

RasKrebs: Create a python class called `PrincipalComponentAnalysis` that implements the Principal Component Analysis algorithm from scratch. During initialization the class should take two arguments:

- `n_components`, if not specified it should default to all.
- `decomposition_method`, which should be either 'eigen' or 'svd'. If not specified it should default to 'eigen'. 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.

GitHub Copilot: Here is the Python class `PrincipalComponentAnalysis` that implements the Principal Component Analysis algorithm from scratch:

```
import numpy as np
from typing import Optional, Union

class PrincipalComponentAnalysis:
    """
    Principal Component Analysis (PCA) class.
    """

    def __init__(self, n_components: Optional[int] = None, decomposition_method: str = 'eigen'):
        """
        Initialize the PCA class.

        Parameters:
        n_components (int): Number of components to keep. If not specified, all components are kept.
        decomposition_method (str): The method to use for decomposition. Either 'eigen' or 'svd'. Defaults to 'eigen'.
        """

        self.n_components = n_components
        self.decomposition_method = decomposition_method
        self.components = None
        self.explained_variance_ = None
        self.explained_variance_ratio_ = None
        self.cumulative_explained_variance_ratio_ = None

    def fit(self, X: Union[np.ndarray, list]):
        """
        Fit the model with X.

        Parameters:
        X (numpy.ndarray or list): Training data.
        """

        X = np.asarray(X)
        X = X - np.mean(X, axis=0)

        if self.decomposition_method == 'eigen':
            covariance_matrix = np.cov(X.T)
            eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
            idx = eigenvalues.argsort()[::-1]
            self.components = eigenvectors[:, idx]
            self.explained_variance_ = eigenvalues[idx]
        elif self.decomposition_method == 'svd':
            _, s, Vt = np.linalg.svd(X, full_matrices=False)
            self.components = Vt.T
            self.explained_variance_ = s**2 / (X.shape[0] - 1)

        self.explained_variance_ratio_ = self.explained_variance_ / self.explained_variance_.sum()
        self.cumulative_explained_variance_ratio_ = np.cumsum(self.explained_variance_ratio_)

        if self.n_components is not None:
            self.components = self.components[:, :self.n_components]
            self.explained_variance_ = self.explained_variance_[:self.n_components]
            self.explained_variance_ratio_ = self.explained_variance_ratio_[:self.n_components]
            self.cumulative_explained_variance_ratio_ = self.cumulative_explained_variance_ratio_[:self.n_components]

    def transform(self, X: Union[np.ndarray, list], n_components: Optional[int] = None):
        """
```

```

    Apply dimensionality reduction to X.

    Parameters:
    X (numpy.ndarray or list): The input data.
    n_components (int): Number of components to keep. If not specified, all components are kept.

    Returns:
    X_new (numpy.ndarray): The input data transformed to the PCA space.
    """
    X = np.asarray(X)
    X = X - np.mean(X, axis=0)
    X_new = np.dot(X, self.components[:, :n_components])
    return X_new

def fit_transform(self, X: Union[np.ndarray, list], n_components: Optional[int] = None):
    """
    Fit the model with X and apply the dimensionality reduction on X.

    Parameters:
    X (numpy.ndarray or list): The input data.
    n_components (int): Number of components to keep. If not specified, all components are kept.

    Returns:
    X_new (numpy.ndarray): The input data transformed to the PCA space.
    """
    self.fit(X)
    return self.transform(X, n_components)

```

This class has three methods: `fit`, `transform`, and `fit_transform`. The `fit` method computes the eigenvalues and eigenvectors of the covariance matrix of the input data, and stores them in the object. The `transform` method projects the input data onto the eigenvectors, and returns the projected data. The `fit_transform` method first calls the `fit` method and then the `transform` method.