In [1]:
```python
# data analysis and wrangling
#https://www.kaggle.com/startupsci/titanic-data-science-solutions

import pandas as pd
import numpy as np
import random as rnd

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```python
#Load the data
aviation = pd.read_csv('GeneralAviationUSLLClean.csv',encoding='latin1',low_memory=False)
aviation.shape
```

Out[2]:
```
(74462, 19)
```

In [3]:
```python
#Split into train and test set
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

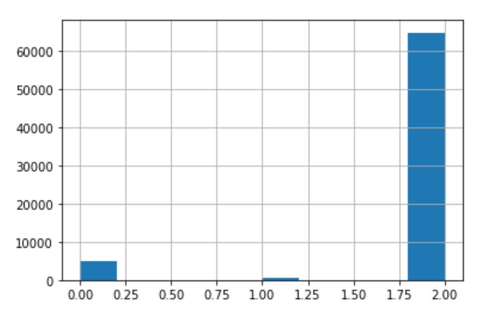
In [4]:
```python
df = aviation[['WeatherCondition','BroadPhaseofFlight','PurposeofFlight', "Lethality"]]
df = df.dropna()
df["Wx"]= df["WeatherCondition"].astype('category')
df["Phase"]= df["BroadPhaseofFlight"].astype('category')
df["Purpose"]= df["PurposeofFlight"].astype('category')

#Categories CWx indicates weather condition. CPhase indicates phase in flight accident occurred. CPurpose indicates the purpose of fight.
df["CWx"]= df["Wx"].cat.codes
df["CPhase"]= df["Phase"].cat.codes
df["CPurpose"]= df["Purpose"].cat.codes
```

In [5]:
```python
df_final = df[["Lethality",'CWx','CPhase','CPurpose']]
df_final.dtypes
```

Out[5]:
```
Lethality    float64
CWx             int8
CPhase          int8
CPurpose        int8
dtype: object
```

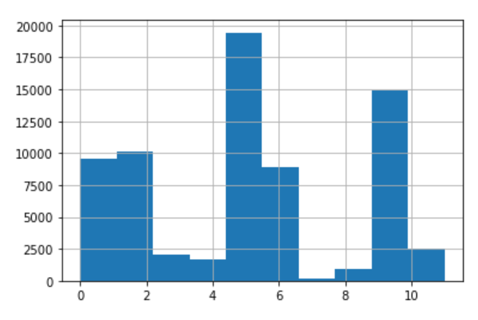
In [6]:
```python
#Weather condition
df_final["CWx"].hist()
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fca2999b550>
```



In [7]:
```python
#Phase of flight accident occurred
df_final["CPhase"].hist()
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fca299b5a50>
```



In [8]:
```python
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
df.shape, train_df, test_df
```

Out[8]:
```
((70234, 10),
       WeatherCondition BroadPhaseofFlight     PurposeofFlight  Lethality   Wx  \
 19366              UNK           STANDING            Personal        0.0  UNK
 61271              VMC            TAKEOFF            Personal        0.0  VMC
 6759               IMC             CRUISE            Personal        1.0  IMC
 3525               VMC        MANEUVERING            Personal        1.0  VMC
 27641              VMC        MANEUVERING       Instructional        0.0  VMC
 ...                ...                ...                 ...        ...  ...
 38888              VMC            LANDING  Aerial Application        0.0  VMC
 7253               VMC        MANEUVERING  Aerial Observation        1.0  VMC
 57182              VMC        MANEUVERING      Other Work Use        0.0  VMC
 1269               IMC        MANEUVERING             Unknown        1.0  IMC
 17419              VMC            TAKEOFF            Personal        0.0  VMC

              Phase             Purpose  CWx  CPhase  CPurpose
 19366     STANDING            Personal    1       8        14
 61271      TAKEOFF            Personal    2       9        14
 6759        CRUISE            Personal    0       2        14
 3525   MANEUVERING            Personal    2       6        14
 27641  MANEUVERING       Instructional    2       6        12
 ...            ...                 ...  ...     ...       ...
 38888      LANDING  Aerial Application    2       5         0
 7253   MANEUVERING  Aerial Observation    2       6         1
 57182  MANEUVERING      Other Work Use    2       6        13
 1269   MANEUVERING             Unknown    0       6        21
 17419      TAKEOFF            Personal    2       9        14

 [56187 rows x 10 columns],
       WeatherCondition BroadPhaseofFlight PurposeofFlight  Lethality   Wx  \
 21686              VMC             CRUISE        Personal        0.0  VMC
 22499              VMC             CRUISE        Personal        0.0  VMC
 61791              VMC            LANDING        Personal        0.0  VMC
 13305              VMC            TAKEOFF        Personal        1.0  VMC
 69125              VMC            LANDING        Personal        0.0  VMC
 ...                ...                ...             ...        ...  ...
 8999               VMC        MANEUVERING        Personal        1.0  VMC
 30122              VMC            TAKEOFF        Business        0.0  VMC
 3903               VMC        MANEUVERING   Instructional        1.0  VMC
 27889              VMC            LANDING        Personal        0.0  VMC
 6141               VMC        MANEUVERING  Other Work Use        1.0  VMC

              Phase         Purpose  CWx  CPhase  CPurpose
 21686       CRUISE        Personal    2       2        14
 22499       CRUISE        Personal    2       2        14
 61791      LANDING        Personal    2       5        14
 13305      TAKEOFF        Personal    2       9        14
 69125      LANDING        Personal    2       5        14
 ...            ...             ...  ...     ...       ...
 8999   MANEUVERING        Personal    2       6        14
 30122      TAKEOFF        Business    2       9         5
 3903   MANEUVERING   Instructional    2       6        12
 27889      LANDING        Personal    2       5        14
 6141   MANEUVERING  Other Work Use    2       6        13

 [14047 rows x 10 columns])
```

In [9]:
```python
train_rg = train_df[['Lethality','CWx','CPhase','CPurpose']]
X_train = train_df[['CWx','CPhase','CPurpose']]
Y_train = train_df["Lethality"]
X_test  = test_df[['CWx','CPhase','CPurpose']]
Y_test  = test_df[['Lethality']]
X_train.shape, Y_train.shape, X_test.shape
```

Out[9]:
```
((56187, 3), (56187,), (14047, 3))
```

In [10]:
```python
from sklearn.model_selection import cross_val_score
X=df_final[['CWx','CPhase','CPurpose']]
y=df_final['Lethality']
```

In [11]:
```python
from sklearn.metrics import mean_absolute_error, average_precision_score
from sklearn.metrics import recall_score, f1_score, mean_squared_error

def rmse(Y_test, Y_pred):
    return
    np.sqrt(((Y_test - Y_pred) ** 2).mean())

def display_scores(Y_test, Y_pred):
    mae=mean_absolute_error(Y_test, Y_pred)
    precision=average_precision_score(Y_test, Y_pred)
    recall=recall_score(Y_test, Y_pred)
    f1=f1_score(Y_test, Y_pred)

    print("MAE:", mae)
    print("Average Precision Score:", precision)
    print("Recall:", recall)
    print("F1:", f1)

def display_cross_val_scores(scores):
    print("Cross Validation Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

In [12]:
```python
# Linear Regresion
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
Y_pred = logreg.predict(X_test)
acc_log = round(logreg.score(X_train, Y_train) * 100, 2)
acc_test_log = round(logreg.score(X_test, Y_test) * 100, 2)
print("Accuracy on Training Set:", acc_log)
print("Accuracy on Test Set:", acc_test_log)
```

```
Accuracy on Training Set: 82.23
Accuracy on Test Set: 82.35
```

In [13]:
```python
#Display SCore for LogisticalRegresssion
display_scores(Y_test, Y_pred)
```

```
MAE: 0.17647896347974656
Average Precision Score: 0.2838375599096816
Recall: 0.22147900408769974
F1: 0.3247071642604195
```

In [14]:
```python
# Cross validation Logistical Regression
scores = cross_val_score(logreg,X,y,scoring="neg_mean_squared_error",cv=5)
tree_rmse_scores = np.sqrt(-scores)
display_cross_val_scores(tree_rmse_scores)
```

```
Cross Validation Scores: [0.42556584 0.40884365 0.43038969 0.42085588 0.42044775]
Mean: 0.42122055979183515
Standard deviation: 0.0071676010189134715
```

In [15]:
```python
# Corelation Coefficient
coeff_df = pd.DataFrame(train_rg.columns.delete(0))
coeff_df.columns = ['Feature']
coeff_df["Correlation"] = pd.Series(logreg.coef_[0])
coeff_df.sort_values(by='Correlation', ascending=False)
```

Out[15]:

	Feature	Correlation
2	CPurpose	0.019924
1	CPhase	-0.026967
0	CWx	-1.073675

In [16]:
```python
svc = SVC()
svc.fit(X_train, Y_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_train, Y_train) * 100, 2)
acc_test_svc = round(svc.score(X_test, Y_test) * 100, 2)
print("Accuracy on Training Set:", acc_svc)
print("Accuracy on Test Set:", acc_test_svc)
```

```
Accuracy on Training Set: 82.42
Accuracy on Test Set: 82.64
```

In [17]:
```python
#Display SCore for SVC
display_scores(Y_test, Y_pred)
```

```
MAE: 0.17363138036591444
Average Precision Score: 0.28896274735752214
Recall: 0.2192493496841323
F1: 0.3260569218016026
```

In [18]:
```python
# Cross validation SVC
scores = cross_val_score(svc,X,y,scoring="neg_mean_squared_error",cv=5)
tree_rmse_scores = np.sqrt(-scores)
display_cross_val_scores(tree_rmse_scores)
```

```
Cross Validation Scores: [0.4231333  0.4060481  0.42964471 0.42009399 0.42103998]
Mean: 0.41999201775546824
Standard deviation: 0.007727657605489291
```

In [19]:
```python
#KNN
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
acc_test_knn = round(knn.score(X_test, Y_test) * 100, 2)
print("Accuracy on Training Set:", acc_knn)
print("Accuracy on Test Set:", acc_test_knn)
```

```
Accuracy on Training Set: 80.03
Accuracy on Test Set: 80.34
```

In [20]:
```python
#Display SCore for KNN
display_scores(Y_test, Y_pred)
```

```
MAE: 0.19662561401010892
Average Precision Score: 0.2744333714959445
Recall: 0.28911185432924563
F1: 0.3603520148216767
```

In [21]:
```python
# Cross validation KNN
scores = cross_val_score(knn,X,y,scoring="neg_mean_squared_error",cv=5)
tree_rmse_scores = np.sqrt(-scores)
display_cross_val_scores(tree_rmse_scores)
```

```
Cross Validation Scores: [0.47324667 0.38940185 0.42656835 0.47204171 0.43892222]
Mean: 0.4400361611017212
Standard deviation: 0.031220984853190706
```

In [22]:
```python
# Gaussian
gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_train, Y_train) * 100, 2)
acc_test_gaussian = round(gaussian.score(X_test, Y_test) * 100, 2)
print("Accuracy on Training Set:", acc_gaussian)
print("Accuracy on Test Set:", acc_test_gaussian)
display_scores(Y_test, Y_pred)
```

```
Accuracy on Training Set: 82.38
Accuracy on Test Set: 82.53
MAE: 0.17469922403360147
Average Precision Score: 0.2937361679811194
Recall: 0.24414715719063546
F1: 0.34872611464968156
```

In [23]:
```python
# Cross validation Gaussian
scores = cross_val_score(gaussian,X,y,scoring="neg_mean_squared_error",cv=5)
tree_rmse_scores = np.sqrt(-scores)
display_cross_val_scores(tree_rmse_scores)
```

```
Cross Validation Scores: [0.42606739 0.40517054 0.43113338 0.41497902 0.41968507]
Mean: 0.41940707942567046
Standard deviation: 0.008991438461695403
```

In [24]:
```python
# Perceptron
perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
Y_pred = perceptron.predict(X_test)
acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)
acc_test_perceptron = round(perceptron.score(X_test, Y_test) * 100, 2)
print("Accuracy on Training Set:", acc_perceptron)
print("Accuracy on Test Set:", acc_test_perceptron)
display_scores(Y_test, Y_pred)
```

```
Accuracy on Training Set: 80.72
Accuracy on Test Set: 81.0
MAE: 0.1900049832704492
Average Precision Score: 0.2038823253821847
Recall: 0.032329988851727984
F1: 0.061202954625395704
```

In [25]:
```python
# Cross validation Perceptron
scores = cross_val_score(perceptron,X,y,scoring="neg_mean_squared_error",cv=5)
tree_rmse_scores = np.sqrt(-scores)
display_cross_val_scores(tree_rmse_scores)
```

```
Cross Validation Scores: [0.43995961 0.47317145 0.43096822 0.41472162 0.43940856]
Mean: 0.4396458925708432
Standard deviation: 0.019082134492139628
```

In [27]:
```python
# SGD
sgd = SGDClassifier()
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)
acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)
acc_test_sgd = round(sgd.score(X_test, Y_test) * 100, 2)
print("Accuracy on Training Set:", acc_sgd)
print("Accuracy on Test Set:", acc_test_sgd)
display_scores(Y_test, Y_pred)
```

```
Accuracy on Training Set: 82.22
Accuracy on Test Set: 82.35
MAE: 0.17647896347974656
Average Precision Score: 0.2838375599096816
Recall: 0.22147900408769974
F1: 0.3247071642604195
```

In [28]:
```python
# Cross validation SGD
scores = cross_val_score(sgd,X,y,scoring="neg_mean_squared_error",cv=5)
tree_rmse_scores = np.sqrt(-scores)
display_cross_val_scores(tree_rmse_scores)
```

```
Cross Validation Scores: [0.42556584 0.40525838 0.43038969 0.42085588 0.41968507]
Mean: 0.4203509706640066
Standard deviation: 0.00844340311204491
```

In [29]:
```python
# Decision Tree
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_train, Y_train) * 100, 2)
acc_test_decision_tree = round(decision_tree.score(X_test, Y_test) * 100, 2)
print("Accuracy on Training Set:", acc_decision_tree)
print("Accuracy on Test Set:", acc_test_decision_tree)
display_scores(Y_test, Y_pred)
```

```
Accuracy on Training Set: 84.09
Accuracy on Test Set: 84.12
MAE: 0.15882394817398732
Average Precision Score: 0.3745373677666026
Recall: 0.41917502787068006
F1: 0.502785825718743
```

In [30]:
```python
# Cross validation Decision Tree
scores = cross_val_score(decision_tree,X,y,scoring="neg_mean_squared_error",cv=5)
tree_rmse_scores = np.sqrt(-scores)
display_cross_val_scores(tree_rmse_scores)
```

```
Cross Validation Scores: [0.43979778 0.37439589 0.40323318 0.40901774 0.39423079]
Mean: 0.404135071916747
Standard deviation: 0.021349455871711106
```

In [31]:
```python
# Random Forest
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
acc_test_random_forest = round(random_forest.score(X_test, Y_test) * 100, 2)
print("Accuracy on Training Set:", acc_random_forest)
print("Accuracy on Test Set:", acc_test_random_forest)
display_scores(Y_test, Y_pred)
```

```
Accuracy on Training Set: 84.09
Accuracy on Test Set: 84.07
MAE: 0.15925108564106216
Average Precision Score: 0.3739693620627699
Recall: 0.42028985507246375
F1: 0.5027783951989331
```

In [32]:
```python
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree'],
    'ScoreTrainingSet': [acc_svc, acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_linear_svc, acc_decision_tree],
    'ScoeTestSet': [acc_test_svc, acc_test_knn, acc_test_log,
              acc_test_random_forest, acc_test_gaussian, acc_test_perceptron,
              acc_test_sgd, acc_test_linear_svc, acc_test_decision_tree]})

models.sort_values(by='ScoreTrainingSet', ascending=False)
```

Out[32]:

	Model	ScoreTrainingSet	ScoeTestSet
3	Random Forest	84.09	84.07
8	Decision Tree	84.09	84.12
0	Support Vector Machines	82.42	82.64
4	Naive Bayes	82.38	82.53
2	Logistic Regression	82.23	82.35
7	Linear SVC	82.23	82.35
6	Stochastic Gradient Decent	82.22	82.35
5	Perceptron	80.72	81.00
1	KNN	80.03	80.34

In [33]:
```python
def rmse(predictions, targets):
    return
    np.sqrt(((predictions - targets) ** 2).mean())
```

In [ ]:

In [34]:
```python
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(tree_rmse_scores)
```

```
Scores: [0.43979778 0.37439589 0.40323318 0.40901774 0.39423079]
Mean: 0.404135071916747
Standard deviation: 0.021349455871711106
```

In [ ]: