```python

from yellowbrick.target import ClassBalance
import pandas as pd
from sklearn.model_selection import train_test_split

def trainTestSplit(dataFrame,
                   test_size=0.2,
                   random_state=765,
                   stratifyColumn=None,
                   shuffle=True):

    origDataSize = len(dataFrame)
    indent = '---> '
    if stratifyColumn is None:
        train, test = train_test_split(dataFrame,
                                       test_size=test_size,
                                       random_state=random_state,
                                       shuffle=shuffle
                                       )
    else:
        train, test = train_test_split(dataFrame,
                                       test_size=test_size,
                                       random_state=random_state,
                                       stratify=dataFrame[[stratifyColumn]],
                                       shuffle=shuffle
                                       )

    print(f'Completed train/test split (test_size = {test_size}):')
    print(f'{indent}Original data size: {origDataSize}')
    print(f'{indent}Training data size: {len(train)}')
    print(f'{indent}Testing data size: {len(test)}')
    if stratifyColumn is None:
        print(f'{indent}Not stratified on any column')
    else:
        print(f'{indent}Stratified on column: {stratifyColumn}')

    return train, test


def classBalanceUndersample(dataFrame,
                            columnName,
                            alreadyBalanced=False):

    #Display the initial state
    tDf = dataFrame.copy()
    visualizer = ClassBalance()
    visualizer.fit(tDf[columnName])
    visualizer.show()

    if alreadyBalanced:
        print("Classes already balanced")
        return

    # Not balanced, need to get some info to get size to balance to
    ttlColName = 'ttlCol'

    # Find the sample size by finding which group/class is smallest
    tDfSize = tDf.groupby([columnName]).size().to_frame(ttlColName).sort_values(by=ttlColName)
    tDfSize.reset_index(inplace=True)
    sample_size = pd.to_numeric(tDfSize[ttlColName][0])
    sample_class = tDfSize[columnName][0]
    print(f'Undersampling data to match min class: {str(sample_class)} of size: {sample_size}')

    # Do the sampling
    tDf = tDf.groupby(columnName, group_keys=False).apply(lambda x: x.sample(sample_size))
    tDf.reset_index(drop=True, inplace=True)

    # Visualize
    visualizer2 = ClassBalance()
    visualizer2.fit(tDf[columnName])
    visualizer2.show()

    # Return the balance dataset
    return tDf


def displayClassBalance(dataFrame,
                        columnName,
                        verbose=False,
                        showRecords=5):
    ttlColName = 'ttlCol'

    visualizer = ClassBalance()
    visualizer.fit(dataFrame[columnName])  # Fit the data to the visualizer
    visualizer.show()  # Finalize and render the figure

    if verbose:
        tDfSize = dataFrame.groupby([columnName]).size().to_frame(ttlColName).sort_values(by=ttlColName)
        tDfSize.reset_index(inplace=True)
        display(tDfSize.head(showRecords))

```