You:

Create a python class called PrincipalComponentAnalysis that implements the Principal Component Analysis algorithm from scratch. During initalization the class should take one argument, n_components, if not specified it should default to all. Add a fit method that takes one input, X, that is either a numpy array, pandas dataframe or a list of lists as input. The fit method should compute the eigen values and eigen vectors (components) for the inputted data, and stores them on the object. If n_components is specified the fit method should store only top n_components eigen values and eigen vectors. In the fit, the method should also compute the explained variance ratio, and cumulative sum of explained variance ratio for each component and store it on the object. Also add a transform method that takes one input, X, that is either a numpy array, pandas dataframe or a list of lists as input. The transform method should project the inputted data onto the components and return the projected data. The transform method should also take an optional argument, n_components, that specifies the number of components to project onto. If n_components is not specified it should default to all. The last method to add is a fit_transform method that takes one input, X, that is either a numpy array, pandas dataframe or a list of lists as input. Like transform, this method should also take an optional n_components, specifying n_components, otherwise use all. The fit_transform method should call the fit method and then the transform method. Make sure to add documentation to the class and methods in the form of docstrings and comments. Also make sure to add type hints to the methods.

ChatGPT:

Here's a Python class called `PrincipalComponentAnalysis` that implements the Principal Component Analysis (PCA) algorithm as per your requirements:

```
python
import numpy as np
import pandas as pd

class PrincipalComponentAnalysis:
    """
    Principal Component Analysis (PCA) class for dimensionality reduction.

    Parameters:
    -----------
    n_components : int, optional
        Number of principal components to retain. Defaults to None, which keeps all components.
    """

    def __init__(self, n_components=None):
        self.n_components = n_components
        self.mean = None
        self.components = None
        self.explained_variance_ratio_ = None
        self.cumulative_variance_ratio_ = None

    def fit(self, X):
        """
        Fit the PCA model to the input data and compute the eigenvalues, eigenvectors,
        explained variance ratio, and cumulative explained variance ratio.

        Parameters:
        -----------
        X : array-like, shape (n_samples, n_features)
            Input data for PCA. Can be a NumPy array, Pandas DataFrame, or a list of lists.
        """

        # Ensure X is a NumPy array
        if isinstance(X, pd.DataFrame):
            X = X.to_numpy()
        elif not isinstance(X, np.ndarray):
            X = np.array(X)

        # Compute the mean of the data
        self.mean = np.mean(X, axis=0)

        # Center the data
        centered_data = X - self.mean

        # Compute the covariance matrix
        cov_matrix = np.cov(centered_data, rowvar=False)

        # Compute the eigenvalues and eigenvectors
        eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)

        # Sort eigenvalues and eigenvectors in descending order
        sorted_indices = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[sorted_indices]
        eigenvectors = eigenvectors[:, sorted_indices]
        # Store all or n_components eigenvectors as components
        if self.n_components is None:
            self.components = eigenvectors
        else:
            self.components = eigenvectors[:, :self.n_components]

        # Compute explained variance ratio and cumulative explained variance ratio
        total_variance = np.sum(eigenvalues)
        self.explained_variance_ratio_ = eigenvalues / total_variance
        self.cumulative_variance_ratio_ = np.cumsum(self.explained_variance_ratio_)

    def transform(self, X, n_components=None):
        """
        Project the input data onto the principal components.

        Parameters:
        ----------
        X : array-like, shape (n_samples, n_features)
            Input data for projection. Can be a NumPy array, Pandas DataFrame, or a list of lists.
        n_components : int, optional
            Number of principal components to project onto. Defaults to None, which uses all components.

        Returns:
        -------
        transformed_data : array-like, shape (n_samples, n_components)
            Projected data.
        """
        if n_components is None:
            n_components = self.n_components

        # Ensure X is a NumPy array
        if isinstance(X, pd.DataFrame):
            X = X.to_numpy()
        elif not isinstance(X, np.ndarray):
            X = np.array(X)

        # Center the data
        centered_data = X - self.mean

        # Project the data onto the specified number of components
        if n_components is None:
            return np.dot(centered_data, self.components)
        else:
            return np.dot(centered_data, self.components[:, :n_components])

    def fit_transform(self, X, n_components=None):
        """
        Fit the PCA model to the input data and then project the data onto the principal components.

        Parameters:
        ----------
        X : array-like, shape (n_samples, n_features)
            Input data for PCA and projection. Can be a NumPy array, Pandas DataFrame, or a list of lists.
        n_components : int, optional
            Number of principal components to retain. Defaults to None, which uses all components.

        Returns:
        -------
        transformed_data : array-like, shape (n_samples, n_components)
            Projected data.
        """
        self.fit(X)
        return self.transform(X, n_components)
```

This class allows you to perform PCA on your data by initializing an instance of `PrincipalComponentAnalysis`, fitting it to your data, and then transforming or fit-transforming it as needed.
