You:

Create a python class called PrincipalComponentAnalysis that implements the Principal Component Analysis algorithm from scratch. It should have three methods, fit, transform and fit_transform. Two arguments should be able to be passed, first is n_components which specifies the number of components to use. If n_components is not specified it should default to all. Second is decomposition_method, which should be either 'eigen' or 'svd'. If not specified it should default to 'eigen'. 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 is a Python class called `PrincipalComponentAnalysis` that implements the Principal Component Analysis (PCA) algorithm from scratch, with the specified methods and optional arguments:

```
python
import numpy as np

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

    Parameters:
    ----------
    n_components : int, optional (default=None)
        Number of components to use. If None, all components will be retained.

    decomposition_method : str, optional (default='eigen')
        Method for PCA decomposition. Should be 'eigen' for eigenvalue decomposition
        or 'svd' for Singular Value Decomposition (SVD).

    Attributes:
    ----------
    n_components : int
        Number of components to use.

    decomposition_method : str
        Method used for PCA decomposition.

    eigenvectors_ : ndarray, shape (n_features, n_components)
        Principal components (eigenvectors).

    explained_variance_ : ndarray, shape (n_components,)
        Variance explained by each principal component.

    explained_variance_ratio_ : ndarray, shape (n_components,)
        Proportion of total variance explained by each principal component.

    mean_ : ndarray, shape (n_features,)
        Mean of the training data.

    Methods:
    -------
    fit(X: ndarray) -> None
        Fit the PCA model on the training data.

    transform(X: ndarray) -> ndarray
        Transform the input data into the reduced feature space.
```

```
    fit_transform(X: ndarray) -> ndarray
        Fit the PCA model on the training data and transform it.

    """
    def __init__(self, n_components=None, decomposition_method='eigen'):
        self.n_components = n_components
        self.decomposition_method = decomposition_method
        self.eigenvectors_ = None
        self.explained_variance_ = None
        self.explained_variance_ratio_ = None
        self.mean_ = None

    def fit(self, X: np.ndarray) -> None:
        """
        Fit the PCA model on the training data.

        Parameters:
        ----------
        X : ndarray, shape (n_samples, n_features)
            Training data.

        Returns:
        -------
        None
        """
        # Calculate the mean of the data
        self.mean_ = np.mean(X, axis=0)

        # Center the data by subtracting the mean
        X_centered = X - self.mean_

        if self.decomposition_method == 'eigen':
            # Eigenvalue decomposition
            cov_matrix = np.cov(X_centered.T)
            eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
        elif self.decomposition_method == 'svd':
            # Singular Value Decomposition (SVD)
            _, S, Vt = np.linalg.svd(X_centered, full_matrices=False)
            eigenvectors = Vt.T
            eigenvalues = S ** 2 / (X.shape[0] - 1)
        else:
            raise ValueError("Invalid decomposition_method. Use 'eigen' or 'svd'.")

        # Sort eigenvalues and eigenvectors in descending order
        sorted_indices = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[sorted_indices]
        eigenvectors = eigenvectors[:, sorted_indices]

        # Store the eigenvectors and explained variance
        self.eigenvectors_ = eigenvectors[:, :self.n_components]
        self.explained_variance_ = eigenvalues[:self.n_components]
        total_variance = np.sum(eigenvalues)
        self.explained_variance_ratio_ = self.explained_variance_ / total_variance

    def transform(self, X: np.ndarray) -> np.ndarray:
        """
        Transform the input data into the reduced feature space.

        Parameters:
        ----------
        X : ndarray, shape (n_samples, n_features)
            Input data to be transformed.
```

```
        Returns:
        --------
        ndarray, shape (n_samples, n_components)
            Transformed data.
        """
        if self.mean_ is None or self.eigenvectors_ is None:
            raise ValueError("The PCA model has not been fitted. Call fit() first.")

        # Center the data and project it onto the selected principal components
        X_centered = X - self.mean_
        transformed_data = np.dot(X_centered, self.eigenvectors_)

        return transformed_data

    def fit_transform(self, X: np.ndarray) -> np.ndarray:
        """
        Fit the PCA model on the training data and transform it.

        Parameters:
        -----------
        X : ndarray, shape (n_samples, n_features)
            Training data.

        Returns:
        --------
        ndarray, shape (n_samples, n_components)
            Transformed data.
        """
        self.fit(X)
        return self.transform(X)
```

You can use this `PrincipalComponentAnalysis` class to perform PCA with the specified options and methods.
