```
  m  import pandas as pd
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier

fema = pd.read_csv('/content/final_project_FEMA (1).csv')
len(fema)
```

```
#fema = fema.sample(frac=0.02)
#len(fema)
```

```
fema = fema.dropna()
len(fema)
```

```
X = fema.drop(['tsaEligible',], axis = 1)
Y = fema['tsaEligible']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', MinMaxScaler(), ['waterLevel', 'grossIncome', 'householdComposition', 'repairAmount']),
        ('cat', OneHotEncoder(handle_unknown='ignore'), ['damagedCity', 'damagedStateAbbreviation', 'disasterNumber', 'residenceType'])
    ], remainder ='passthrough')

pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

X_train = pipeline.fit_transform(X_train)

X_test = pipeline.transform(X_test)
```

```
mlp = MLPClassifier(hidden_layer_sizes = (32, 16), max_iter = 100, activation = 'tanh', alpha = 0.0001, solver = 'sgd', verbose = False, tol = 0.00000001)
mlp.fit(X_train, Y_train)
y_pred = mlp.predict(X_test)
print("Confusion Matrix:")
mlp_cm = confusion_matrix(Y_test, y_pred)
print(mlp_cm)
mlp_accuracy = accuracy_score(Y_test, y_pred)
print("Accuracy:", mlp_accuracy)
mlp_precision = precision_score(Y_test, y_pred, average = 'weighted')
print("Precision:", mlp_precision)
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'hidden_layer_sizes': [(32, 16), (64, 32), (128, 64)],
    'max_iter': [50, 100, 200],
    'activation': ['tanh', 'relu'],
    'alpha': [0.0001, 0.001, 0.01],
    'solver': ['sgd', 'adam']
}

grid_search = GridSearchCV(estimator=mlp, param_grid=param_grid, cv=5, scoring='accuracy', verbose=1)


grid_search.fit(X_train, Y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best parameters:", best_params)
print("Best score:", best_score)


best_mlp = grid_search.best_estimator_
y_pred = best_mlp.predict(X_test)

print("Confusion Matrix:")
mlp_cm = confusion_matrix(Y_test, y_pred)
print(mlp_cm)
mlp_accuracy = accuracy_score(Y_test, y_pred)
print("Accuracy:", mlp_accuracy)
mlp_precision = precision_score(Y_test, y_pred, average='weighted')
print("Precision:", mlp_precision)


dt_model = DecisionTreeClassifier(max_depth=10, min_samples_leaf = 1, criterion ='gini', min_samples_split = 10, random_state=42)
dt_model.fit(X_train, Y_train)
Y_pred = dt_model.predict(X_test)

dt_cm = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:")
print(dt_cm)

dt_accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {dt_accuracy}")

dt_precision = precision_score(Y_test, Y_pred)
print(f"Precision: {dt_precision}")


param_grid_dt = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

dt_model = DecisionTreeClassifier(random_state=42)
grid_search_dt = GridSearchCV(estimator=dt_model, param_grid=param_grid_dt, cv=5, scoring='accuracy', verbose=1)
grid_search_dt.fit(X_train, Y_train)

best_params_dt = grid_search_dt.best_params_
best_score_dt = grid_search_dt.best_score_
```

```
print("Best parameters for Decision Tree:", best_params_dt)
print("Best score for Decision Tree:", best_score_dt)

best_dt_model = grid_search_dt.best_estimator_
y_pred_dt = best_dt_model.predict(X_test)

print("Confusion Matrix (Decision Tree):")
dt_cm = confusion_matrix(Y_test, y_pred_dt)
print(dt_cm)
dt_accuracy = accuracy_score(Y_test, y_pred_dt)
print("Accuracy (Decision Tree):", dt_accuracy)
dt_precision = precision_score(Y_test, y_pred_dt, average='weighted')
print("Precision (Decision Tree):", dt_precision)


rf = RandomForestClassifier(n_estimators=100, max_depth=20, min_samples_split=10, min_samples_leaf=2)
rf.fit(X_train, Y_train)
Y_pred = rf.predict(X_test)

rf_cm = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:")
print(rf_cm)

rf_accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {rf_accuracy}")

rf_precision = precision_score(Y_test, Y_pred)
print(f"Precision: {rf_precision}")
```

Suggested code may be subject to a license | arvimani/Exploratory-analysis-of-various-Machine-Learning-Models-using-ROC-AUC-Methodology | Jrmcclaskey/Music-Bot | Asterikss/ai-notebooks

```
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf_model = RandomForestClassifier(random_state=42)
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=param_grid_rf, cv=5, scoring='accuracy', verbose=1)
grid_search_rf.fit(X_train, Y_train)

best_params_rf = grid_search_rf.best_params_
best_score_rf = grid_search_rf.best_score_

print("Best parameters for Random Forest:", best_params_rf)
print("Best score for Random Forest:", best_score_rf)

best_rf_model = grid_search_rf.best_estimator_
y_pred_rf = best_rf_model.predict(X_test)

print("Confusion Matrix (Random Forest):")
rf_cm = confusion_matrix(Y_test, y_pred_rf)
print(rf_cm)
rf_accuracy = accuracy_score(Y_test, y_pred_rf)
print("Accuracy (Random Forest):", rf_accuracy)
rf_precision = precision_score(Y_test, y_pred_rf, average='weighted')
print("Precision (Random Forest):", rf_precision)
```