1 Logistic Regression

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

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

Out[2]:

	Unnamed: 0	neo	pha	H	epoch	epoch_mjd	epoch_cal	e	a	q	...
0	0	N	N	3.40	2458600.5	58600	20190427.0	0.076009	2.769165	2.558684	...
1	1	N	N	4.20	2459000.5	59000	20200531.0	0.229972	2.773841	2.135935	...
2	2	N	N	5.33	2459000.5	59000	20200531.0	0.256936	2.668285	1.982706	...
3	3	N	N	3.00	2458600.5	58600	20190427.0	0.088721	2.361418	2.151909	...
4	4	N	N	6.90	2459000.5	59000	20200531.0	0.190913	2.574037	2.082619	...

5 rows × 35 columns

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

Out[3]:

	Unnamed: 0	H	epoch	epoch_mjd	epoch_cal	e	
count	932335.000000	932335.000000	9.323350e+05	932335.000000	9.323350e+05	932335.000000	9
mean	473165.568655	16.890009	2.458895e+06	58894.728019	2.019763e+07	0.156221	
std	277616.874797	1.801243	6.439097e+02	643.909665	1.775660e+04	0.093001	
min	0.000000	-1.100000	2.425052e+06	25051.000000	1.927062e+07	0.000003	
25%	233084.500000	16.000000	2.459000e+06	59000.000000	2.020053e+07	0.092159	
50%	466168.000000	16.900000	2.459000e+06	59000.000000	2.020053e+07	0.144933	
75%	716936.500000	17.700000	2.459000e+06	59000.000000	2.020053e+07	0.200589	
max	958523.000000	33.200000	2.459000e+06	59000.000000	2.020053e+07	0.999851	

8 rows × 32 columns

In [3]:
```
df.drop(['Unnamed: 0'], axis=1, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 932335 entries, 0 to 932334
Data columns (total 34 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   neo        932335 non-null  object 
 1   pha        932335 non-null  object 
 2   H          932335 non-null  float64
 3   epoch      932335 non-null  float64
 4   epoch_mjd  932335 non-null  int64  
 5   epoch_cal  932335 non-null  float64
 6   e          932335 non-null  float64
 7   a          932335 non-null  float64
 8   q          932335 non-null  float64
 9   i          932335 non-null  float64
 10  om         932335 non-null  float64
 11  w          932335 non-null  float64
 12  ma         932335 non-null  float64
 13  ad         932335 non-null  float64
 14  n          932335 non-null  float64
 15  tp         932335 non-null  float64
 16  tp_cal     932335 non-null  float64
 17  per        932335 non-null  float64
 18  per_y      932335 non-null  float64
 19  moid       932335 non-null  float64
 20  moid_ld    932335 non-null  float64
 21  sigma_e    932335 non-null  float64
 22  sigma_a    932335 non-null  float64
 23  sigma_q    932335 non-null  float64
 24  sigma_i    932335 non-null  float64
 25  sigma_om   932335 non-null  float64
 26  sigma_w    932335 non-null  float64
 27  sigma_ma   932335 non-null  float64
 28  sigma_ad   932335 non-null  float64
 29  sigma_n    932335 non-null  float64
 30  sigma_tp   932335 non-null  float64
 31  sigma_per  932335 non-null  float64
 32  class      932335 non-null  object 
 33  rms        932335 non-null  float64
dtypes: float64(30), int64(1), object(3)
memory usage: 241.8+ MB
```

In [4]:
```
# Address object columns

df['pha'] = df['pha'].map(lambda x: 1 if x == 'Y' else 0)
df['neo'] = df['neo'].map(lambda x: 1 if x == 'Y' else 0)
```

In [5]:
```
# Get dummy variables

dummies = pd.get_dummies(df['class'], prefix='class')
df_cleaned = pd.concat([df, dummies], axis=1)
df_cleaned.drop('class', axis=1, inplace=True)
df_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 932335 entries, 0 to 932334
Data columns (total 45 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   neo        932335 non-null  int64  
 1   pha        932335 non-null  int64  
 2   H          932335 non-null  float64
 3   epoch      932335 non-null  float64
 4   epoch_mjd  932335 non-null  int64  
 5   epoch_cal  932335 non-null  float64
 6   e          932335 non-null  float64
 7   a          932335 non-null  float64
 8   q          932335 non-null  float64
 9   i          932335 non-null  float64
 10  om         932335 non-null  float64
 11  w          932335 non-null  float64
 12  ma         932335 non-null  float64
 13  ad         932335 non-null  float64
 14  n          932335 non-null  float64
 15  tp         932335 non-null  float64
 16  tp_cal     932335 non-null  float64
 17  per        932335 non-null  float64
 18  per_y      932335 non-null  float64
 19  moid       932335 non-null  float64
 20  moid_ld    932335 non-null  float64
 21  sigma_e    932335 non-null  float64
 22  sigma_a    932335 non-null  float64
 23  sigma_q    932335 non-null  float64
 24  sigma_i    932335 non-null  float64
 25  sigma_om   932335 non-null  float64
 26  sigma_w    932335 non-null  float64
 27  sigma_ma   932335 non-null  float64
 28  sigma_ad   932335 non-null  float64
 29  sigma_n    932335 non-null  float64
 30  sigma_tp   932335 non-null  float64
 31  sigma_per  932335 non-null  float64
 32  rms        932335 non-null  float64
 33  class_AMO  932335 non-null  uint8  
 34  class_APO  932335 non-null  uint8  
 35  class_AST  932335 non-null  uint8  
 36  class_ATE  932335 non-null  uint8  
 37  class_CEN  932335 non-null  uint8  
 38  class_IEO  932335 non-null  uint8  
 39  class_IMB  932335 non-null  uint8  
 40  class_MBA  932335 non-null  uint8  
 41  class_MCA  932335 non-null  uint8  
 42  class_OMB  932335 non-null  uint8  
 43  class_TJN  932335 non-null  uint8  
 44  class_TNO  932335 non-null  uint8  
```

```
dtypes: float64(30), int64(3), uint8(12)
memory usage: 245.4 MB
```

1.1 Iteration 1: Baseline

In [6]:
```
from sklearn.model_selection import train_test_split, GridSearchCV

y = df_cleaned['pha']
X = df_cleaned.drop('pha', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
```

In [9]:
```
# Perform SMOTE to address class imbalance

from sklearn.model_selection import train_test_split, GridSearchCV
from imblearn.over_sampling import SMOTE

# Previous class distribution
print('Original class distribution: \n')
print(y_train.value_counts())

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_sample(X_train, y_train)
# Synthetic class distribution:
print('-----------------------------')
print('Synthetic sample class distribution: \n')
print(y_train_resampled.value_counts())
```

```
Original class distribution: 

0    697673
1      1578
Name: pha, dtype: int64
-----------------------------
Synthetic sample class distribution: 

1    697673
0    697673
Name: pha, dtype: int64
```

In [10]:
```
# Drop last dummy variable for logistic regression

X_train_logit = X_train_resampled.drop('class_TNO', axis=1)
X_test_logit = X_test.drop('class_TNO', axis=1)
```

In [9]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
```

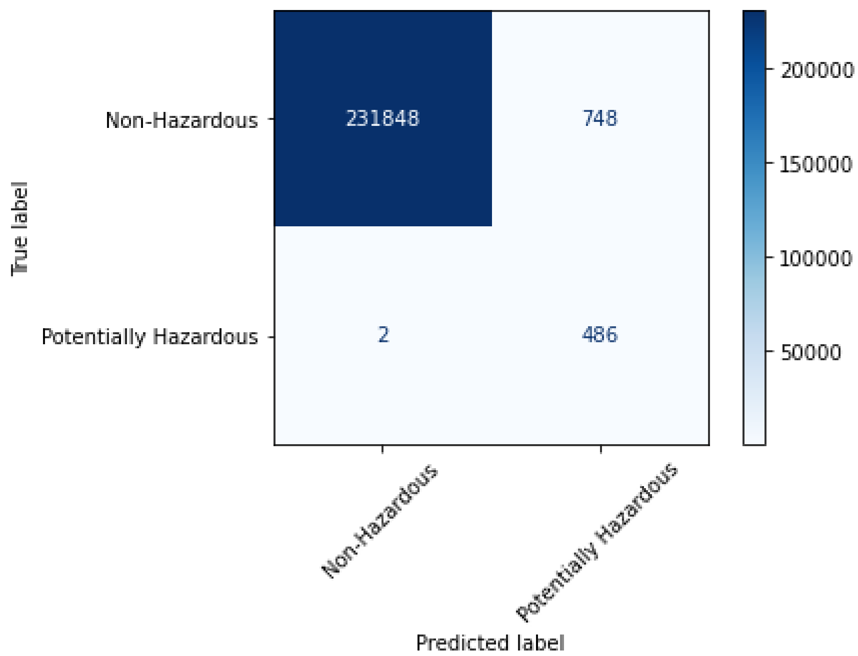
In [12]:
```
log_pipe = Pipeline([('ss', StandardScaler()),
                     ('logreg', LogisticRegression(solver='liblinear', C=1e
log_pipe.fit(X_train_logit, y_train_resampled)
y_pred = log_pipe.predict(X_test_logit)
```

In [13]:
```
# Classification reports for train and test data
print('Train Report')
print(classification_report(y_train_resampled, log_pipe.predict(X_train_log
print('\n')
print('Test Report')
print(classification_report(y_test, y_pred))
```

```
Train Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    697673
           1       1.00      1.00      1.00    697673

    accuracy                           1.00   1395346
   macro avg       1.00      1.00      1.00   1395346
weighted avg       1.00      1.00      1.00   1395346



Test Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    232596
           1       0.39      1.00      0.56       488

    accuracy                           1.00    233084
   macro avg       0.70      1.00      0.78    233084
weighted avg       1.00      1.00      1.00    233084
```

The model is overfit to the training data. This could be do to the manner in which class imbalance was addressed (SMOTE).

In [14]:
```
plot_confusion_matrix(log_pipe, X_test_logit, y_test, cmap=plt.cm.Blues,
                      display_labels=['Non-Hazardous', 'Potentially Hazardo
```



1.2 Iteration 2

1.2.0.1 Important features of baseline model

Outliers will be removed from features that have the highest weights from Iteration 1.

In [15]:
```
coef = log_pipe.steps[1][1].coef_
coef[0]
```

Out[15]:
```
array([-2.75430704e+00, -5.43653653e+00, -7.78820632e-01, -1.32102690e+0
0,
        2.33771951e+00,  6.54867959e-03, -3.39488698e+00, -1.44150963e+0
0,
        7.15747971e-02, -6.58501564e-02, -2.76793872e-02,  5.85260895e-0
2,
       -3.35830715e+00, -4.22793847e-02,  1.43341066e-01, -3.88102488e-0
1,
       -8.93861627e-02, -8.93861627e-02, -9.25093889e+01, -9.25093889e+0
1,
       -5.54324941e+00,  4.25526966e-02, -3.83318327e-01,  1.16895459e+0
0,
        5.63493249e-01,  1.60979975e-01,  1.62030876e-01,  3.89900452e-0
2,
       -8.26688020e-01,  1.62648465e-01,  7.37491572e-02,  9.84660422e-0
2,
       -2.79944830e+00, -1.47770569e+01,  1.26272800e-01, -5.11451468e+0
0,
       -3.75233068e+00, -3.67205637e-01,  9.58664865e-01,  4.34974720e+0
0,
       -2.93320930e+00, -3.41215235e+00,  2.73904900e+00])
```

In [16]:
```
coef_df = pd.DataFrame(zip(X_train_logit.columns, np.transpose(coef[0])), c
coef_df.head()
```

Out[16]:

	features	coef
0	neo	-2.754307
1	H	-5.436537
2	epoch	-0.778821
3	epoch_mjd	-1.321027
4	epoch_cal	2.337720

In [17]:
```
coef_sorted = coef_df.iloc[coef_df['coef'].abs().argsort()[::-1]]
coef_sorted
```

Out[17]:

	features	coef
18	moid	-92.509389
19	moid_ld	-92.509389
33	class_APO	-14.777057
20	sigma_e	-5.543249
1	H	-5.436537
35	class_ATE	-5.114515
39	class_MBA	4.349747
36	class_CEN	-3.752331
41	class_OMB	-3.412152
6	a	-3.394887
12	ad	-3.358307
40	class_MCA	-2.933209
32	class_AMO	-2.799448
0	neo	-2.754307
42	class_TJN	2.739049
4	epoch_cal	2.337720
7	q	-1.441510
3	epoch_mjd	-1.321027
23	sigma_i	1.168955
38	class_IMB	0.958665
28	sigma_n	-0.826688
2	epoch	-0.778821
24	sigma_om	0.563493
15	tp_cal	-0.388102
22	sigma_q	-0.383318
37	class_IEO	-0.367206
29	sigma_tp	0.162648
26	sigma_ma	0.162031
25	sigma_w	0.160980
14	tp	0.143341
34	class_AST	0.126273
31	rms	0.098466

	features	coef
17	per_y	-0.089386
16	per	-0.089386
30	sigma_per	0.073749
8	i	0.071575
9	om	-0.065850
11	ma	0.058526
21	sigma_a	0.042553
13	n	-0.042279
27	sigma_ad	0.038990
10	w	-0.027679
5	e	0.006549

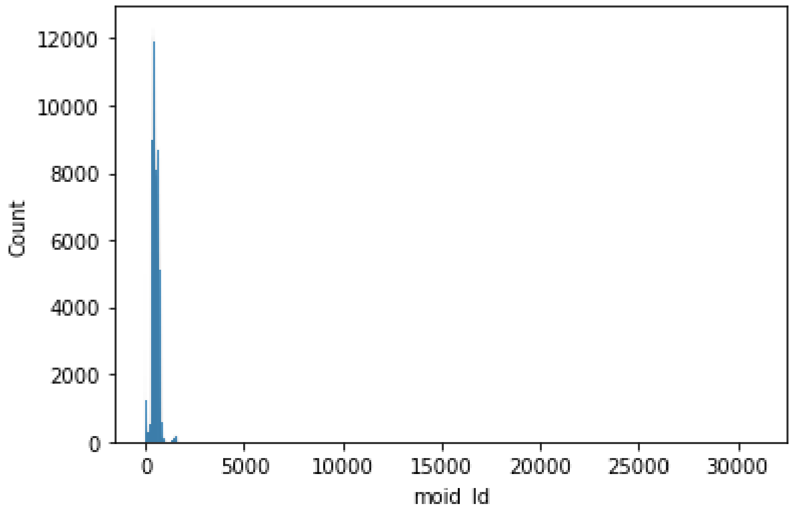
1.2.0.2 Remove outliers

In [18]:
```
top = coef_sorted.iloc[:20, 0].to_list()
top
```

Out[18]:
```
['moid',
 'moid_ld',
 'class_APO',
 'sigma_e',
 'H',
 'class_ATE',
 'class_MBA',
 'class_CEN',
 'class_OMB',
 'a',
 'ad',
 'class_MCA',
 'class_AMO',
 'neo',
 'class_TJN',
 'epoch_cal',
 'q',
 'epoch_mjd',
 'sigma_i',
 'class_IMB']
```

In [20]:
```
sns.histplot(x=df_cleaned['moid_ld'], data=df_cleaned)
```

Out[20]: <AxesSubplot:xlabel='moid_ld', ylabel='Count'>



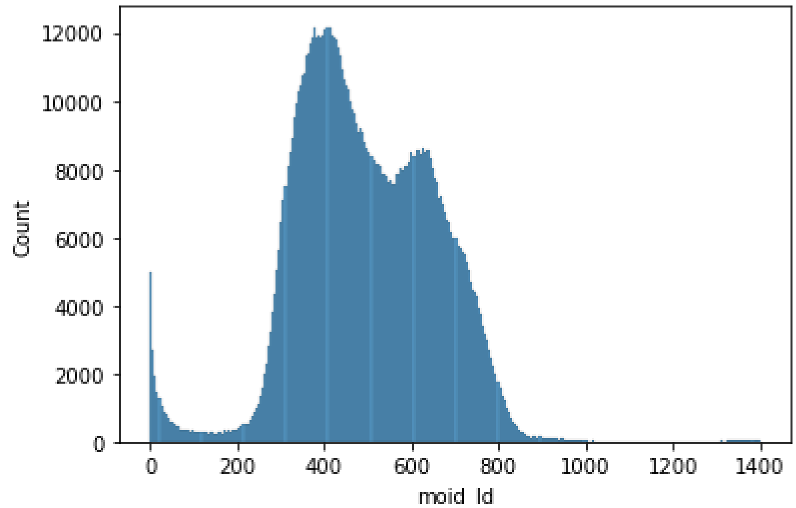
In [21]:
```
for i in range(90, 101):
    q = i/100
    print('{} percentile: {}'.format(q, df_cleaned['moid_ld'].quantile(q=q)
```

```
0.9 percentile: 713.2902345
0.91 percentile: 721.3888622000002
0.92 percentile: 729.8221761
0.93 percentile: 739.1350142000001
0.94 percentile: 749.1872753
0.95 percentile: 760.4588060099998
0.96 percentile: 774.096645988
0.97 percentile: 791.7741484000002
0.98 percentile: 823.2164380439997
0.99 percentile: 1465.7542433660003
1.0 percentile: 30929.908422000004
```

In [22]:
```
df_no_fliers = df_cleaned[df_cleaned['moid_ld'] <= 1400]
```

In [23]:
```
sns.histplot(x=df_no_fliers['moid_ld'], data=df_no_fliers)
```

Out[23]:
```
<AxesSubplot:xlabel='moid_ld', ylabel='Count'>
```



In [24]:
```
for i in range(90, 101):
    q = i/100
    print('{} percentile: {}'.format(q, df_no_fliers['H'].quantile(q=q)))
```

```
0.9 percentile: 18.503
0.91 percentile: 18.6
0.92 percentile: 18.7
0.93 percentile: 18.8
0.94 percentile: 18.926
0.95 percentile: 19.1
0.96 percentile: 19.3
0.97 percentile: 19.678
0.98 percentile: 20.7
0.99 percentile: 23.9
1.0 percentile: 33.2
```

In [25]:
```
for i in range(90, 101):
    q = i/100
    print('{} percentile: {}'.format(q, df_no_fliers['a'].quantile(q=q)))
```

```
0.9 percentile: 3.137504811878804
0.91 percentile: 3.1456752689627785
0.92 percentile: 3.1538579175794847
0.93 percentile: 3.161942623963375
0.94 percentile: 3.169504698189642
0.95 percentile: 3.1780513372391166
0.96 percentile: 3.188635074591071
0.97 percentile: 3.2016359950394575
0.98 percentile: 3.2198620240993794
0.99 percentile: 3.3427507957795735
1.0 percentile: 8850.823836220028
```

In [26]:
```
df_no_fliers[df_no_fliers['a'] > 50]
```

Out[26]:

	neo	pha	H	epoch	epoch_mjd	epoch_cal	e	a	q	
65406	0	0	12.300	2459000.5	59000	20200531.0	0.954279	53.711349	2.455753	11
545693	0	0	13.900	2459000.5	59000	20200531.0	0.990378	258.236026	2.484772	3
547621	0	0	14.000	2459000.5	59000	20200531.0	0.955952	52.560568	2.315170	6
548391	0	0	10.500	2459000.5	59000	20200531.0	0.963222	90.594428	3.331839	4
558466	0	0	15.300	2459000.5	59000	20200531.0	0.995107	547.968271	2.681115	5
570912	0	0	15.300	2459000.5	59000	20200531.0	0.976241	99.420445	2.362180	16
579052	0	0	13.700	2459000.5	59000	20200531.0	0.951859	63.065588	3.036066	6
586407	0	0	14.100	2459000.5	59000	20200531.0	0.994971	815.300107	4.099865	11
608821	0	0	14.912	2459000.5	59000	20200531.0	0.994286	467.703829	2.672544	7
661281	0	0	14.900	2459000.5	59000	20200531.0	0.941033	62.903042	3.709228	10
662762	0	0	14.600	2459000.5	59000	20200531.0	0.962631	69.913840	2.612622	5
684789	0	0	13.100	2459000.5	59000	20200531.0	0.988807	278.540566	3.117749	11
695263	0	0	17.570	2455890.5	55890	20111125.0	0.967299	64.417859	2.106512	14
702737	0	0	15.300	2459000.5	59000	20200531.0	0.959436	62.997346	2.555432	14
721402	0	0	17.100	2459000.5	59000	20200531.0	0.976448	66.256305	1.560469	10
729833	0	0	16.900	2456398.5	56398	20130416.0	0.976082	117.398848	2.807932	9
731037	0	0	16.100	2459000.5	59000	20200531.0	0.968737	81.536772	2.549046	15
789633	0	0	14.900	2459000.5	59000	20200531.0	0.955092	72.462607	3.254172	10
826504	0	0	14.500	2459000.5	59000	20200531.0	0.965619	80.206014	2.757567	9
830492	0	0	15.300	2459000.5	59000	20200531.0	0.956012	54.256063	2.386611	9
848293	0	0	15.100	2459000.5	59000	20200531.0	0.957674	63.387085	2.682937	2
858182	0	0	18.100	2459000.5	59000	20200531.0	0.959518	67.427405	2.729576	17
885392	0	0	15.600	2459000.5	59000	20200531.0	0.984603	191.855483	2.953986	15
892673	0	0	13.300	2459000.5	59000	20200531.0	0.952298	66.798336	3.186388	6
893922	0	0	17.800	2459000.5	59000	20200531.0	0.970597	58.970559	1.733905	9
899736	0	0	17.900	2459000.5	59000	20200531.0	0.992421	243.218889	1.843290	15
904392	0	0	15.300	2459000.5	59000	20200531.0	0.975922	130.289321	3.137077	14
904408	1	0	21.200	2459000.5	59000	20200531.0	0.996476	352.628376	1.242620	10
909829	0	0	16.200	2459000.5	59000	20200531.0	0.983633	97.754076	1.599953	6
911263	0	0	13.600	2459000.5	59000	20200531.0	0.960742	94.348891	3.703908	4
911901	0	0	15.300	2459000.5	59000	20200531.0	0.989302	357.957517	3.829530	6
914543	0	0	15.400	2459000.5	59000	20200531.0	0.980949	155.876721	2.969552	9
915765	0	0	15.800	2458626.5	58626	20190523.0	0.988976	121.506582	1.339528	16

	neo	pha	H	epoch	epoch_mjd	epoch_cal	e	a	q
920616	0	0	16.312	2458717.5	58717	20190822.0	0.996659	685.920963	2.291345
925733	1	0	18.167	2458728.5	58728	20190902.0	0.978750	59.195648	1.257886
928882	0	0	12.735	2458784.5	58784	20191028.0	0.998459	1717.507044	2.645908
929683	0	0	10.278	2458868.5	58868	20200120.0	0.999591	8850.823836	3.622724
931029	0	0	21.369	2458884.5	58884	20200205.0	0.966594	52.316301	1.747697
932123	0	0	17.641	2458975.5	58975	20200506.0	0.994756	491.712757	2.578756

39 rows × 45 columns

In [27]:
```
df_no_fliers = df_no_fliers[df_no_fliers['a'] <= 50]
```

In [28]:
```
for i in range(90, 101):
    q = i/100
    print('{} percentile: {}'.format(q, df_no_fliers['ad'].quantile(q=q)))
```

```
0.9 percentile: 3.641633351346189
0.91 percentile: 3.6688832051345863
0.92 percentile: 3.6982889831120067
0.93 percentile: 3.7307385573753167
0.94 percentile: 3.767001220398261
0.95 percentile: 3.808535392293984
0.96 percentile: 3.8557661788597186
0.97 percentile: 3.917206662808669
0.98 percentile: 4.014537203875679
0.99 percentile: 4.29638433415904
1.0 percentile: 93.72990930613251
```

In [29]:
```
df_no_fliers[df_no_fliers['ad'] > 20].count()
```

Out[29]:
```
neo            74
pha            74
H              74
epoch          74
epoch_mjd      74
epoch_cal      74
e              74
a              74
q              74
i              74
om             74
w              74
ma             74
ad             74
n              74
tp             74
tp_cal         74
per            74
per_y          74
moid           74
moid_ld        74
sigma_e        74
sigma_a        74
sigma_q        74
sigma_i        74
sigma_om       74
sigma_w        74
sigma_ma       74
sigma_ad       74
sigma_n        74
sigma_tp       74
sigma_per      74
rms            74
class_AMO      74
class_APO      74
class_AST      74
class_ATE      74
class_CEN      74
class_IEO      74
class_IMB      74
class_MBA      74
class_MCA      74
class_OMB      74
class_TJN      74
class_TNO      74
dtype: int64
```

In [30]:
```
df_no_fliers = df_no_fliers[df_no_fliers['ad'] <= 20]
```

In [31]:
```
for i in range(90, 101):
    q = i/100
    print('{} percentile: {}'.format(q, df_no_fliers['epoch_cal'].quantile(
```

```
0.9 percentile: 20200531.0
0.91 percentile: 20200531.0
0.92 percentile: 20200531.0
0.93 percentile: 20200531.0
0.94 percentile: 20200531.0
0.95 percentile: 20200531.0
0.96 percentile: 20200531.0
0.97 percentile: 20200531.0
0.98 percentile: 20200531.0
0.99 percentile: 20200531.0
1.0 percentile: 20200531.0
```

In [32]:
```
for i in range(90, 101):
    q = i/100
    print('{} percentile: {}'.format(q, df_no_fliers['tp'].quantile(q=q)))
```

```
0.9 percentile: 2459617.208064441
0.91 percentile: 2459636.95749063
0.92 percentile: 2459657.527962704
0.93 percentile: 2459680.955364708
0.94 percentile: 2459707.446765461
0.95 percentile: 2459735.3205843735
0.96 percentile: 2459766.1696941904
0.97 percentile: 2459804.595133252
0.98 percentile: 2459858.4804402855
0.99 percentile: 2459930.309236824
1.0 percentile: 2464852.8332943683
```

In [33]:
```
for i in range(90, 101):
    q = i/100
    print('{} percentile: {}'.format(q, df_no_fliers['q'].quantile(q=q)))
```

```
0.9 percentile: 2.79853431224254
0.91 percentile: 2.8181881397961766
0.92 percentile: 2.838484815050618
0.93 percentile: 2.8597483744264274
0.94 percentile: 2.8829397479652257
0.95 percentile: 2.9083376709869504
0.96 percentile: 2.9364175876479077
0.97 percentile: 2.9695171128663094
0.98 percentile: 3.0131733946377546
0.99 percentile: 3.084404840557823
1.0 percentile: 4.612030525950536
```

The most outliers were removed based on `moid_ld` , `a` , and `ad` .

In [35]:
```
df_no_fliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 921430 entries, 0 to 932334
Data columns (total 45 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   neo        921430 non-null  int64  
 1   pha        921430 non-null  int64  
 2   H          921430 non-null  float64
 3   epoch      921430 non-null  float64
 4   epoch_mjd  921430 non-null  int64  
 5   epoch_cal  921430 non-null  float64
 6   e          921430 non-null  float64
 7   a          921430 non-null  float64
 8   q          921430 non-null  float64
 9   i          921430 non-null  float64
 10  om         921430 non-null  float64
 11  w          921430 non-null  float64
 12  ma         921430 non-null  float64
 13  ad         921430 non-null  float64
 14  n          921430 non-null  float64
 15  tp         921430 non-null  float64
 16  tp_cal     921430 non-null  float64
 17  per        921430 non-null  float64
 18  per_y      921430 non-null  float64
 19  moid       921430 non-null  float64
 20  moid_ld    921430 non-null  float64
 21  sigma_e    921430 non-null  float64
 22  sigma_a    921430 non-null  float64
 23  sigma_q    921430 non-null  float64
 24  sigma_i    921430 non-null  float64
 25  sigma_om   921430 non-null  float64
 26  sigma_w    921430 non-null  float64
 27  sigma_ma   921430 non-null  float64
 28  sigma_ad   921430 non-null  float64
 29  sigma_n    921430 non-null  float64
 30  sigma_tp   921430 non-null  float64
 31  sigma_per  921430 non-null  float64
 32  rms        921430 non-null  float64
 33  class_AMO  921430 non-null  uint8  
 34  class_APO  921430 non-null  uint8  
 35  class_AST  921430 non-null  uint8  
 36  class_ATE  921430 non-null  uint8  
 37  class_CEN  921430 non-null  uint8  
 38  class_IEO  921430 non-null  uint8  
 39  class_IMB  921430 non-null  uint8  
 40  class_MBA  921430 non-null  uint8  
 41  class_MCA  921430 non-null  uint8  
 42  class_OMB  921430 non-null  uint8  
 43  class_TJN  921430 non-null  uint8  
 44  class_TNO  921430 non-null  uint8  
dtypes: float64(30), int64(3), uint8(12)
memory usage: 249.6 MB
```

In [63]:
```
y = df_no_fliers['pha']
X = df_no_fliers.drop('pha', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
```

In [37]:
```
# Previous class distribution
print('Original class distribution: \n')
print(y_train.value_counts())

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_sample(X_train, y_train)
# Synthetic class distribution:
print('-----------------------------')
print('Synthetic sample class distribution: \n')
print(y_train_resampled.value_counts())
```

```
Original class distribution: 

0    689548
1      1524
Name: pha, dtype: int64
-----------------------------
Synthetic sample class distribution: 

1    689548
0    689548
Name: pha, dtype: int64
```

In [38]:
```
# Drop last dummy variable for logistic regression

X_train_logit = X_train_resampled.drop('class_TNO', axis=1)
X_test_logit = X_test.drop('class_TNO', axis=1)
```

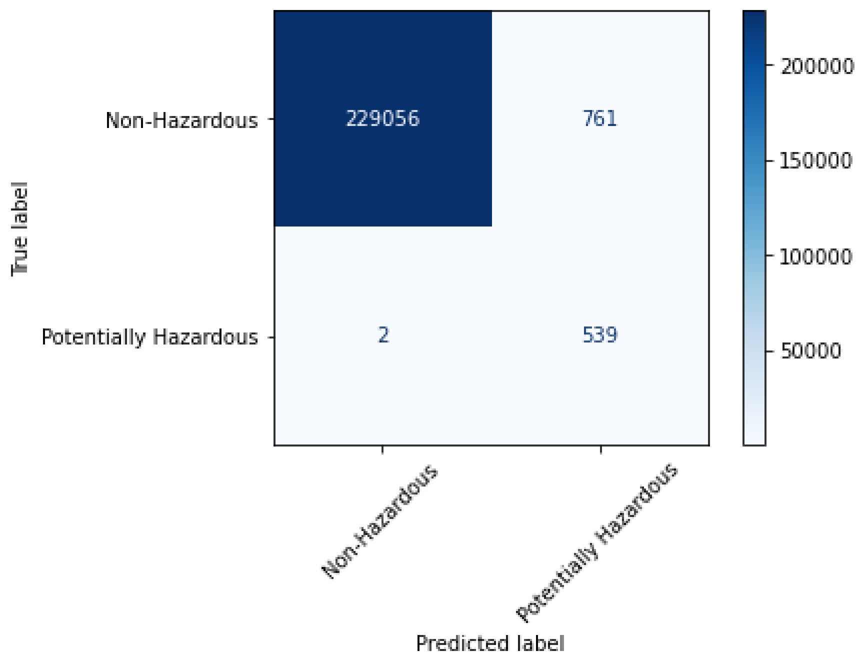
In [39]:
```
log_pipe2 = Pipeline([('ss', StandardScaler()),
                      ('logreg', LogisticRegression(solver='liblinear', C=1e
log_pipe2.fit(X_train_logit, y_train_resampled)
y_pred = log_pipe2.predict(X_test_logit)
```

In [40]:
```
print('Train Report')
print(classification_report(y_train_resampled, log_pipe2.predict(X_train_lo
print('\n')
print('Test Report')
print(classification_report(y_test, y_pred))
```

```
Train Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    689548
           1       1.00      1.00      1.00    689548

    accuracy                           1.00   1379096
   macro avg       1.00      1.00      1.00   1379096
weighted avg       1.00      1.00      1.00   1379096



Test Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    229817
           1       0.41      1.00      0.59       541

    accuracy                           1.00    230358
   macro avg       0.71      1.00      0.79    230358
weighted avg       1.00      1.00      1.00    230358
```

With the removal of outliers, performance on the test data has improved compared to the previous iteration. However, the model is still overfit to the training data.

In [41]:
```
plot_confusion_matrix(log_pipe2, X_test_logit, y_test, cmap=plt.cm.Blues,
                      display_labels=['Non-Hazardous', 'Potentially Hazardo
```



1.3 Iteration 3: No SMOTE, no outliers

This iteration will address overfitting by using the `class_weight` hyperparameter rather than SMOTE.

In [7]:
```
X_train = X_train.drop('class_TNO', axis=1)
X_test = X_test.drop('class_TNO', axis=1)
```

In [10]:
```
# handle class imbalance in class_weight parameter instead of using SMOTE

log_pipe3 = Pipeline([('ss', StandardScaler()),
                      ('logreg', LogisticRegression(solver='liblinear', C=1e
log_pipe3.fit(X_train, y_train)
```

Out[10]:
```
Pipeline(steps=[('ss', StandardScaler()),
                ('logreg',
                 LogisticRegression(C=1e+25, class_weight={0: 1, 1: 200},
                                    max_iter=1000, solver='liblinear'))])
```

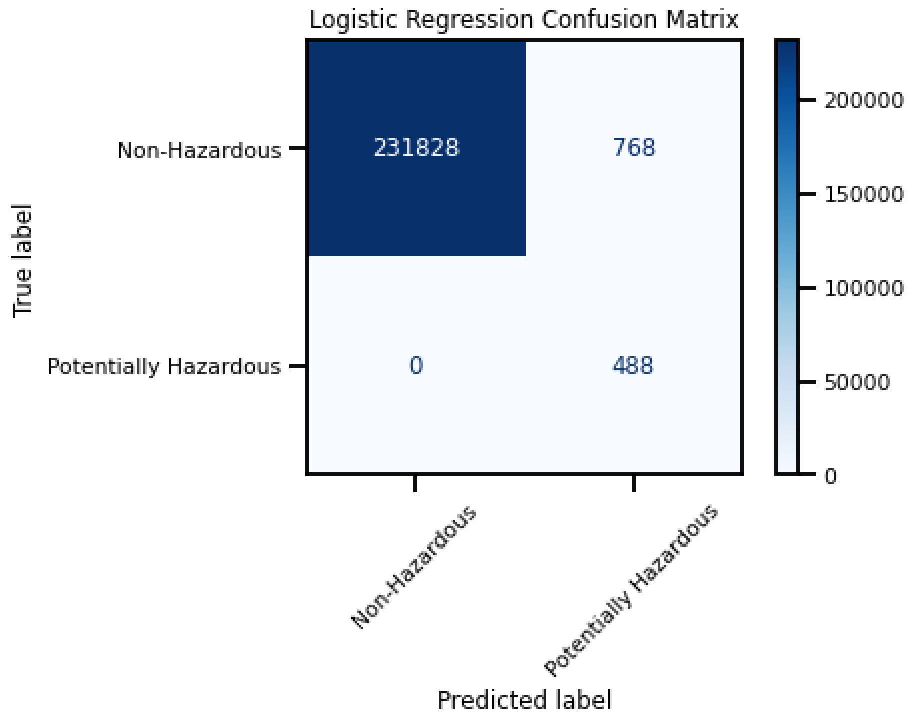
In [11]:
```
y_pred = log_pipe3.predict(X_test)
```

In [12]:
```
print('Train Report')
print(classification_report(y_train, log_pipe3.predict(X_train)))
print('\n')
print('Test Report')
print(classification_report(y_test, y_pred))
```

```
Train Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    697673
           1       0.42      1.00      0.59      1578

    accuracy                           1.00    699251
   macro avg       0.71      1.00      0.80    699251
weighted avg       1.00      1.00      1.00    699251



Test Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    232596
           1       0.39      1.00      0.56       488

    accuracy                           1.00    233084
   macro avg       0.69      1.00      0.78    233084
weighted avg       1.00      1.00      1.00    233084
```

The model is no longer overfit to the training data, as the test and train reports have similar scores. It is possible that the minority class samples had enough variation that SMOTE generated synthetic minority samples that were not real neighbors.

In [14]:
```
sns.set_context('talk', font_scale=0.65)

plot_confusion_matrix(log_pipe3, X_test, y_test, cmap=plt.cm.Blues,
                      display_labels=['Non-Hazardous', 'Potentially Hazardo
```



This model seems to have missed no phas (very high recall for the positive class), but it has a large number of false positives.

In [46]:
```
df_no_fliers.to_csv('data/data_no_fliers.csv')
```

1.4 Iterations 4 & 5: Addressing Multicolinearity

Iteration 4 will remove features on the basis of multicolinearity. A new list of important features will be used to remove outliers for an iteration (Iteration 5) that can be compared with Iteration 3, which retains all features.

1.4.0.1 Iteration 4: With outliers, less multicolinearity

In [15]:
```
def corrFilter(x: pd.DataFrame, bound: float):
    xCorr = x.corr()
    xFiltered = xCorr[((xCorr >= bound) | (xCorr <= -bound))]
    xFlattened = xFiltered.unstack().sort_values().drop_duplicates()
    return xFlattened
```

In [16]:
```
corr = corrFilter(df_cleaned, .94)
corr
```

Out[16]:
```
q          class_TNO    0.945561
class_TNO  moid_ld      0.946813
           moid         0.946813
a          per          0.981604
per_y      a            0.981604
per        ad           0.983439
per_y      ad           0.983439
sigma_a    sigma_ad     0.996709
tp         tp_cal       0.998229
sigma_tp   sigma_w      0.998972
           sigma_ma     0.998973
ad         a            0.999560
epoch_cal  epoch_mjd    0.999575
           epoch        0.999575
moid_ld    q            0.999734
moid       q            0.999734
sigma_w    sigma_ma     1.000000
moid_ld    moid         1.000000
sigma_w    sigma_w      1.000000
per        per_y        1.000000
epoch_mjd  epoch        1.000000
neo        pha               NaN
dtype: float64
```

In [17]:
```
type(corrFilter(df_cleaned, .94))
```

Out[17]:
```
pandas.core.series.Series
```

In [18]:
```
# Remove some highly correlated features.

to_drop = ['epoch_mjd', 'per_y', 'moid', 'sigma_ma', 'q', 'epoch_cal', 'ad'
df_cleaned2 = df_cleaned.drop(to_drop, axis=1)
```

In [19]:
```
X = df_cleaned2.drop('pha', axis=1)
y = df_cleaned2['pha']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r

X_train = X_train.drop('class_TNO', axis=1)
X_test = X_test.drop('class_TNO', axis=1)
```

In [20]:
```
log_pipe4 = Pipeline([('ss', StandardScaler()),
                      ('logreg', LogisticRegression(solver='liblinear', C=1e
log_pipe4.fit(X_train, y_train)
y_pred = log_pipe4.predict(X_test)
```

In [21]:
```
print('Train Report')
print(classification_report(y_train, log_pipe4.predict(X_train)))
print('\n')
print('Test Report')
print(classification_report(y_test, y_pred))
```

```
Train Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    697673
           1       0.42      1.00      0.59      1578

    accuracy                           1.00    699251
   macro avg       0.71      1.00      0.80    699251
weighted avg       1.00      1.00      1.00    699251



Test Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    232596
           1       0.39      1.00      0.56       488

    accuracy                           1.00    233084
   macro avg       0.69      1.00      0.78    233084
weighted avg       1.00      1.00      1.00    233084
```

Simply removing multicolinear features has not improved the model on any metric.

In [22]:
```
# Find features with the largest weights

coef = log_pipe4.steps[1][1].coef_
coef[0]
```

Out[22]:
```
array([-3.54723253e+00, -4.85789832e+00,  1.56955003e-01,  7.36451578e-0
2,
       -2.93767604e+01,  1.25427351e-01, -6.50956560e-02, -2.28792891e-0
2,
        6.70442569e-02, -9.42148482e-02, -1.98761018e-01, -3.20969372e-0
1,
       -3.16982152e+02, -3.60264006e+00,  2.56053590e-01,  5.52555044e-0
1,
        4.34572721e+00, -6.33564483e+00, -4.04583149e+00,  7.38074848e-0
1,
        3.48618071e-01,  8.13635438e-02, -2.12988924e+00, -2.63739432e+0
0,
        3.80023383e-01, -9.53309061e-01, -6.60216933e-02, -1.01767824e-0
1,
        1.97715077e+00,  4.55573191e+00, -4.84371605e+00, -6.20301473e+0
0,
        1.10900365e+01])
```

In [23]:
```
coef_df = pd.DataFrame(zip(X_train.columns, np.transpose(coef[0])), columns
coef_df.head()
```

Out[23]:

	features	coef
0	neo	-3.547233
1	H	-4.857898
2	epoch	0.156955
3	e	0.073645
4	a	-29.376760

In [24]:
```
coef_sorted = coef_df.iloc[coef_df['coef'].abs().argsort()[::-1]]
coef_sorted
```

Out[24]:

	features	coef
12	moid_ld	-316.982152
4	a	-29.376760
32	class_TJN	11.090037
17	sigma_om	-6.335645
31	class_OMB	-6.203015
1	H	-4.857898
30	class_MCA	-4.843716
29	class_MBA	4.555732
16	sigma_i	4.345727
18	sigma_n	-4.045831
13	sigma_e	-3.602640
0	neo	-3.547233
23	class_APO	-2.637394
22	class_AMO	-2.129889
28	class_IMB	1.977151
25	class_ATE	-0.953309
19	sigma_tp	0.738075
15	sigma_q	0.552555
24	class_AST	0.380023
20	sigma_per	0.348618
11	per	-0.320969
14	sigma_a	0.256054
10	tp	-0.198761
2	epoch	0.156955
5	i	0.125427
27	class_IEO	-0.101768
9	n	-0.094215
21	rms	0.081364
3	e	0.073645
8	ma	0.067044
26	class_CEN	-0.066022
6	om	-0.065096

	features	coef
7	w	-0.022879

Remove outliers based on new list of important features:

In [25]:
```
for i in range(90, 101):
    q = i/100
    print('{} percentile: {}'.format(q, df_cleaned2['moid_ld'].quantile(q=q
```

```
0.9 percentile: 713.2902345
0.91 percentile: 721.3888622000002
0.92 percentile: 729.8221761
0.93 percentile: 739.1350142000001
0.94 percentile: 749.1872753
0.95 percentile: 760.4588060099998
0.96 percentile: 774.096645988
0.97 percentile: 791.7741484000002
0.98 percentile: 823.2164380439997
0.99 percentile: 1465.7542433660003
1.0 percentile: 30929.908422000004
```

In [26]:
```
for i in range(90, 101):
    q = i/100
    print('{} percentile: {}'.format(q, df_cleaned2['a'].quantile(q=q)))
```

```
0.9 percentile: 3.1461263054346817
0.91 percentile: 3.154391291029324
0.92 percentile: 3.162582433918047
0.93 percentile: 3.17019134970903
0.94 percentile: 3.178982879839905
0.95 percentile: 3.1899532879771173
0.96 percentile: 3.2034328321728465
0.97 percentile: 3.22318831974346
0.98 percentile: 3.4109147143645298
0.99 percentile: 5.178501664039422
1.0 percentile: 33488.895954563486
```

In [27]:
```
df_no_fliers2 = df_cleaned2[df_cleaned2['moid_ld'] <= 1400]
df_no_fliers2 = df_no_fliers2[df_no_fliers2['a'] <= 50]
```

1.4.0.2 Iteration 5: No outliers, less multicolinearity

In [28]:
```
X = df_no_fliers2.drop('pha', axis=1)
y = df_no_fliers2['pha']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r

X_train = X_train.drop('class_TNO', axis=1)
X_test = X_test.drop('class_TNO', axis=1)
```

In [29]:
```
log_pipe5 = Pipeline([('ss', StandardScaler()),
                      ('logreg', LogisticRegression(solver='liblinear', C=1e
log_pipe5.fit(X_train, y_train)
y_pred = log_pipe5.predict(X_test)
```

In [30]:
```
print('Train Report')
print(classification_report(y_train, log_pipe5.predict(X_train)))
print('\n')
print('Test Report')
print(classification_report(y_test, y_pred))
```

```
Train Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    689582
           1       0.41      1.00      0.58      1546

    accuracy                           1.00    691128
   macro avg       0.71      1.00      0.79    691128
weighted avg       1.00      1.00      1.00    691128



Test Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    229856
           1       0.43      1.00      0.60       520

    accuracy                           1.00    230376
   macro avg       0.71      1.00      0.80    230376
weighted avg       1.00      1.00      1.00    230376
```

Removing multicolinearity features seems to have improved the precision and f1-score of the positive class by 4%. After comparing the classification reports of Iteration 3 to those of a baseline desision tree, it seems that logistic regression will not be the best model for this dataset and I will proceed with other modeling techniques.

In [37]:
```
from matplotlib import rcParams
rcParams.update({'figure.autolayout': False})

sns.set_context('talk', font_scale=0.65)

plot_confusion_matrix(log_pipe5, X_test, y_test, cmap=plt.cm.Blues,
                      display_labels=['Non-Hazardous', 'Potentially Hazardo
plt.title('Logistic Regression Confusion Matrix')
plt.savefig('Images/LR-matrix.png', bbox_inches='tight');
```