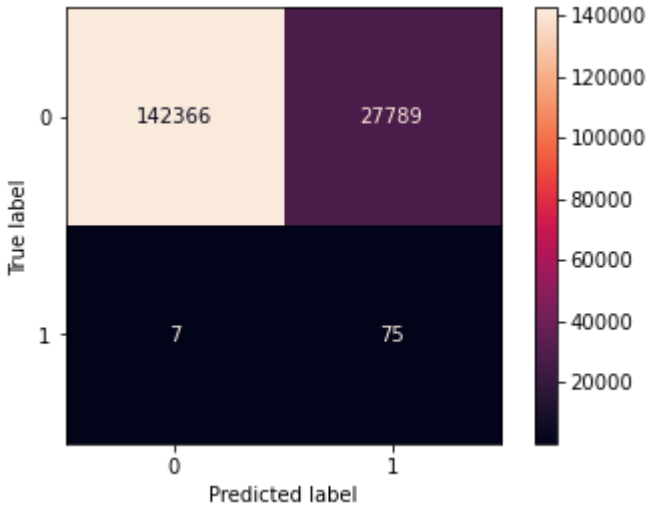

```
Accuracy Score:  0.9997180401440345
Precision Score:  1.0
Recall Score:  0.4146341463414634
F1 Score:  0.5862068965517241
AdaBoostClassifier()   0.9995216720331708
```



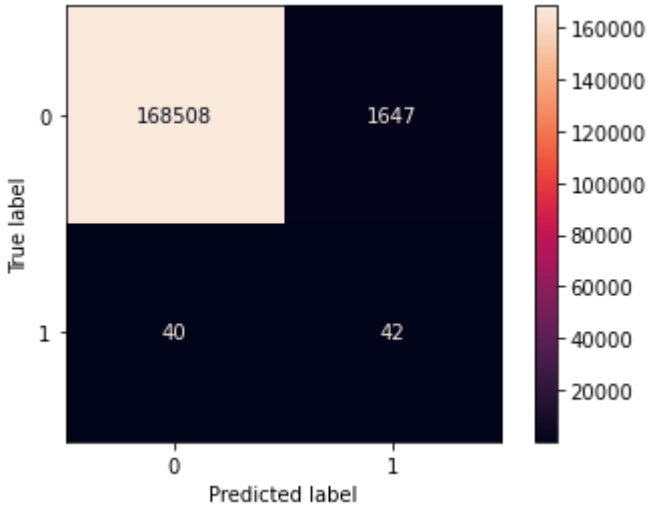

```
Accuracy Score:  0.9994771994337306
Precision Score:  0.0
Recall Score:  0.0
F1 Score:  0.0
GradientBoostingClassifier()   0.9994788742677176
```




```
Accuracy Score:  0.9993832128150755
Precision Score:  0.10344827586206896
Recall Score:  0.036585365853658534
F1 Score:  0.05405405405405405
GaussianNB()   0.8368524005770146
```




```
Accuracy Score:  0.8367217467413077
Precision Score:  0.002691645133505599
Recall Score:  0.9146341463414634
F1 Score:  0.005367494453589065
CategoricalNB()   0.9901993117112309
```

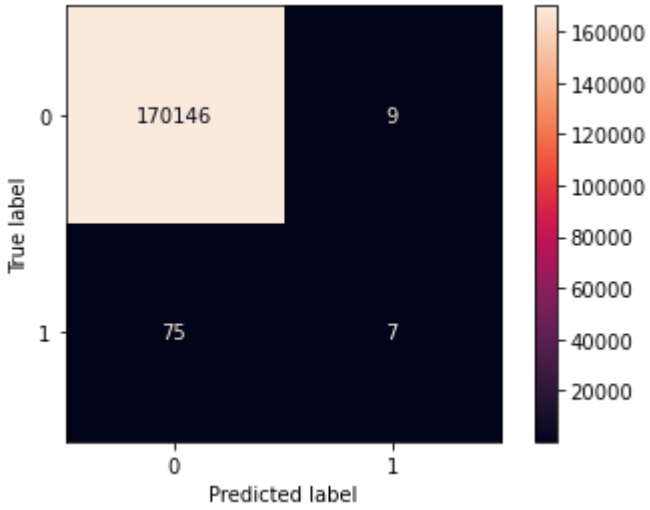



```
Accuracy Score:  0.9900902858955456
Precision Score:  0.02486678507992895
Recall Score:  0.5121951219512195
F1 Score:  0.047430830039525695
```

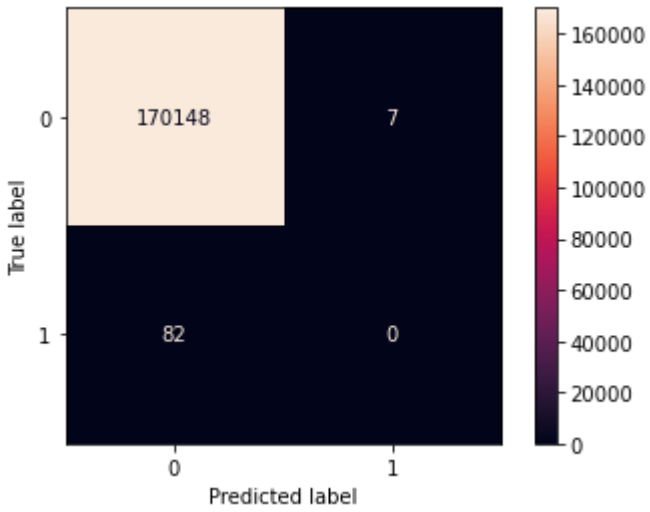
Out of the above models, Random Forest and Categorical B
the best. Let's see if after applying feature selection if they
better than the other models.

In [49]:
```
model_opt(models, X2, y2, X_valid2, y_valid2) # this is the reduced feature s
```

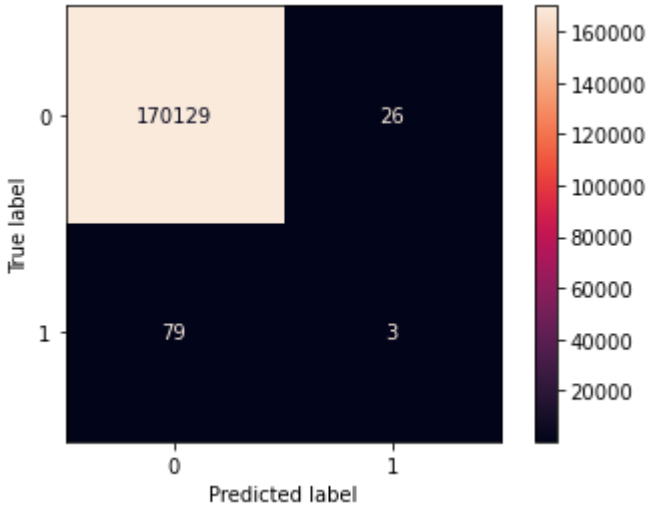
```
RandomForestClassifier()   0.9996198551421516
```



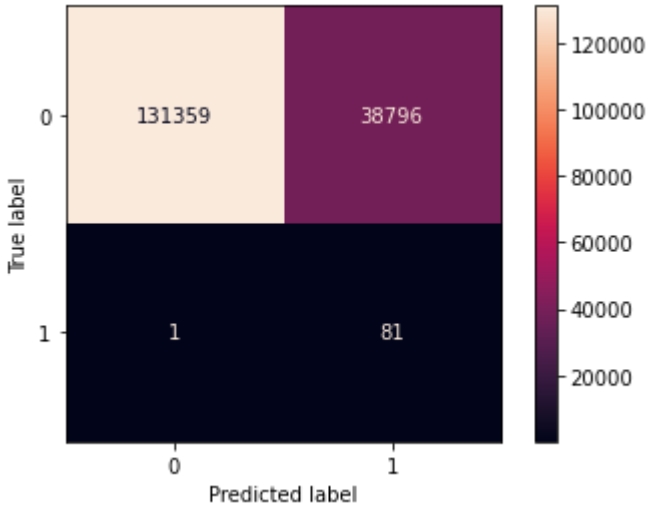

```
Accuracy Score:  0.9995065702520604
Precision Score:  0.4375
Recall Score:  0.08536585365853659
F1 Score:  0.14285714285714288
AdaBoostClassifier()   0.9995065669394815
```



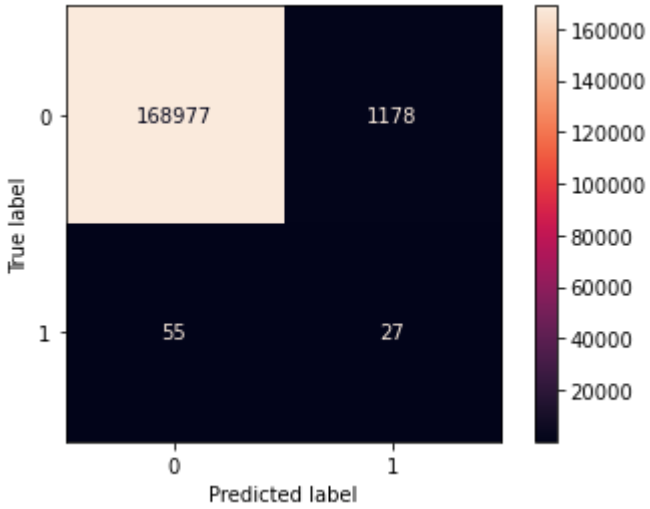

```
Accuracy Score:  0.9994771994337306
Precision Score:  0.0
Recall Score:  0.0
F1 Score:  0.0
GradientBoostingClassifier()   0.9994788742677176
```




```
Accuracy Score:  0.9993832128150755
Precision Score:  0.10344827586206896
Recall Score:  0.036585365853658534
F1 Score:  0.05405405405405405
GaussianNB()   0.7723511330079025
```




```
Accuracy Score:  0.7721000722522131
Precision Score:  0.0020834940967667257
Recall Score:  0.9878048780487805
F1 Score:  0.0041582176133884334
CategoricalNB()   0.9929887190125297
```




```
Accuracy Score:  0.992757156199886
Precision Score:  0.022406639004149378
Recall Score:  0.32926829268292684
F1 Score:  0.04195804195804196
```

Random Forest performed significantly w reduced set of features, while Categorica sideways, reducing overall false positives to ~1150 but also losing true positives.

These are the top features that indicate any amount of deat an accident in Chicago area crashes.

*EJECTION_TRAPPED/EXTRICATED (trapped within the vehicle requiring incident)

*PRIM_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF DRIVER (i

*TRAFFICWAY_TYPE_Y-INTERSECTION (atypical traffic conditions)

*FIRST_CRASH_TYPE_PEDESTRIAN (a pedestrian is involved)

*CRASH_TYPE_INJURY AND / OR TOW DUE TO CRASH

*AIRBAG_DEPLOYED_DEPLOYED, COMBINATION (higher speeds indica

*EJECTION_TOTALLY EJECTED (completed thrown from vehicle)

*SEC_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF DRIVER (in

In [50]:

```
# GridsearchCV and improving the full set Random Forest classifier
# GridsearchCV and improving the partial set CategoricalNB classifier
```

Tuning Hyper-parameters for a random forest model and a model

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

In [52]:
```
for model in selected_models:
    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'])
dict_keys(['alpha', 'class_prior', 'fit_prior'])
```

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

In [91]:
```
ran_for = RandomForestClassifier()
cat_bayes = CategoricalNB()
param_random = {
    "max_depth": [3,4,5],
    "n_estimators": [5,15,25],
    "max_leaf_nodes": range(3,10,2)
    }
param_bayes = {
    "alpha": [.05,.1,.2,.5,1,5,8,9,10,11,12,13,14],
    }
```

In [92]:
```
# 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, sc
                        verbose=1, n_jobs=6)
gsbayes = GridSearchCV(cat_bayes, param_grid=param_bayes, cv=5, scoring='prec
                         verbose=1, n_jobs=6)
```

In [56]:
```
# initial dataset
gsforest.fit(X, y)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits


[Parallel(n_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.
[Parallel(n_jobs=6)]: Done  38 tasks      | elapsed:  1.4min
[Parallel(n_jobs=6)]: Done 180 out of 180 | elapsed:  5.7min finished


GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=6,
             param_grid={'max_depth': [3, 4, 5],
                         'max_leaf_nodes': range(3, 10, 2),
                         'n_estimators': [5, 15, 25]},
             scoring='precision', verbose=1)
```

In [57]:
```
display(gsforest.best_estimator_)
display(gsforest.best_score_)
```

```
RandomForestClassifier(max_depth=3, max_leaf_nodes=3, n_estimators=5)


0.0
```

In [58]:
```
gsforest_f1 = gsforest.best_estimator_
gsforest_f1
```

```
RandomForestClassifier(max_depth=3, max_leaf_nodes=3, n_estimators=5)
```

In [59]:
```
print(gsforest_f1.score(X, y))
print(gsforest_f1.score(X_valid, y_valid))
```

```
0.9995342596112452
0.9995183185793922
```

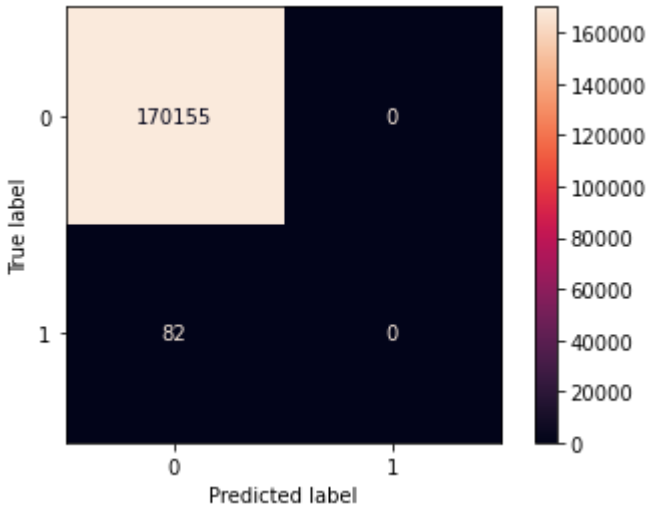
In [60]:
```
gsforest_f1.fit(X, y)
```

```
RandomForestClassifier(max_depth=3, max_leaf_nodes=3, n_estimators=5)
```

In [61]:
```
y_preds = gsforest_f1.predict(X_valid)
sum(y_preds)
```

```
0
```

In [62]:
```
plot_confusion_matrix(gsforest_f1, X_valid, y_valid, cmap='rocket', values_fo
plt.show() # model is not predicting any fatalities at all
```



In [ ]:

In [63]:
```
gsforest.fit(X2, y2)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

[Parallel(n_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.
[Parallel(n_jobs=6)]: Done  38 tasks      | elapsed:   21.8s
[Parallel(n_jobs=6)]: Done 180 out of 180 | elapsed:  1.7min finished

GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=6,
             param_grid={'max_depth': [3, 4, 5],
                         'max_leaf_nodes': range(3, 10, 2),
                         'n_estimators': [5, 15, 25]},
             scoring='precision', verbose=1)
```

In [64]:
```
gsforest_high_precision_partial = gsforest.best_estimator_
gsforest_high_precision_partial
```

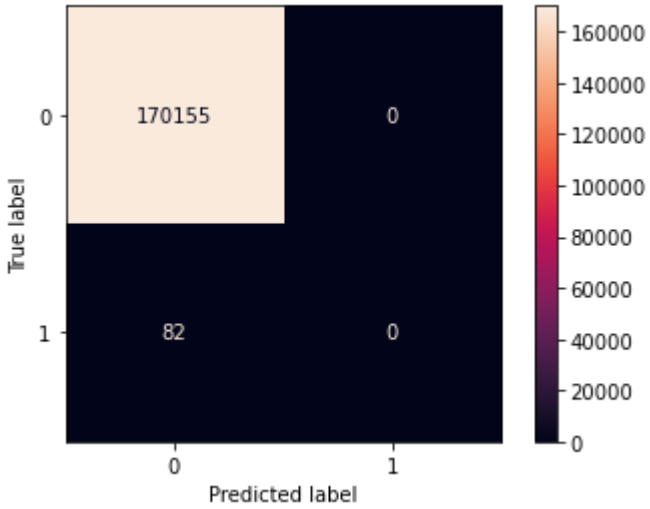
```
RandomForestClassifier(max_depth=3, max_leaf_nodes=3, n_estimators=5)
```

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

```
RandomForestClassifier(max_depth=3, max_leaf_nodes=3, n_estimators=5)
```

In [89]:
```
rcf_new = RandomForestClassifier(max_depth=3, max_leaf_nodes=3, n_estimators=
rcf_new.fit(X2, y2)
y_preds2 = rcf_new.predict(X_valid2)
```

In [90]:
```
plot_confusion_matrix(rcf_new, X_valid2, y_valid2, cmap='rocket', values_forr
plt.show() # model is not predicting any fatalities at all - how do I make it
```

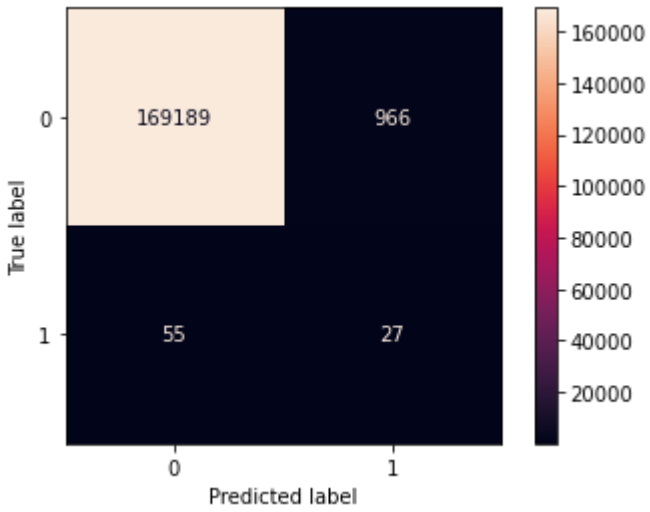


In [95]:
```
cat_bayes = CategoricalNB(alpha=10)
cat_bayes.fit(X2, y2)
```

```
CategoricalNB(alpha=10)
```

In [96]:
```
y_valid2_preds = cat_bayes.predict(X_valid2)
```

In [98]:
```
plot_confusion_matrix(cat_bayes, X_valid2, y_valid2, cmap='rocket', values_fo
plt.show() # decently pleased with this model, but I want to do better as wel
# occurrence in accidents with fatalities
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
print(gsforest_high_precision.score(X, y)) # full feature set
print(gsforest_high_precision_partial.score(X2, y2)) # top 18 feature set
print(gsbayes_high_precision_partial.score(X2, y2)) # top 18 feature set
```

In [ ]:
```
# going to plot and analyze forest_partial_best confusion matrix
# as well as gsbayes_partial confusion matrix
```

In [ ]:
```
# predict on gsforest_high_precision full set
forest_y_preds = gsforest_high_precision.predict(X_valid)
# define forest_partial model y predict
forest_partial_y_preds = gsforest_high_precision_partial.predict(X_valid2)
# define bayes_partial model y predict
bayes_partial_y_preds = gsbayes_high_precision_partial.predict(X_valid2)
final_models = [forest_y_preds, forest_partial_y_preds, bayes_partial_y_preds
```

In [ ]:
```
def plot_confusion(ytrue, ypred):
    cm_norm = confusion_matrix(ytrue, ypred, normalize='true')
    sns.heatmap(cm_norm, cmap=sns.color_palette('Blues'), fmt='{:.2f}', annot
    cm = confusion_matrix(ytrue, ypred)
    sns.heatmap(cm, cmap=sns.color_palette('Blues'), fmt='{:.2f}', annot=True
    plt.show()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
def model_scoring(models, )
for model in models:
    train_score = model.score(x_train_resampled, y_train_resampled)
    test_score = model.score(x_test, y_test)
    avg_score = (train_score + test_score)/2
    print(model ,train_score, test_score, avg_score)
```

In [ ]:
```
def evaluate_test_data(logreg, df_test, feature_for_modeling, target='y'):
    df_test = transform_dataframe(df_test)
    X = df_test[feature_for_modeling]
    y = df_test[target]
    score = logreg.score(X, y)
    print(f"Score of model = {score}")
    y_pred = logreg.predict(X)
    plot_confusion(y, y_pred)
    pass

evaluate_test_data(cat_bayes,df_test=df_valid, feature_for_modeling=feature_
```

In [ ]:
```
def cost_benefit_analysis(model, X_test, y_test):
    y_preds = model.predict(X_test)
    label_dict = {"TP":0, "FP": 0, "TN": 0, "FN": 0}
    for yt, yp in zip(y_test, y_preds):
        if yt==yp:
            if yt==1:
                label_dict["TP"] += 1
            else:
                label_dict["TN"] += 1
        else:
            if yp==1:
                label_dict["FP"] += 1
            else:
                label_dict["FN"] += 1
    cb_dict = {"TP": 50, "FP": -10, "TN": 0, "FN": -60}
    total = 0
    for key in label_dict.keys():
        total += cb_dict[key]*label_dict[key]
    return total / sum(label_dict.values())
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: