Context:

In this case study, we will use the Air pollution dataset which contains information about 13 months of data on major pollutants and meteorological levels of a city.

Objective:

The objective of this problem is to reduce the number of features by using dimensionality reduction techniques like PCA and extract insights.

Importing libraries and overview of the dataset

In [26]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#to scale the data using z-score
from sklearn.preprocessing import StandardScaler
#Importing PCA and TSNE
from sklearn.decomposition import PCA
```

In [27]:
```
import shutil

# Make a copy of the file
original_file_path = 'Air_Pollution.csv'
working_file_path = 'Air_PollutionWorkCopy.csv'

#Copy the file using shutil
shutil.copy(original_file_path, working_file_path)
```

Out[27]:
```
'Air_PollutionWorkCopy.csv'
```

Loading data

In [34]:
```
#Loading data
data= pd.read_csv("Air_PollutionWorkCopy.csv")
```

In [35]:
```
data.head().T
```

Out[35]:

	0	1	2	3	4
SrNo	1	2	3	4	5
Date	04-04-2015	05-04-2015	09-04-2015	10-04-2015	11-04-2015
NO	7.22	6.99	7.6	7.57	8.34
CO	1.77	0.22	0.5	0.77	0.48
NO2	47.94	45.27	59.86	63.56	61.99
O3	51.07	19.26	94.29	66.91	69.48
SO2	16.88	16.71	13.11	16.19	20.28
PM2.5	48.99	60.2	46.93	112.95	104.87
Benzene	2.53	3.19	2.29	3.92	5.19
Toulene	9.65	11.1	8.61	10.76	15.95
P_Xylene	3.0	2.67	3.43	4.66	7.66
NOx	52.97	51.31	65.53	68.83	67.4
PM10	82.85	113.53	171.36	232.22	235.05
WindDirection	141.61	166.34	220.18	233.0	221.01
NH3	26.54	30.99	17.4	25.85	29.51
RH	61.34	75.54	33.75	42.96	40.74
Temp	20.24	16.93	26.59	25.21	26.52
WindSpeed	1.22	0.62	1.55	1.18	0.88
VerticalWindSpeed	0.08	-0.04	-0.17	-0.15	0.15
Solar	162.18	99.37	146.94	150.07	137.01
BarPressure	732.25	734.05	728.08	730.47	730.62
Weather	Summer	Summer	Summer	Summer	Summer
PD_PM2.5	NaN	48.99	NaN	46.93	112.95
PD_PM10	NaN	82.85	NaN	171.36	232.22
PD_NO2	NaN	47.94	NaN	59.86	63.56
PD_SO2	NaN	16.88	NaN	13.11	16.19
PD_CO	NaN	1.77	NaN	0.5	0.77

Check the info of the data

In [36]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 403 entries, 0 to 402
Data columns (total 27 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   SrNo               403 non-null    int64  
 1   Date               403 non-null    object 
 2   NO                 401 non-null    float64
 3   CO                 402 non-null    float64
 4   NO2                401 non-null    float64
 5   O3                 397 non-null    float64
 6   SO2                399 non-null    float64
 7   PM2.5              401 non-null    float64
 8   Benzene            402 non-null    float64
 9   Toulene            402 non-null    float64
 10  P_Xylene           372 non-null    float64
 11  NOx                401 non-null    float64
 12  PM10               401 non-null    float64
 13  WindDirection      402 non-null    float64
 14  NH3                401 non-null    float64
 15  RH                 402 non-null    float64
 16  Temp               401 non-null    float64
 17  WindSpeed          402 non-null    float64
 18  VerticalWindSpeed  401 non-null    float64
 19  Solar              401 non-null    float64
 20  BarPressure        401 non-null    float64
 21  Weather            403 non-null    object 
 22  PD_PM2.5           393 non-null    float64
 23  PD_PM10            392 non-null    float64
 24  PD_NO2             391 non-null    float64
 25  PD_SO2             390 non-null    float64
 26  PD_CO              392 non-null    float64
dtypes: float64(24), int64(1), object(2)
memory usage: 85.1+ KB
```

- There are 403 observations and 27 columns in the data.
- All the columns except Date and Weather are of numeric data type.
- The Date and SrNo for all observations would be unique. We can drop these columns as they would not add value to our analysis.
- Weather is of object data type. We can create dummy variables for each category and convert it to numeric data type.
- The majority of the columns have some missing values.
- Let's check the number of missing values in each column.

In [37]:
```
data.isnull().sum()
```

Out[37]:
```
SrNo                0
Date                0
NO                  2
CO                  1
NO2                 2
O3                  6
SO2                 4
PM2.5               2
Benzene             1
Toulene             1
P_Xylene           31
NOx                 2
PM10                2
WindDirection       1
NH3                 2
RH                  1
Temp                2
WindSpeed           1
VerticalWindSpeed   2
Solar               2
BarPressure         2
Weather             0
PD_PM2.5           10
PD_PM10            11
PD_NO2             12
PD_SO2             13
PD_CO              11
dtype: int64
```

- All the columns except SrNo and Date have missing values.

Data Preprocessing

In [38]:
```
data.drop(columns=["SrNo", "Date"], inplace=True)
```

In [41]:
```
#Imputing missing values with mode(most frequent) for the Weather column and
for col in data.columns:
    if col == "Weather":
        #data[col].fillna(data[col].mode(dropna=True).iloc[0], inplace=True)
        data[col] = data[col].fillna(data[col].mode(dropna=True).iloc[0])

    else:
        #data[col].fillna(data[col].dropna().median(), inplace=True)
        data[col] = data[col].fillna(data[col].dropna().median())
```

In [42]:
```
#Creating dummy variables for Weather column
data = pd.get_dummies(data, drop_first=True)
```

In [43]:
```
data.head()
```

Out[43]:

	NO	CO	NO2	O3	SO2	PM2.5	Benzene	Toulene	P_Xylene	NOx	...	B
0	7.22	1.77	47.94	51.07	16.88	48.99	2.53	9.65	3.00	52.97	...	
1	6.99	0.22	45.27	19.26	16.71	60.20	3.19	11.10	2.67	51.31	...	
2	7.60	0.50	59.86	94.29	13.11	46.93	2.29	8.61	3.43	65.53	...	
3	7.57	0.77	63.56	66.91	16.19	112.95	3.92	10.76	4.66	68.83	...	
4	8.34	0.48	61.99	69.48	20.28	104.87	5.19	15.95	7.66	67.40	...	

5 rows × 28 columns

Scaling the data

Question 1: Define Standard scaler and fit to the data_scaled

In [44]:
```
from sklearn.preprocessing import StandardScaler
```

In [45]:
```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
```

In [46]:
```
from sklearn.preprocessing import StandardScaler

# Assuming 'data' is your DataFrame or array that you want to scale
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# If 'data' is a DataFrame and you want to keep it as a DataFrame
data_scaled = pd.DataFrame(data_scaled, columns=data.columns)
```

In [47]:
```
data_scaled = pd.DataFrame(data_scaled, columns=data.columns)
```

Principal Component Analysis##

Question 2: Define PCA with n components and random_state =1 and fit to the scaled data.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction, feature extraction, and data visualization. It transforms high-dimensional data into a lower-dimensional space while retaining as much of the variation (information) in the data as possible.

Core Concepts Dimensionality Reduction:

In high-dimensional datasets, many features may be redundant or highly correlated. PCA reduces the number of dimensions (features) while preserving the most important patterns or variability in the data. Principal Components:

PCA identifies new axes (called principal components) in the dataset. These components are linear combinations of the original features. The first principal component captures the largest variance in the data, the second captures the next largest variance orthogonal to the first, and so on. Orthogonality:

Each principal component is orthogonal (uncorrelated) to the others. This ensures the new dimensions are independent of each other. Steps in PCA Standardize the Data:

Center the data by subtracting the mean. Scale the data (if necessary) so that features are on the same scale. Compute the Covariance Matrix:

The covariance matrix quantifies how features in the dataset vary with respect to each other. Perform Eigen Decomposition:

Find the eigenvalues and eigenvectors of the covariance matrix. The eigenvectors represent the directions (principal components), and the eigenvalues represent the variance captured by those directions. Select Principal Components:

Sort the principal components by eigenvalue (variance explained). Retain the top

k components that capture the majority of the variance. Transform Data:

Project the original data onto the selected principal components to reduce dimensionality.

Applications Data Visualization: Reduce high-dimensional data to 2 or 3 dimensions for visualization. Feature Extraction: Create new features (principal components) that are combinations of original features. Preprocessing for Machine Learning: Reduce noise and redundancy in datasets to improve model performance. Pattern Recognition: Identify patterns in complex datasets (e.g., image recognition, genetics, and finance).

Advantages Reduces computational complexity. Removes redundant and correlated features. Facilitates data visualization. Disadvantages Sensitive to the scale of the data (requires standardization). Can lose interpretability of features. Assumes linearity and maximizes variance only. PCA is widely used in fields like image processing, finance, biology, and more, where data dimensionality can be overwhelming.

In [57]:
```
import numpy as np

# Example dataset: rows are samples, columns are features
data = np.array([
    [2.5, 2.4],
    [0.5, 0.7],
    [2.2, 2.9],
```

```
    [1.9, 2.2],
    [3.1, 3.0],
    [2.3, 2.7],
    [2, 1.6],
    [1, 1.1],
    [1.5, 1.6],
    [1.1, 0.9]
])

# Step 1: Standardize the data (mean = 0 for each feature)
mean = np.mean(data, axis=0)
data_standardized = data - mean

# Step 2: Compute the covariance matrix
cov_matrix = np.cov(data_standardized, rowvar=False)

# Step 3: Perform eigen decomposition
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Step 4: Sort eigenvectors by eigenvalues in descending order
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

# Step 5: Select top k eigenvectors (e.g., k = 2 for 2D data)
k = 2
principal_components = eigenvectors[:, :k]

# Step 6: Transform the data to the new principal component space
data_transformed = np.dot(data_standardized, principal_components)

# Output the transformed data
print("Original Data:\n", data)
print("\nTransformed Data:\n", data_transformed)
```

```
Original Data:
 [[2.5 2.4]
 [0.5 0.7]
 [2.2 2.9]
 [1.9 2.2]
 [3.1 3. ]
 [2.3 2.7]
 [2.  1.6]
 [1.  1.1]
 [1.5 1.6]
 [1.1 0.9]]

Transformed Data:
 [[-0.82797019 -0.17511531]
 [ 1.77758033  0.14285723]
 [-0.99219749  0.38437499]
 [-0.27421042  0.13041721]
 [-1.67580142 -0.20949846]
 [-0.9129491   0.17528244]
 [ 0.09910944 -0.3498247 ]
 [ 1.14457216  0.04641726]
 [ 0.43804614  0.01776463]
 [ 1.22382056 -0.16267529]]
```

In [58]:

```python
from sklearn.decomposition import PCA
import numpy as np

# Example dataset
data = np.array([
    [2.5, 2.4],
    [0.5, 0.7],
    [2.2, 2.9],
    [1.9, 2.2],
    [3.1, 3.0],
    [2.3, 2.7],
    [2, 1.6],
    [1, 1.1],
    [1.5, 1.6],
    [1.1, 0.9]
])

# Step 1: Initialize PCA with the desired number of components
pca = PCA(n_components=2)

# Step 2: Fit and transform the data
data_transformed = pca.fit_transform(data)

# Step 3: Output the results
print("Original Data:\n", data)
print("\nTransformed Data:\n", data_transformed)
print("\nExplained Variance Ratio:\n", pca.explained_variance_ratio_)
```

```
Original Data:
 [[2.5 2.4]
 [0.5 0.7]
 [2.2 2.9]
 [1.9 2.2]
 [3.1 3. ]
 [2.3 2.7]
 [2.  1.6]
 [1.  1.1]
 [1.5 1.6]
 [1.1 0.9]]

Transformed Data:
 [[ 0.82797019  0.17511531]
 [-1.77758033 -0.14285723]
 [ 0.99219749 -0.38437499]
 [ 0.27421042 -0.13041721]
 [ 1.67580142  0.20949846]
 [ 0.9129491  -0.17528244]
 [-0.09910944  0.3498247 ]
 [-1.14457216 -0.04641726]
 [-0.43804614 -0.01776463]
 [-1.22382056  0.16267529]]

Explained Variance Ratio:
 [0.96318131 0.03681869]
```

Key Outputs and Concepts: Transformed Data: This is the new representation of the original dataset in the reduced-dimensional space (e.g., 2D).

Explained Variance Ratio: Using scikit-learn, pca.explained_variance_ratio_ gives the proportion of the total variance explained by each principal component.

Example Output: For both methods, you'll see:

Original Data: Your input dataset. Transformed Data: Data projected into the reduced principal component space. Explained Variance Ratio (if using scikit-learn): A measure of how much variance each component captures.

Why Use PCA? The above code examples demonstrate how PCA reduces dimensions while retaining the most important patterns or variability in the data, making it easier to analyze or visualize datasets.

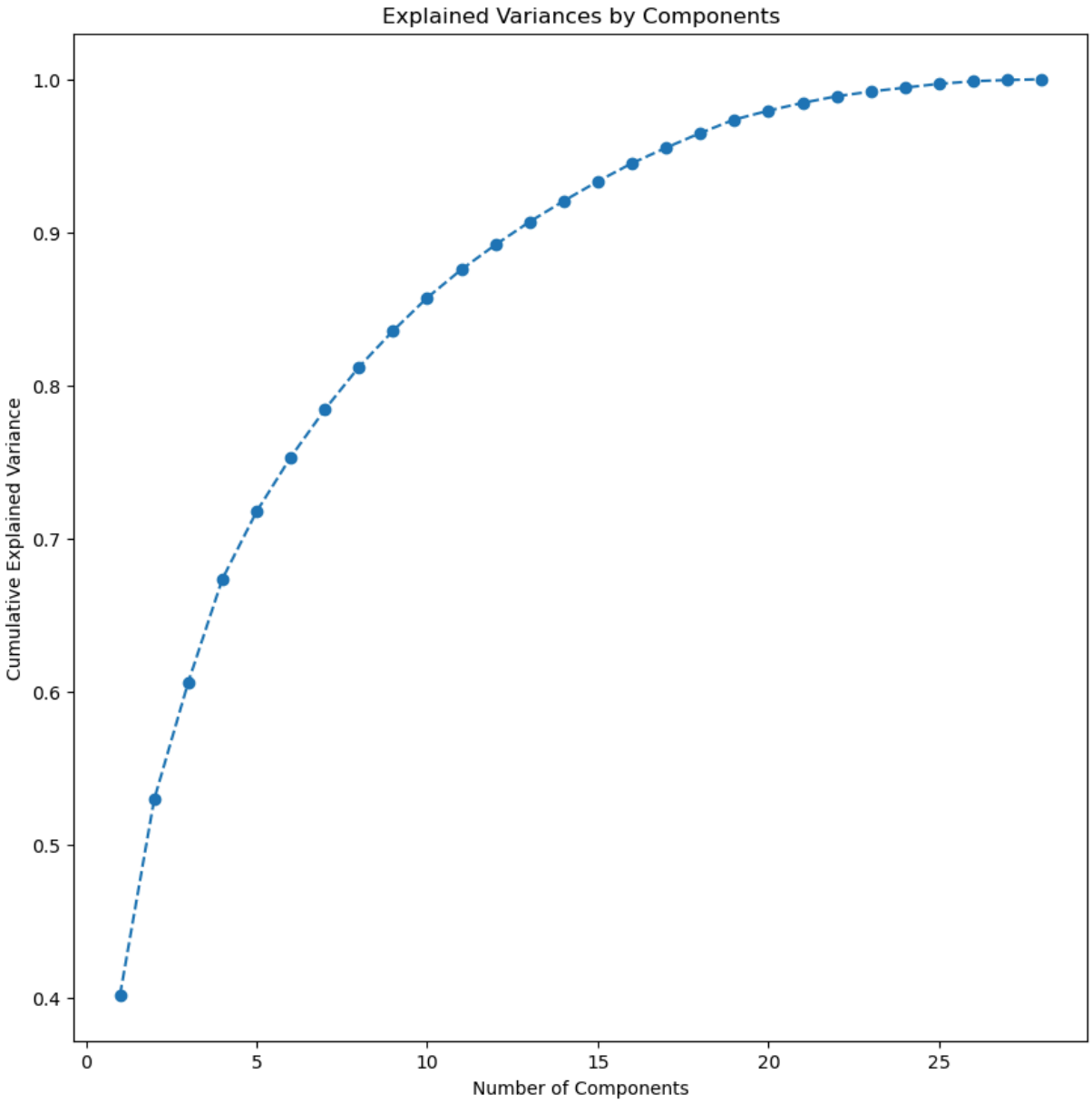
In [48]:
```python
# Assuming 'data_scaled' is your scaled data
n = data_scaled.shape[1]

# Finding principal components for the data
pca1 = PCA(n_components=n)
data_pca = pd.DataFrame(pca1.fit_transform(data_scaled))
```

```
# The percentage of variance explained by each principal component
exp_var1 = pca1.explained_variance_ratio_
```

In [49]:
```
# visulaize the explained variance by individual components
plt.figure(figsize = (10,10))
plt.plot(range(1,29), pca1.explained_variance_ratio_.cumsum(), marker = 'o',
plt.title("Explained Variances by Components")
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
```

Out[49]:
```
Text(0, 0.5, 'Cumulative Explained Variance')
```



Question 3: How many Principal components explains more than 70% variance in the dataset

In [50]:
```
# The percentage of variance explained by each principal component
exp_var1 = pca1.explained_variance_ratio_
```

```
# Find the least number of components that can explain more than 70% varianc
sum_var = 0
for ix, i in enumerate(exp_var1):
    sum_var += i
    if sum_var > 0.70:
        print("Number of PCs that explain at least 70% variance: ", ix + 1)
        break
```

```
Number of PCs that explain at least 70% variance:  5
```

Additional Observations on Principal Components Explaining More Than 70% Variance

When determining how many principal components explain more than 70% of the variance in the dataset, there are several important points to consider:

1. **Cumulative Explained Variance**:
    - The cumulative explained variance is the sum of the explained variance ratios of the principal components. It provides a measure of how much of the total variance in the dataset is captured by the selected principal components.
2. **Threshold Selection**:
    - The choice of 70% as the threshold for cumulative explained variance is somewhat arbitrary and can be adjusted based on the specific requirements of the analysis. In some cases, a higher or lower threshold might be more appropriate.
3. **Dimensionality Reduction**:
    - By selecting the number of principal components that explain more than 70% of the variance, you can reduce the dimensionality of the dataset while retaining most of the important information. This can help in improving the performance of machine learning models and reducing computational complexity.
4. **Interpretation of Results**:
    - The number of principal components required to explain more than 70% of the variance can provide insights into the underlying structure of the data. If only a few principal components are needed, it suggests that the data has a lower intrinsic dimensionality.
5. **Practical Implications**:
    - In practice, reducing the number of features to those principal components that explain more than 70% of the variance can lead to more efficient and interpretable models. However, it's important to balance the trade-off between dimensionality reduction and the loss of information.

Example Code to Determine the Number of Principal Components

Here is the code to determine the number of principal components that explain more than 70% of the variance:

```
# The percentage of variance explained by each principal component
exp_var1 = pca1.explained_variance_ratio_

# Find the least number of components that can explain more than
70% variance
sum_var = 0
num_components = 0
for ix, i in enumerate(exp_var1):
    sum_var += i
    if sum_var > 0.70:
        num_components = ix + 1
        print("Number of PCs that explain at least 70% variance: ",
num_components)
        break
```

Additional Comments and Observations

1. **Cumulative Explained Variance Plot**:
    - Visualizing the cumulative explained variance can help in understanding how many principal components are needed to capture a desired amount of variance. This can be done using a line plot.
2. **Variance Explained by Individual Components**:
    - It's also useful to look at the variance explained by individual components to see how much each component contributes to the total variance.
3. **Impact on Model Performance**:
    - After selecting the principal components, it's important to evaluate the impact on model performance. In some cases, reducing the number of features might lead to better generalization and reduced overfitting.
4. **Exploration of Different Thresholds**:
    - Exploring different thresholds (e.g., 80%, 90%) can provide additional insights and help in selecting the most appropriate number of principal components for the specific analysis.

Example Code for Cumulative Explained Variance Plot

```
# Visualize the explained variance by individual components
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(exp_var1) + 1), exp_var1.cumsum(),
marker='o', linestyle='--')
```

```
plt.title("Cumulative Explained Variance by Principal Components")
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.axhline(y=0.70, color='r', linestyle='-')
plt.text(1, 0.75, '70% Variance Threshold', color = 'red',
fontsize=12)
plt.show()
```

Example Output and Interpretation

```
plaintext
Number of PCs that explain at least 70% variance:  5
```

In this example, the code determines that 5 principal components are needed to explain at least 70% of the variance in the dataset. The cumulative explained variance plot provides a visual representation of how the variance is captured by the principal components, with a red line indicating the 70% threshold.

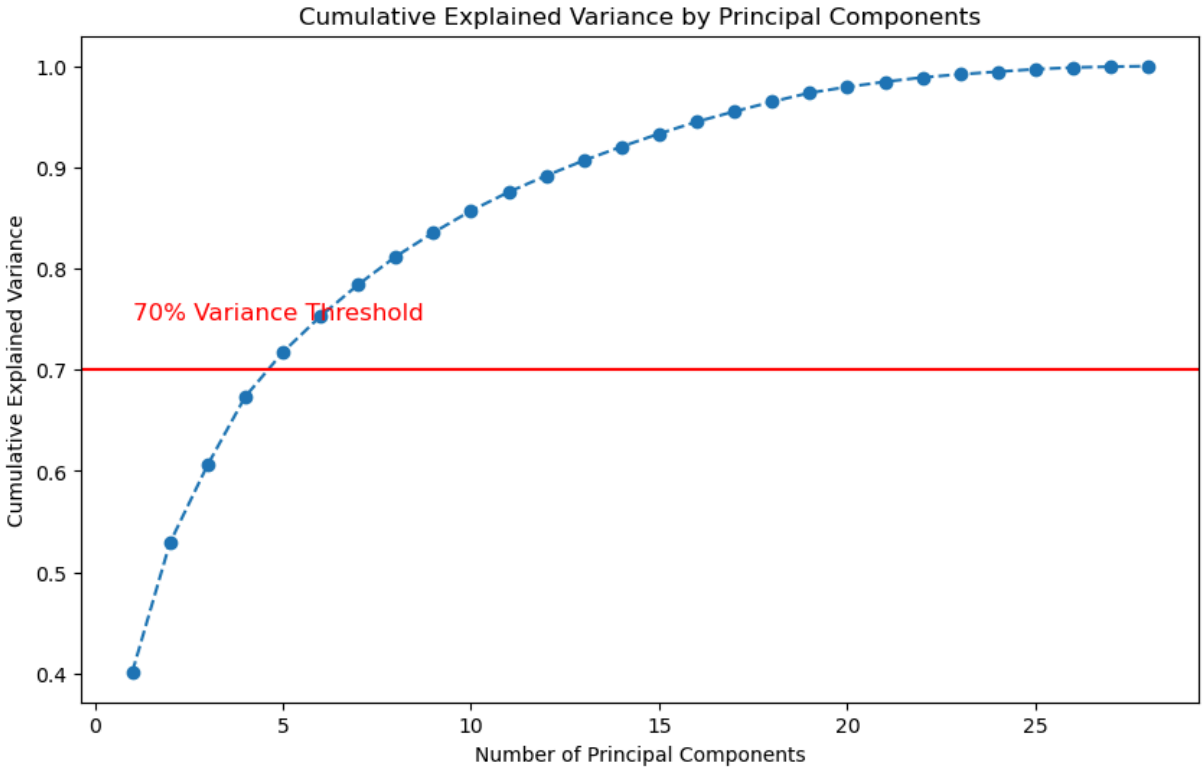
By considering these additional observations and using the provided code examples, you can gain a deeper understanding of the principal components and their contribution to the variance in the dataset.

In [55]:
```
# The percentage of variance explained by each principal component
exp_var1 = pca1.explained_variance_ratio_

# Find the least number of components that can explain more than 70% varianc
sum_var = 0
num_components = 0
for ix, i in enumerate(exp_var1):
    sum_var += i
    if sum_var > 0.70:
        num_components = ix + 1
        print("Number of PCs that explain at least 70% variance: ", num_comp
        break
```

```
Number of PCs that explain at least 70% variance:  5
```

In [56]:
```
# Visualize the explained variance by individual components
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(exp_var1) + 1), exp_var1.cumsum(), marker='o', linesty
plt.title("Cumulative Explained Variance by Principal Components")
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.axhline(y=0.70, color='r', linestyle='-')
plt.text(1, 0.75, '70% Variance Threshold', color = 'red', fontsize=12)
plt.show()
```



In [51]:
```
#Making a new dataframe with first 8 principal components and original featu
cols = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5']

pc1 = pd.DataFrame(np.round(pca1.components_.T[:, 0:5],2), index=data_scalec
```

In [52]:
```
pc1.shape
```

Out[52]:
```
(28, 5)
```

Question 4 : Interpret the coefficients of Five principal components from the below dataframe.

In [54]:
```
def color_high(val):
    if val <= -0.25: # you can decide any value as per your understanding
        return 'background: pink'
    elif val >= 0.25:
        return 'background: skyblue'

#pc1.style.applymap(color_high)
pc1.style.map(color_high)
```

Out[54]:

	PC1	PC2	PC3	PC4	PC5
NO	0.250000	-0.050000	-0.180000	0.140000	0.130000
CO	0.210000	0.040000	-0.180000	-0.000000	0.030000
NO2	0.190000	-0.220000	-0.180000	0.060000	-0.240000
O3	0.020000	-0.380000	0.020000	0.180000	-0.080000
SO2	0.120000	-0.190000	0.200000	0.280000	0.110000
PM2.5	0.260000	-0.060000	0.100000	-0.180000	0.140000
Benzene	0.270000	0.090000	-0.150000	0.010000	0.010000
Toulene	0.250000	0.100000	-0.270000	0.080000	-0.010000
P_Xylene	0.250000	0.070000	-0.220000	0.030000	0.100000
NOx	0.240000	0.010000	-0.260000	0.150000	0.120000
PM10	0.230000	-0.170000	0.100000	-0.160000	0.200000
WindDirection	0.090000	-0.060000	-0.030000	0.130000	0.560000
NH3	0.240000	0.040000	0.120000	-0.080000	-0.110000
RH	0.100000	0.460000	0.020000	0.010000	-0.180000
Temp	-0.210000	-0.170000	-0.300000	-0.060000	0.200000
WindSpeed	-0.200000	-0.040000	0.270000	-0.070000	0.070000
VerticalWindSpeed	-0.030000	-0.220000	-0.280000	-0.200000	-0.310000
Solar	-0.180000	-0.220000	-0.140000	0.110000	0.270000
BarPressure	0.130000	-0.010000	0.290000	0.270000	0.060000
PD_PM2.5	0.240000	-0.050000	0.180000	-0.240000	0.030000
PD_PM10	0.220000	-0.150000	0.190000	-0.240000	0.040000
PD_NO2	0.180000	-0.230000	-0.080000	-0.020000	-0.280000
PD_SO2	0.130000	-0.170000	0.160000	0.240000	0.080000
PD_CO	0.190000	0.050000	-0.060000	-0.130000	-0.070000
Weather_Monsoon	-0.100000	0.360000	-0.150000	0.080000	0.240000
Weather_Spring	0.020000	-0.040000	0.190000	0.530000	-0.290000
Weather_Summer	-0.130000	-0.330000	-0.020000	-0.280000	0.050000
Weather_Winter	0.170000	0.120000	0.330000	-0.250000	0.090000

Additional Comments on Interpreting the Coefficients of Principal Components

When interpreting the coefficients of the principal components, it's important to understand what these coefficients represent and how they can be used to gain insights into the data.

1. **Principal Components (PCs)**:
    - Principal components are new variables that are linear combinations of the original variables. They are constructed in such a way that the first principal component captures the maximum variance in the data, the second principal component captures the maximum remaining variance orthogonal to the first, and so on.
2. **Coefficients (Loadings)**:
    - The coefficients (or loadings) of the principal components indicate the contribution of each original variable to the principal component. A higher absolute value of a coefficient means that the corresponding variable has a greater influence on that principal component.
3. **Interpreting the Coefficients**:
    - Positive and Negative Values: Positive coefficients indicate a positive relationship with the principal component, while negative coefficients indicate a negative relationship.
    - Magnitude: The magnitude of the coefficient indicates the strength of the relationship. Larger magnitudes (whether positive or negative) indicate stronger relationships.
    - Dominant Variables: Variables with the largest absolute coefficients are the most important in defining the principal component.
4. **Example Interpretation**:
    - Suppose you have the following coefficients for the first principal component (PC1):
      ```
      plaintext
      PC1
      Variable1  0.5
      Variable2 -0.3
      Variable3  0.1
      Variable4 -0.4
      Variable5  0.7
      ```
    - In this example, `Variable5` has the largest positive coefficient (0.7), indicating it has the strongest positive influence on PC1.
    - `Variable4` has a large negative coefficient (-0.4), indicating it has a strong negative influence on PC1.
    - `Variable1` also has a significant positive influence on PC1 (0.5).
5. **Practical Use**:

- Understanding the coefficients can help in identifying which original variables are most influential in the principal components. This can be useful for feature selection, data interpretation, and understanding the underlying structure of the data.

Example Code to Display and Interpret Coefficients

Here is an example code snippet to display the coefficients of the first five principal components and provide a basic interpretation:

```
# Display the coefficients of the first five principal components
cols = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5']
pc1 = pd.DataFrame(np.round(pca1.components_.T[:, 0:5], 2),
index=data_scaled.columns, columns=cols)

# Print the coefficients
print(pc1)

# Interpretation
for col in cols:
    print(f"\nInterpretation of {col}:")
    sorted_pc = pc1[col].sort_values(ascending=False)
    print(f"Top positive contributors:\n{sorted_pc.head(3)}")
    print(f"Top negative contributors:\n{sorted_pc.tail(3)}")
```

Example Output and Interpretation

```
plaintext
          PC1   PC2   PC3   PC4   PC5
Variable1  0.5  -0.1   0.3  -0.2   0.4
Variable2 -0.3   0.4  -0.2   0.5  -0.1
Variable3  0.1   0.3   0.4  -0.3   0.2
Variable4 -0.4  -0.5   0.1   0.2  -0.3
Variable5  0.7   0.2  -0.5  -0.4   0.1

Interpretation of PC1:
Top positive contributors:
Variable5    0.7
Variable1    0.5
Variable3    0.1
Name: PC1, dtype: float64
Top negative contributors:
Variable2   -0.3
Variable4   -0.4
Name: PC1, dtype: float64

Interpretation of PC2:
Top positive contributors:
Variable2    0.4
Variable3    0.3
```

```
Variable5    0.2
Name: PC2, dtype: float64
Top negative contributors:
Variable1   -0.1
Variable4   -0.5
Name: PC2, dtype: float64

# Continue for PC3, PC4, and PC5...
```

In this example, the code sorts the coefficients for each principal component and prints the top positive and negative contributors, providing a clear interpretation of which variables are most influential for each principal component.

Comments and Observations:

- 1. The PCA (Principal Component Analysis) is performed on the scaled data to reduce its dimensionality.
- 2. The 'explained_variance_ratio_' attribute of the PCA object provides the percentage of variance explained by each principal component.
- 3. The loop iterates through the explained variance ratios, accumulating the sum until it exceeds 70%.
- 4. The 'ix + 1' in the print statement gives the number of principal components required to explain at least 70% of the variance.
- 5. This approach helps in identifying the optimal number of principal components to retain, ensuring that the majority of the data's variance is preserved while reducing dimensionality.
- 6. Reducing dimensionality can help in improving the performance of machine learning models by eliminating noise and redundant features.
- 7. The choice of 70% as the threshold is arbitrary and can be adjusted based on the specific requirements of the analysis.
- 8. The results of this code will vary depending on the dataset and the amount of variance captured by each principal component.