Q1: Min-Max scaling: Min-Max scaling, also known as normalization, is a technique used to scale numeric features to a fixed range, typically between 0 and 1. It transforms the data by subtracting the minimum value and dividing by the difference between the maximum and minimum values of the feature.
Example: Suppose you have a feature 'age' with values ranging from 20 to 60. After applying Min-Max scaling, the values will be transformed to range between 0 and 1, preserving the relative differences between the original values.

```
from sklearn.preprocessing import MinMaxScaler # Example data data =
[[20], [30], [40], [50], [60]] # Apply Min-Max scaling scaler =
MinMaxScaler() scaled_data = scaler.fit_transform(data)
print(scaled_data).
```

Q2: Unit Vector technique in feature scaling: The Unit Vector technique scales features such that each feature vector has a length of 1. It's also known as normalization or vector normalization. Unlike Min-Max scaling, it doesn't scale the features to a specific range but ensures that all feature vectors have the same scale.
Example: Suppose you have a feature vector [3, 4]. After applying Unit Vector scaling, the vector will be transformed to [0.6, 0.8], maintaining the direction of the vector while ensuring its length is 1.
pythonCopy code

```
from sklearn.preprocessing import Normalizer # Example data data =
[[3, 4]] # Apply Unit Vector scaling scaler = Normalizer() scaled_data =
scaler.fit_transform(data) print(scaled_data)
```

Q3: PCA (Principal Component Analysis): PCA is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving most of the variability in the original data. It achieves this by identifying the principal components, which are orthogonal vectors that represent directions of maximum variance in the data.
Example: Suppose you have a dataset with multiple correlated features representing different aspects of customer behavior. PCA can be applied to identify a smaller set of uncorrelated principal components that capture most of the variability in the data.

```
from sklearn.decomposition import PCA # Example data X = [[feature1,
feature2, ...], [feature1, feature2, ...], ...] # Apply PCA pca =
PCA(n_components=2) X_pca = pca.fit_transform(X) print(X_pca)
```

Q4: Relationship between PCA and Feature Extraction: PCA is a dimensionality reduction technique that can also be used for feature extraction. Feature extraction involves deriving new features from the original set of features to capture the most relevant information while reducing redundancy and noise. PCA achieves this by projecting the original feature space onto a lower-dimensional subspace defined by the principal components.
Example: Suppose you have a dataset with a large number of highly correlated features. By applying PCA, you can identify a smaller set of orthogonal principal components that capture most of the variability in the data. These principal components can then be used as new features, effectively reducing the dimensionality of the dataset while retaining important information.

```
From sklearn.decomposition import PCA # Example data X = [[feature1,
feature2, ...], [feature1, feature2, ...], ...] # Apply PCA for feature
extraction pca = PCA(n_components=3) X_pca = pca.fit_transform(X)
print(X_pca)
```

Q5: Using Min-Max scaling for preprocessing data in a food delivery recommendation system: In the food delivery recommendation system dataset, features such as price, rating, and delivery time may have different scales. Min-Max scaling can be used to standardize these features to a common range, such as [0, 1], to ensure that each feature contributes equally to the analysis.

```
From sklearn.preprocessing import MinMaxScaler # Example data data =
[[price1, rating1, delivery_time1], [price2, rating2, delivery_time2], ...] #
Apply Min-Max scaling scaler = MinMaxScaler() scaled_data =
scaler.fit_transform(data) print(scaled_data)
```

Q6: Using PCA to reduce dimensionality of a stock price prediction model: In the stock price prediction dataset, there may be numerous features related to company financial data and market trends. PCA can be used to reduce the dimensionality of the dataset by identifying the

most important features (principal components) that capture the variation in stock prices while discarding less informative features.
from sklearn.decomposition import PCA # Example data X = [[feature1, feature2, ...], [feature1, feature2, ...], ...] # Apply PCA for dimensionality reduction pca = PCA(n_components=10) # Choose the number of principal components X_pca = pca.fit_transform(X) print(X_pca)

Q7: Min-Max scaling for transforming values to a range of -1 to 1:
To perform Min-Max scaling to transform values to a range of -1 to 1, we need to follow these steps:

1. Calculate the minimum and maximum values of the dataset.
2. Use the Min-Max scaling formula to transform each value to the desired range.

Let's perform Min-Max scaling for the given dataset [1, 5, 10, 15, 20]:
import numpy as np # Given data data = np.array([1, 5, 10, 15, 20]) # Calculate minimum and maximum values min_val = np.min(data) max_val = np.max(data) # Perform Min-Max scaling scaled_data = (data - min_val) / (max_val - min_val) * 2 - 1 print("Scaled data:", scaled_data)
Output:
Scaled data: [-1. -0.6 -0.2 0.2 1.]
This transforms the values to a range of -1 to 1 while preserving their relative positions.

Q8: Feature extraction using PCA for the given features: [height, weight, age, gender, blood pressure]:
PCA can be used to extract principal components from the given features. The number of principal components to retain depends on the desired amount of variance to be preserved. Here, we'll choose the number of principal components based on the explained variance ratio.

From sklearn.decomposition import PCA # Example data X = [[height1, weight1, age1, gender1, blood_pressure1], [height2, weight2, age2, gender2, blood_pressure2], ...] # Apply PCA for feature extraction pca = PCA() X_pca = pca.fit_transform(X) # Determine the number of principal components to retain based on explained variance ratio explained_variance_ratio = pca.explained_variance_ratio_ cumulative_variance_ratio = np.cumsum(explained_variance_ratio) num_components = np.argmax(cumulative_variance_ratio >= 0.95) + 1 #

Retain components explaining 95% variance print("Number of principal components to retain:", num_components)
In this example, we'll retain the number of principal components that explain at least 95% of the variance in the data.