

In [1]:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 %matplotlib inline
5 import seaborn as sns
6 from IPython import get_ipython
7 import warnings
8 warnings.filterwarnings("ignore")
```

In [2]:

```
1 data = pd.read_csv('nearest_earth_objects.csv')
```

In [3]:

```
1 data.head()
```

Out[3]:

	id	name	est_diameter_min	est_diameter_max	relative_velocity	miss_distance	orbit
0	2162635	162635 (2000 SS164)	1.198271	2.679415	13569.24922	54839744.08	
1	2277475	277475 (2005 WK4)	0.265800	0.594347	73588.72666	61438126.52	
2	2512244	512244 (2015 YE18)	0.722030	1.614507	114258.69210	49798724.94	
3	3596030	(2012 BV13)	0.096506	0.215794	24764.30314	25434972.72	
4	3667127	(2014 GE35)	0.255009	0.570217	42737.73376	46275567.00	

In [4]:



```
1 data.tail()
```

Out[4]:

	id	name	est_diameter_min	est_diameter_max	relative_velocity	miss_distance
90831	3763337	(2016 VX1)	0.026580	0.059435	52078.886690	12300389.18
90832	3837603	(2019 AD3)	0.016771	0.037501	46114.605070	54321206.42
90833	54017201	(2020 JP3)	0.031956	0.071456	7566.807732	28400768.16
90834	54115824	(2021 CN5)	0.007321	0.016370	69199.154480	68692060.53
90835	54205447	(2021 TW7)	0.039862	0.089133	27024.455550	59772130.59



In [5]:



```
1 data.duplicated().sum()
```

Out[5]:

0

In [6]:



```
1 data.isnull().sum()
```

Out[6]:

```
id          0
name        0
est_diameter_min  0
est_diameter_max  0
relative_velocity  0
miss_distance  0
orbiting_body  0
sentry_object  0
absolute_magnitude  0
hazardous    0
dtype: int64
```

In [7]:



```
1 data.shape
```

Out[7]:

```
(90836, 10)
```

In [8]:



```
1 data.columns
```

Out[8]:

```
Index(['id', 'name', 'est_diameter_min', 'est_diameter_max',  
      'relative_velocity', 'miss_distance', 'orbiting_body', 'sentry_object',  
      'absolute_magnitude', 'hazardous'],  
      dtype='object')
```

In [9]:



```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 90836 entries, 0 to 90835  
Data columns (total 10 columns):  
#   Column                Non-Null Count  Dtype    
---  -  
0   id                    90836 non-null  int64    
1   name                  90836 non-null  object   
2   est_diameter_min      90836 non-null  float64  
3   est_diameter_max      90836 non-null  float64  
4   relative_velocity     90836 non-null  float64  
5   miss_distance         90836 non-null  float64  
6   orbiting_body         90836 non-null  object   
7   sentry_object         90836 non-null  bool     
8   absolute_magnitude    90836 non-null  float64  
9   hazardous            90836 non-null  bool     
dtypes: bool(2), float64(5), int64(1), object(2)  
memory usage: 5.7+ MB
```

In [10]:

```
1 data.describe()
```

Out[10]:

	id	est_diameter_min	est_diameter_max	relative_velocity	miss_distance	abs
count	9.083600e+04	90836.000000	90836.000000	90836.000000	9.083600e+04	
mean	1.438288e+07	0.127432	0.284947	48066.918918	3.706655e+07	
std	2.087202e+07	0.298511	0.667491	25293.296961	2.235204e+07	
min	2.000433e+06	0.000609	0.001362	203.346432	6.745533e+03	
25%	3.448110e+06	0.019256	0.043057	28619.020648	1.721082e+07	
50%	3.748362e+06	0.048368	0.108153	44190.117890	3.784658e+07	
75%	3.884023e+06	0.143402	0.320656	62923.604635	5.654900e+07	
max	5.427591e+07	37.892650	84.730541	236990.128100	7.479865e+07	

In [11]:

```
1 data.nunique()
```

Out[11]:

```
id                27423
name              27423
est_diameter_min   1638
est_diameter_max   1638
relative_velocity  90785
miss_distance      90536
orbiting_body      1
sentry_object      1
absolute_magnitude 1638
hazardous          2
dtype: int64
```

In [12]:

```
1 data['hazardous'].unique()
```

Out[12]:

```
array([False,  True])
```

In [13]:



```
1 data['hazardous'].value_counts()
```

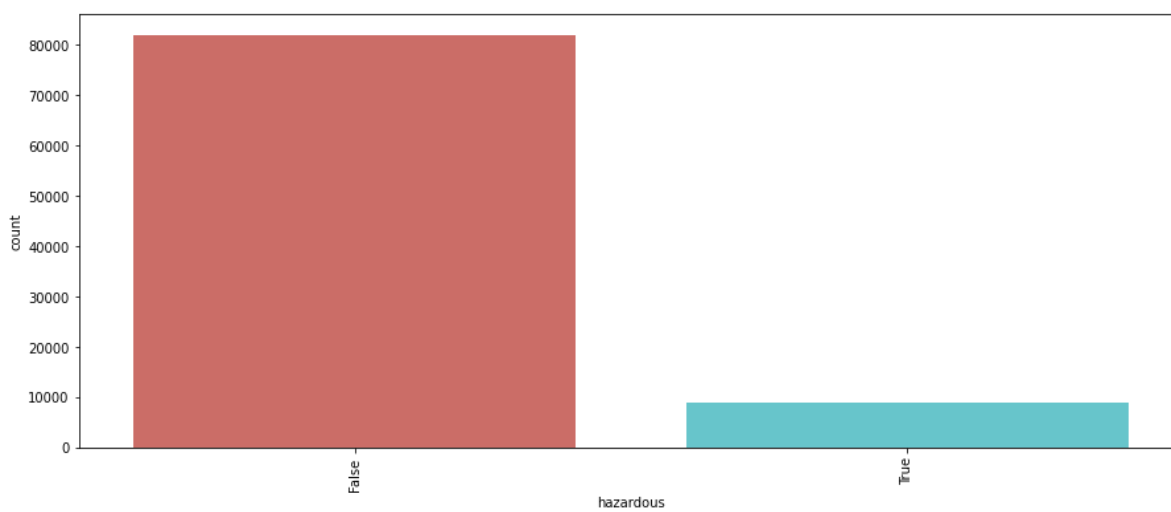
Out[13]:

```
False    81996  
True      8840  
Name: hazardous, dtype: int64
```

In [14]:



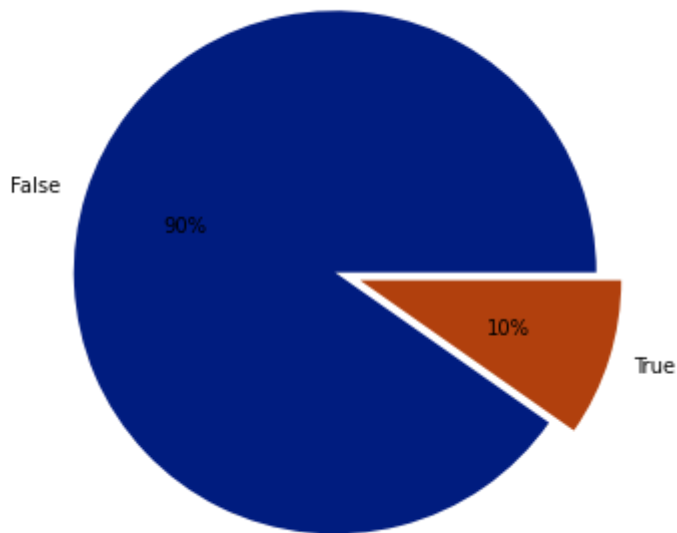
```
1 plt.figure(figsize=(15,6))  
2 sns.countplot('hazardous', data = data, palette = 'hls')  
3 plt.xticks(rotation = 90)  
4 plt.show()
```



In [16]:



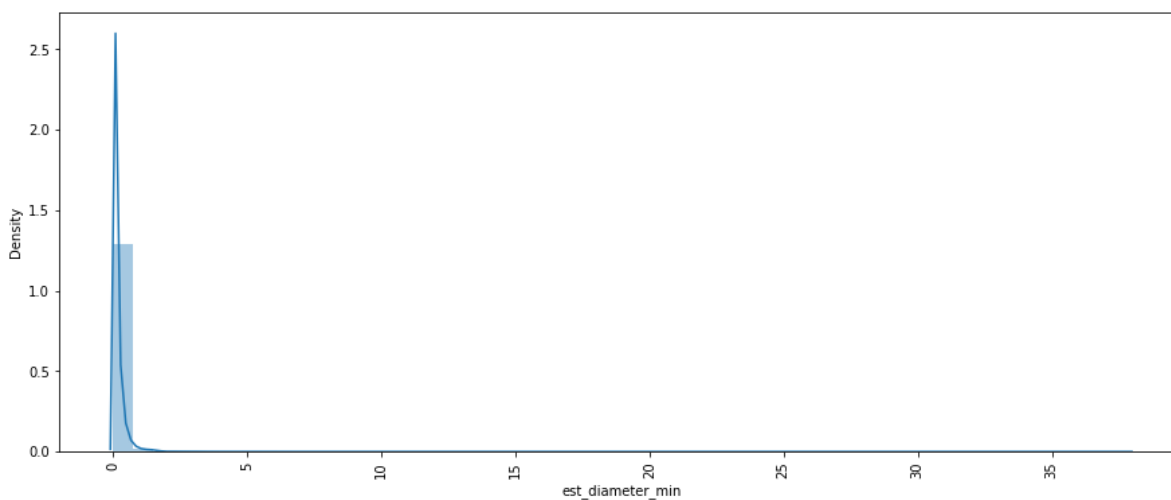
```
1 plt.figure(figsize=(15,6))
2 palette_color = sns.color_palette('dark')
3 explode = [0, 0.1]
4 plt.pie(data['hazardous'].value_counts(), labels= ['False', 'True'],
5         colors=palette_color, explode=explode,
6         autopct='%.0f%%')
7 plt.show()
```



In [17]:



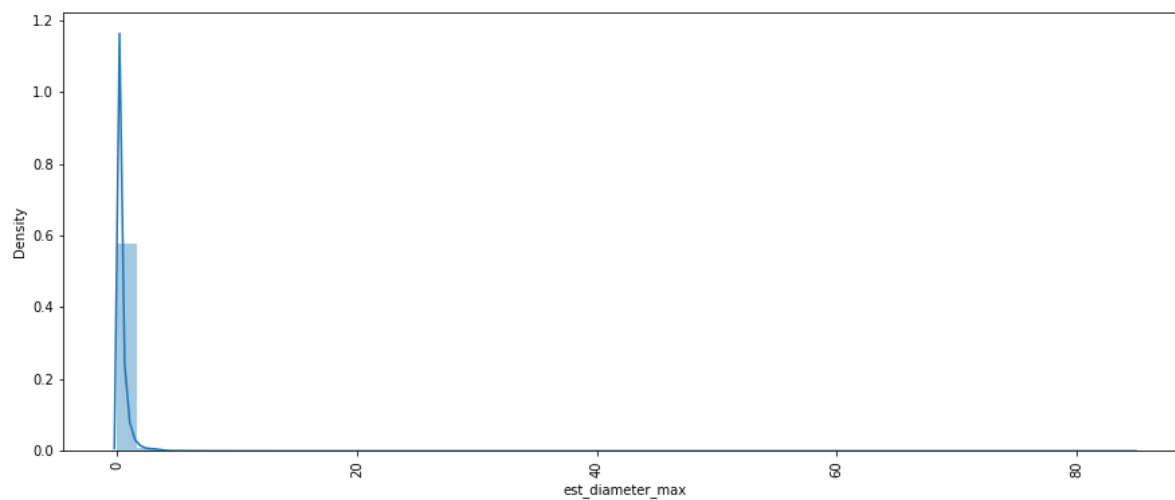
```
1 plt.figure(figsize=(15,6))
2 sns.distplot(data['est_diameter_min'])
3 plt.xticks(rotation = 90)
4 plt.show()
```



In [18]:



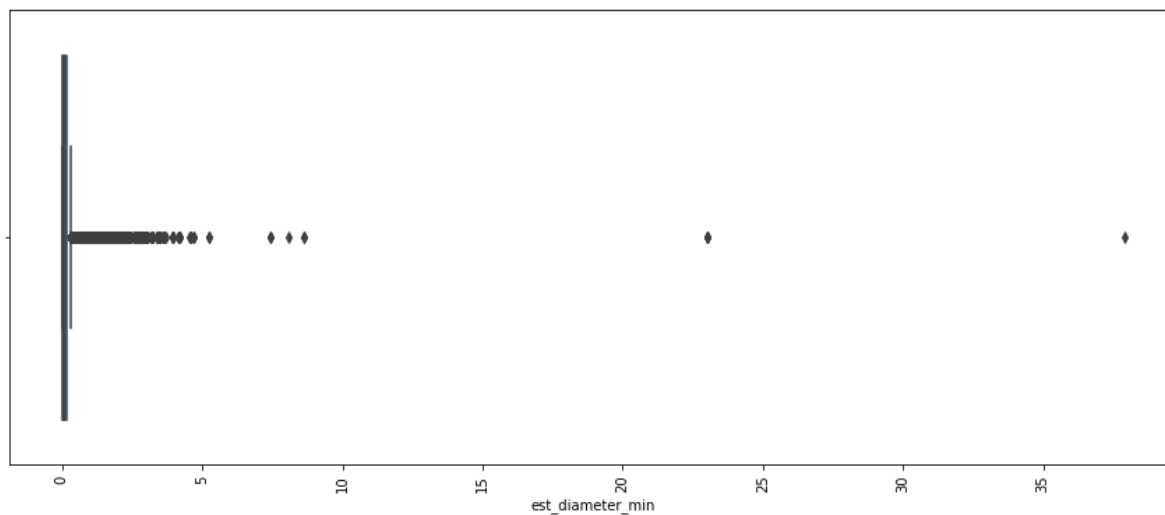
```
1 plt.figure(figsize=(15,6))
2 sns.distplot(data['est_diameter_max'])
3 plt.xticks(rotation = 90)
4 plt.show()
```



In [19]:

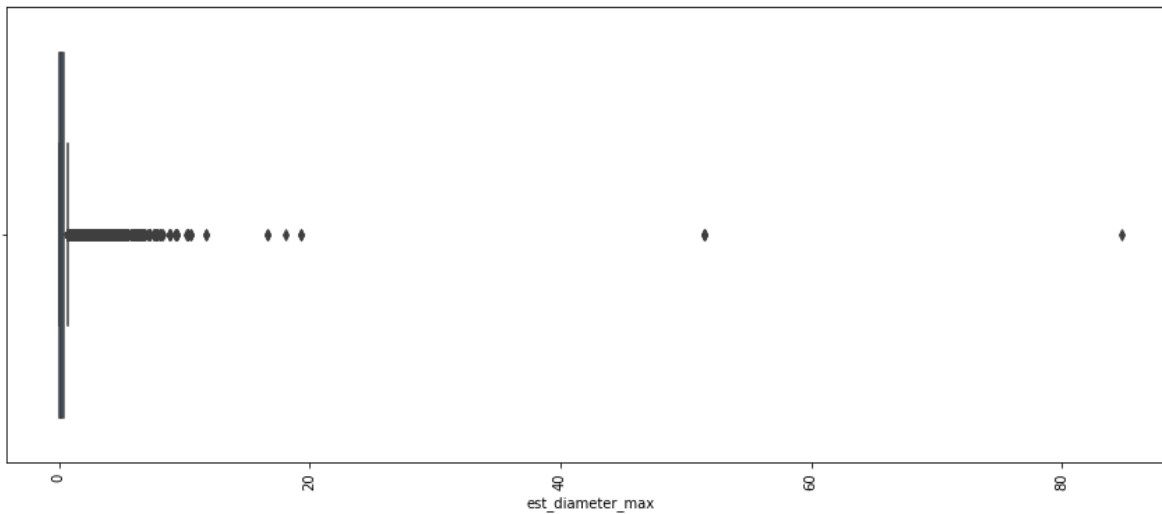


```
1 plt.figure(figsize=(15,6))
2 sns.boxplot(data['est_diameter_min'])
3 plt.xticks(rotation = 90)
4 plt.show()
```



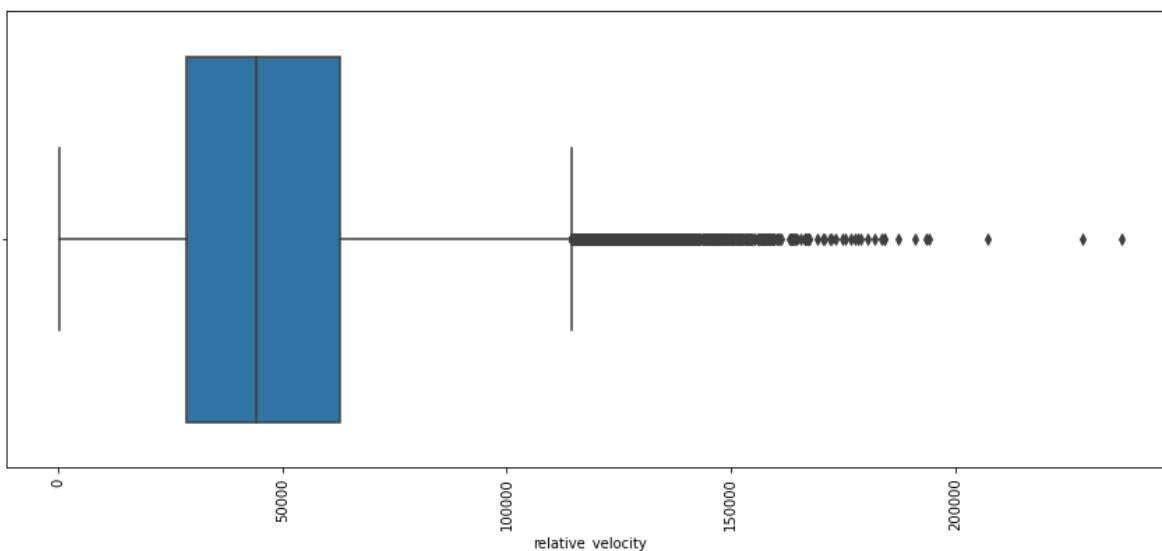
In [20]:

```
1 plt.figure(figsize=(15,6))
2 sns.boxplot(data['est_diameter_max'])
3 plt.xticks(rotation = 90)
4 plt.show()
```



In [21]:

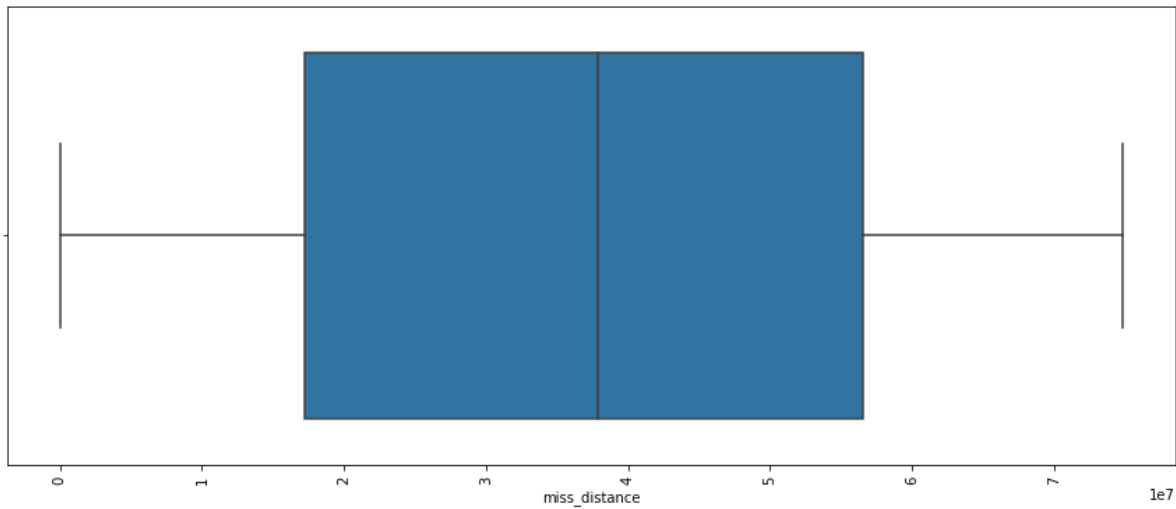
```
1 plt.figure(figsize=(15,6))
2 sns.boxplot(data['relative_velocity'])
3 plt.xticks(rotation = 90)
4 plt.show()
```



In [22]:



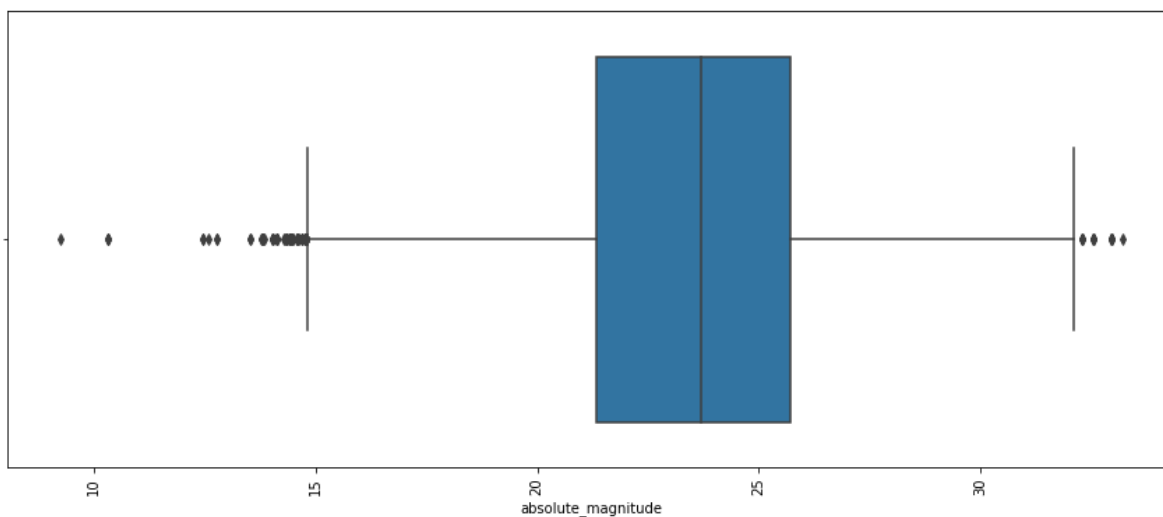
```
1 plt.figure(figsize=(15,6))
2 sns.boxplot(data['miss_distance'])
3 plt.xticks(rotation = 90)
4 plt.show()
```



In [23]:



```
1 plt.figure(figsize=(15,6))
2 sns.boxplot(data['absolute_magnitude'])
3 plt.xticks(rotation = 90)
4 plt.show()
```



In [26]:



```
1 from contextlib import contextmanager
2 from time import time
3 from tqdm import tqdm
4 import lightgbm as lgbm
5 import category_encoders as ce
6
7 from tensorflow.keras.utils import to_categorical
8 from sklearn.metrics import classification_report, log_loss, accuracy_score
9 from sklearn.metrics import mean_squared_error
10 from sklearn.model_selection import KFold
```

In [27]:



```
1 from sklearn.preprocessing import LabelEncoder
2
3 def labelencoder(df):
4     for c in df.columns:
5         if df[c].dtype=='object':
6             df[c] = df[c].fillna('N')
7             lbl = LabelEncoder()
8             lbl.fit(list(df[c].values))
9             df[c] = lbl.transform(df[c].values)
10     return df
```

In [28]:



```
1 data1=labelencoder(data)
```

In [30]:



```
1 import random
```

In [31]:



```
1 m=len(data1)
2 M=list(range(m))
3 random.seed(2021)
4 random.shuffle(M)
```

In [39]:



```
1 target=['hazardous']
2 dataY=data1[target]
3 dataX=data1.drop(target,axis=1)
```

In [40]:



```
1 dataX.shape
```

Out[40]:

(90836, 9)

In [41]:

```
1 dataY.shape
```

Out[41]:

```
(90836, 1)
```

In [42]:

```
1 dataX.columns
```

Out[42]:

```
Index(['id', 'name', 'est_diameter_min', 'est_diameter_max',  
      'relative_velocity', 'miss_distance', 'orbiting_body', 'sentry_object',  
      'absolute_magnitude'],  
      dtype='object')
```

In [43]:

```
1 df_columns = list(dataX.columns)
```

In [44]:

```
1 trainX=dataX.iloc[M[0:(m//4)*3]]  
2 trainY=dataY.iloc[M[0:(m//4)*3]]  
3 testX=dataX.iloc[M[(m//4)*3:]]  
4 testY=dataY.iloc[M[(m//4)*3:]]
```

In [45]:

```
1 train_df=trainX  
2 test_df=testX
```

In [46]:

```
1 df_columns = list(dataX.columns)  
2 train_df.columns=df_columns  
3 test_df.columns=df_columns
```

In [47]:

```
1 def create_numeric_feature(input_df):  
2     use_columns = df_columns  
3     return input_df[use_columns].copy()
```

In [48]:



```
1 from contextlib import contextmanager
2 from time import time
3
4 class Timer:
5     def __init__(self, logger=None, format_str='{:.3f}[s]', prefix=None, suffix=None):
6
7         if prefix: format_str = str(prefix) + sep + format_str
8         if suffix: format_str = format_str + sep + str(suffix)
9         self.format_str = format_str
10        self.logger = logger
11        self.start = None
12        self.end = None
13
14        @property
15        def duration(self):
16            if self.end is None:
17                return 0
18            return self.end - self.start
19
20        def __enter__(self):
21            self.start = time()
22
23        def __exit__(self, exc_type, exc_val, exc_tb):
24            self.end = time()
25            out_str = self.format_str.format(self.duration)
26            if self.logger:
27                self.logger.info(out_str)
28            else:
29                print(out_str)
```

In [49]:



```
1 from tqdm import tqdm
2
3 def to_feature(input_df):
4
5     processors = [
6         create_numeric_feature,
7     ]
8
9     out_df = pd.DataFrame()
10
11     for func in tqdm(processors, total=len(processors)):
12         with Timer(prefix='create' + func.__name__ + ' '):
13             _df = func(input_df)
14
15         assert len(_df) == len(input_df), func.__name__
16         out_df = pd.concat([out_df, _df], axis=1)
17
18     return out_df
```

In [50]:



```
1 train_feat_df = to_feature(train_df)
2 test_feat_df = to_feature(test_df)
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 1/1 [00:00<00:00, 17.85
it/s]
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 1/1 [00:00<00:00, 125.04
it/s]
```

```
createcreate_numeric_feature 0.048[s]
createcreate_numeric_feature 0.000[s]
```

In [51]:



```
1 import lightgbm as lgbm
2 from sklearn.metrics import mean_squared_error
3
4 def fit_lgbm(X, y, cv,
5             params: dict=None,
6             verbose: int=50):
7
8     if params is None:
9         params = {}
10
11     models = []
12     oof_pred = np.zeros_like(y, dtype=np.float)
13
14     for i, (idx_train, idx_valid) in enumerate(cv):
15         x_train, y_train = X[idx_train], y[idx_train]
16         x_valid, y_valid = X[idx_valid], y[idx_valid]
17
18         clf = lgbm.LGBMRegressor(**params)
19
20         with Timer(prefix='fit fold={}' .format(i)):
21             clf.fit(x_train, y_train,
22                   eval_set=[(x_valid, y_valid)],
23                   early_stopping_rounds=100,
24                   verbose=verbose)
25
26         pred_i = clf.predict(x_valid)
27         oof_pred[idx_valid] = pred_i
28         models.append(clf)
29         print(f'Fold {i} RMSLE: {mean_squared_error(y_valid, pred_i) ** .5:.4f}')
30         print()
31
32     score = mean_squared_error(y, oof_pred) ** .5
33     print('-' * 50)
34     print('FINISHED | Whole RMSLE: {:.4f}'.format(score))
35     return oof_pred, models
```

In [52]:



```
1 params = {
2     'objective': 'rmse',
3     'learning_rate': .1,
4     'reg_lambda': 1.,
5     'reg_alpha': .1,
6     'max_depth': 5,
7     'n_estimators': 10000,
8     'colsample_bytree': .5,
9     'min_child_samples': 10,
10    'subsample_freq': 3,
11    'subsample': .9,
12    'importance_type': 'gain',
13    'random_state': 71,
14    'num_leaves': 62
15 }
```

In [53]:



```
1 y = trainY
2 ydf=pd.DataFrame(y)
3 ydf
```

Out[53]:

hazardous	
55817	False
65066	False
24247	False
71521	True
50888	False
...	...
86721	False
89457	False
25957	True
9203	False
45194	False

68127 rows × 1 columns

In [54]:



```

1 from sklearn.model_selection import KFold
2
3 for i in range(1):
4     fold = KFold(n_splits=5, shuffle=True, random_state=71)
5     ydfi=ydf.iloc[:,i]
6     y=np.array(ydfi)
7     cv = list(fold.split(train_feat_df, y))
8     oof, models = fit_lgbm(train_feat_df.values, y, cv, params=params, verbose=500)
9
10    fig,ax = plt.subplots(figsize=(6,6))
11    ax.set_title(target[i],fontsize=20)
12    ax.set_xlabel('predicted',fontsize=12)
13    ax.set_ylabel('actual',fontsize=12)
14    ax.scatter(oof,y)

```

```

[500]    valid_0's rmse: 0.208091
[1000]    valid_0's rmse: 0.199212
[1500]    valid_0's rmse: 0.19451
[2000]    valid_0's rmse: 0.191958
[2500]    valid_0's rmse: 0.189737
[3000]    valid_0's rmse: 0.188604
[3500]    valid_0's rmse: 0.187707
[4000]    valid_0's rmse: 0.186799
fit fold=0  16.076[s]
Fold 0 RMSLE: 0.1867

```

```

[500]    valid_0's rmse: 0.212186
[1000]    valid_0's rmse: 0.203067
[1500]    valid_0's rmse: 0.198317
[2000]    valid_0's rmse: 0.195867
[2500]    valid_0's rmse: 0.194005
[3000]    valid_0's rmse: 0.192648
[3500]    valid_0's rmse: 0.191834
[4000]    valid_0's rmse: 0.191206
fit fold=1  15.216[s]
Fold 1 RMSLE: 0.1910

```

```

[500]    valid_0's rmse: 0.21388
[1000]    valid_0's rmse: 0.205295
[1500]    valid_0's rmse: 0.200106
[2000]    valid_0's rmse: 0.196901
[2500]    valid_0's rmse: 0.195201
[3000]    valid_0's rmse: 0.194026
[3500]    valid_0's rmse: 0.193056
[4000]    valid_0's rmse: 0.19239
fit fold=2  16.449[s]
Fold 2 RMSLE: 0.1922

```

```

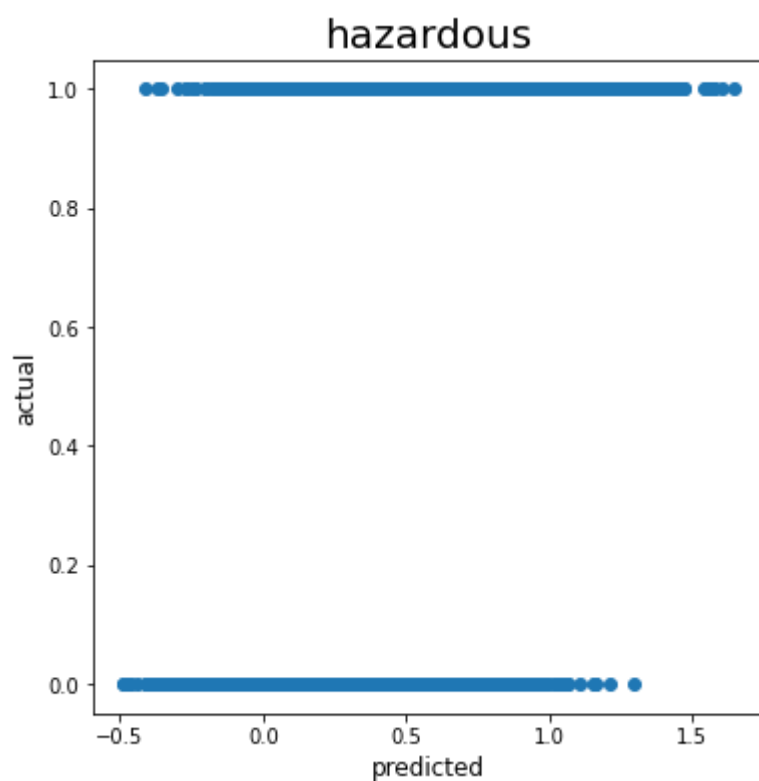
[500]    valid_0's rmse: 0.213621
[1000]    valid_0's rmse: 0.20372
[1500]    valid_0's rmse: 0.197921
[2000]    valid_0's rmse: 0.194829
[2500]    valid_0's rmse: 0.192182
[3000]    valid_0's rmse: 0.190936
[3500]    valid_0's rmse: 0.189823
[4000]    valid_0's rmse: 0.188969

```

```
fit fold=3 13.456[s]  
Fold 3 RMSLE: 0.1889
```

```
[500] valid_0's rmse: 0.210971  
[1000] valid_0's rmse: 0.202455  
[1500] valid_0's rmse: 0.197414  
[2000] valid_0's rmse: 0.194025  
[2500] valid_0's rmse: 0.192288  
[3000] valid_0's rmse: 0.190982  
[3500] valid_0's rmse: 0.190223  
[4000] valid_0's rmse: 0.189702  
fit fold=4 13.405[s]  
Fold 4 RMSLE: 0.1896
```

FINISHED | Whole RMSLE: 0.1897



In [55]:



```
1 def visualize_importance(models, feat_train_df):
2
3     feature_importance_df = pd.DataFrame()
4     for i, model in enumerate(models):
5         _df = pd.DataFrame()
6         _df['feature_importance'] = model.feature_importances_
7         _df['column'] = feat_train_df.columns
8         _df['fold'] = i + 1
9         feature_importance_df = pd.concat([feature_importance_df, _df],
10                                           axis=0, ignore_index=True)
11
12     order = feature_importance_df.groupby('column')\
13         .sum()[['feature_importance']]\
14         .sort_values('feature_importance', ascending=False).index[:50]
15
16     fig, ax = plt.subplots(figsize=(8, max(6, len(order) * .25)))
17     sns.boxenplot(data=feature_importance_df,
18                  x='feature_importance',
19                  y='column',
20                  order=order,
21                  ax=ax,
22                  palette='viridis',
23                  orient='h')
24
25     ax.tick_params(axis='x', rotation=0)
26     #ax.set_title('Importance')
27     ax.grid()
28     fig.tight_layout()
29
30     return fig, ax
```

In [56]:



```
1 for i in range(1):
2     fold = KFold(n_splits=5, shuffle=True, random_state=71)
3     ydfi=ydf.iloc[:,i]
4     y=np.array(ydfi)
5     cv = list(fold.split(train_feat_df, y))
6     oof, models = fit_lgbm(train_feat_df.values, y, cv, params=params, verbose=500)
7     fig, ax = visualize_importance(models, train_feat_df)
8     ax.set_title(target[i]+' Imortance',fontsize=20)
```

```
[500]    valid_0's rmse: 0.208091
[1000]    valid_0's rmse: 0.199212
[1500]    valid_0's rmse: 0.19451
[2000]    valid_0's rmse: 0.191958
[2500]    valid_0's rmse: 0.189737
[3000]    valid_0's rmse: 0.188604
[3500]    valid_0's rmse: 0.187707
[4000]    valid_0's rmse: 0.186799
fit fold=0 16.436[s]
Fold 0 RMSLE: 0.1867
```

```
[500]    valid_0's rmse: 0.212186
[1000]    valid_0's rmse: 0.203067
[1500]    valid_0's rmse: 0.198317
[2000]    valid_0's rmse: 0.195867
[2500]    valid_0's rmse: 0.194005
[3000]    valid_0's rmse: 0.192648
[3500]    valid_0's rmse: 0.191834
[4000]    valid_0's rmse: 0.191206
fit fold=1 14.439[s]
Fold 1 RMSLE: 0.1910
```

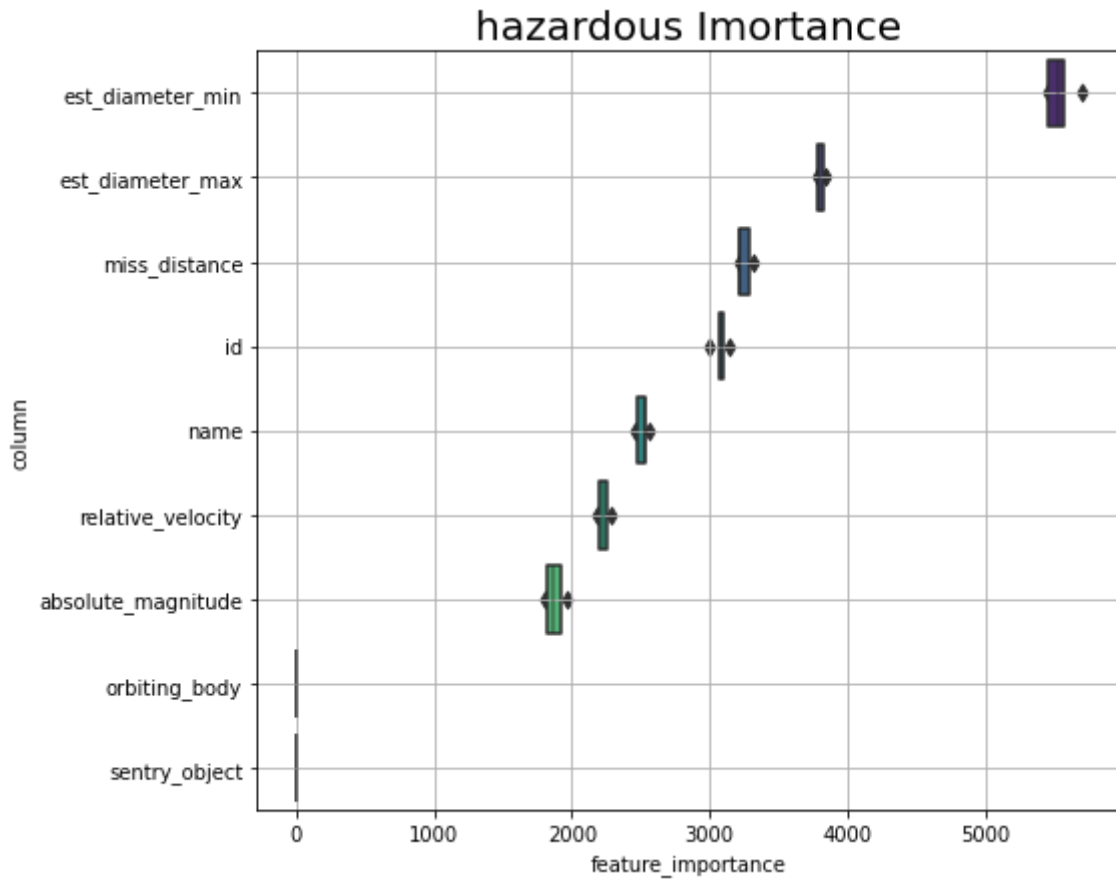
```
[500]    valid_0's rmse: 0.21388
[1000]    valid_0's rmse: 0.205295
[1500]    valid_0's rmse: 0.200106
[2000]    valid_0's rmse: 0.196901
[2500]    valid_0's rmse: 0.195201
[3000]    valid_0's rmse: 0.194026
[3500]    valid_0's rmse: 0.193056
[4000]    valid_0's rmse: 0.19239
fit fold=2 15.624[s]
Fold 2 RMSLE: 0.1922
```

```
[500]    valid_0's rmse: 0.213621
[1000]    valid_0's rmse: 0.20372
[1500]    valid_0's rmse: 0.197921
[2000]    valid_0's rmse: 0.194829
[2500]    valid_0's rmse: 0.192182
[3000]    valid_0's rmse: 0.190936
[3500]    valid_0's rmse: 0.189823
[4000]    valid_0's rmse: 0.188969
fit fold=3 13.511[s]
Fold 3 RMSLE: 0.1889
```

```
[500]    valid_0's rmse: 0.210971
[1000]    valid_0's rmse: 0.202455
[1500]    valid_0's rmse: 0.197414
[2000]    valid_0's rmse: 0.194025
```

```
[2500] valid_0's rmse: 0.192288
[3000] valid_0's rmse: 0.190982
[3500] valid_0's rmse: 0.190223
[4000] valid_0's rmse: 0.189702
fit fold=4 14.723[s]
Fold 4 RMSLE: 0.1896
```

FINISHED | Whole RMSLE: 0.1897



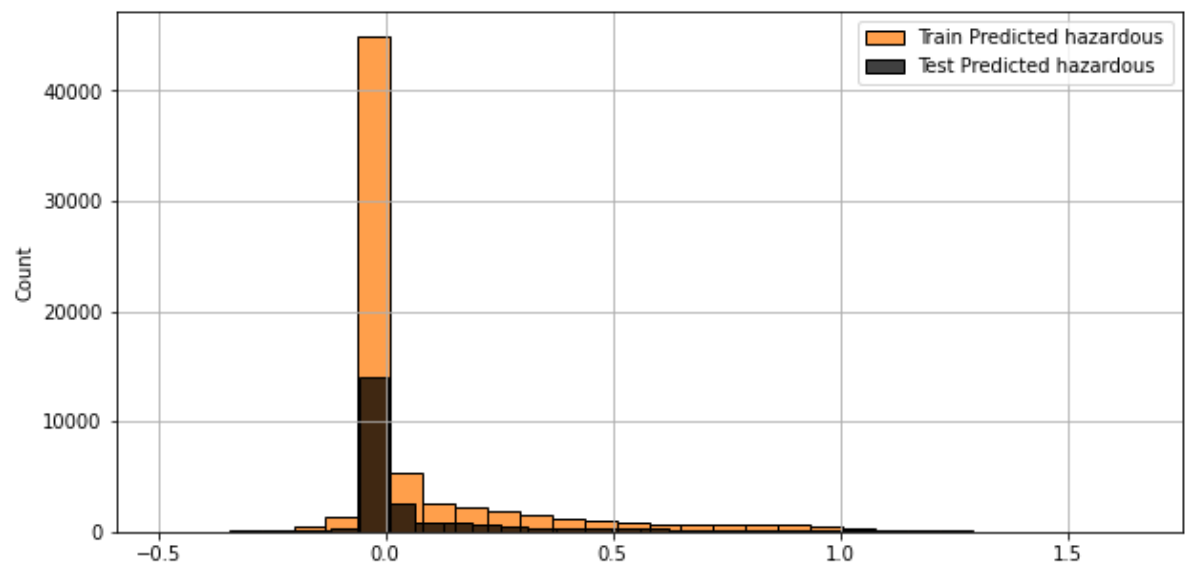
In [57]:

```
1 preds=[]
2 for i in range(5):
3     preds += [models[i].predict(test_feat_df.values)/5]
4 predsT=np.array(preds).T
5 preds2=[]
6 preds3=[]
7 for item in predsT:
8     value=sum(item)
9     preds2+= [value]
10    preds3+= [int(np.where(value<0.5,0,1))]
11 print(preds2[0:5])
12 print(preds3[0:5])
```

```
[-0.002899381794562423, 0.0005748616101924781, 0.3833633806196074, 0.00013
233605794080955, 0.8473538524387714]
[0, 0, 0, 0, 1]
```

In [58]:

```
1 for i in range(1):
2     fig, ax = plt.subplots(figsize=(10,5))
3     sns.histplot(oof, label='Train Predicted '+target[i], ax=ax, color='C1',bins=30
4     sns.histplot(preds2, label='Test Predicted '+target[i], ax=ax, color='black',bi
5     ax.legend()
6     ax.grid()
```



In [59]:

```
1 from sklearn.metrics import classification_report
2 print(classification_report(testY,preds3))
```

	precision	recall	f1-score	support
False	0.97	0.99	0.98	20496
True	0.86	0.69	0.77	2213
accuracy			0.96	22709
macro avg	0.91	0.84	0.87	22709
weighted avg	0.96	0.96	0.96	22709

In []:

```
1
```