PCA(Principal Component Analysis)

(Code: Subhajit Das)

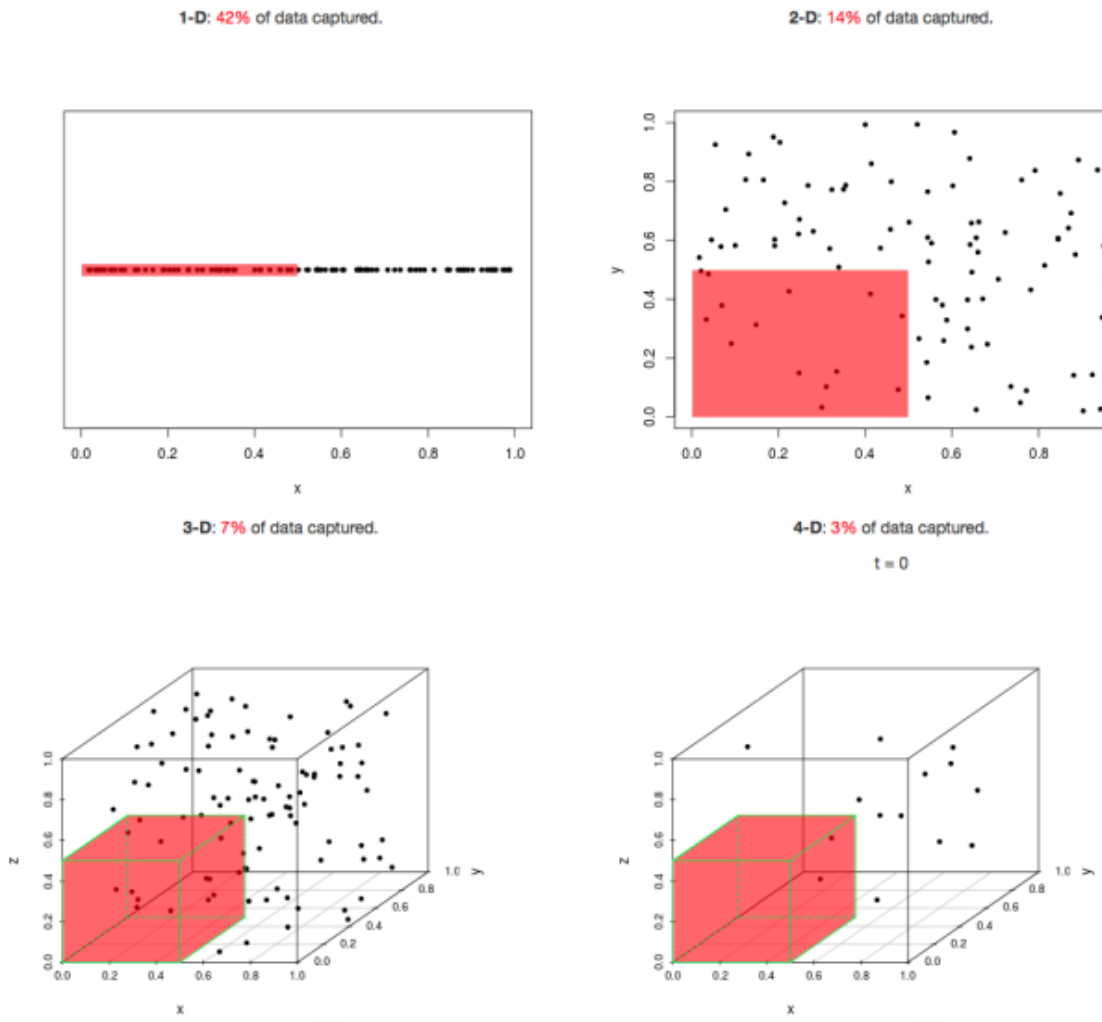
Curse of Dimensionality

The "Curse of Dimensionality" refers to a range of problems that arise when working with high-dimensional data, which are data sets with a large number of features or attributes. As the dimensionality of the data increases, the data becomes sparser in the space it occupies, making it more difficult to analyze and requiring significantly more data to achieve accurate models. Here are some key aspects of the curse of dimensionality:

- **Data Sparsity**: In high-dimensional spaces, data points are spread out, which means that the volume of the space increases exponentially with the number of dimensions. This sparsity makes it difficult to find patterns or to generalize from the data without a huge amount of data.
- **Distance Concentration**: As dimensionality increases, the contrast between different data points becomes less pronounced. This means that traditional notions of distance (such as Euclidean distance) become less useful because all points tend to be approximately equidistant from each other.
- **Combinatorial Explosion**: With each additional dimension, the number of possible combinations of features increases exponentially. This can lead to a combinatorial explosion, where the number of potential configurations becomes unmanageable.
- **Increased Computational Complexity**: High-dimensional data sets require more computational resources for processing and analysis. Algorithms that work well in low-dimensional spaces may become infeasible in high-dimensional spaces due to the increased computational complexity.
- **Overfitting**: In machine learning, having too many features relative to the number of observations can lead to overfitting, where the model learns the noise in the training data instead of the underlying pattern. This results in poor generalization to new data.

To combat the curse of dimensionality, techniques such as dimensionality reduction are used. Principal Component Analysis (PCA) is one such technique that transforms high-dimensional data into a lower-dimensional space while retaining most of the variance in the data[1]. Other methods include feature selection, where irrelevant or redundant features are removed, and manifold learning, which seeks to uncover the low-dimensional structure embedded in high-dimensional data.

Understanding and addressing the curse of dimensionality is crucial for effective data analysis, especially in fields that deal with large and complex data sets, such as bioinformatics, image processing, and machine learning.



Different ways to remove Curse of dimensionality:

There are several strategies to combat the Curse of Dimensionality, which can help in managing high-dimensional data and improving the performance of machine learning models. Here are some of the common methods:

- **Feature Selection**: This involves selecting a subset of the most relevant features for use in model construction. Feature selection techniques can be filter-based, wrapper-based, or embedded methods[1].
- **Feature Extraction**: This is the process of transforming high-dimensional data into a lower-dimensional space. Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are examples of feature extraction techniques that reduce the number of features while retaining the most important information[2].
- **Regularization**: Techniques like L1 (Lasso) and L2 (Ridge) regularization can help prevent overfitting by penalizing large weights in the model, effectively reducing the complexity of the model.
- **Ensemble Methods**: Using ensemble methods like Random Forests can help in dealing with high-dimensional data by building multiple models and aggregating their predictions.
- **Dimensionality Reduction**: Techniques like PCA, t-Distributed Stochastic Neighbor Embedding (t-SNE), and Uniform Manifold Approximation and Projection (UMAP) can reduce the dimensionality of the data while preserving its structure[1].
- **Bin or Bucket Features**: This involves grouping continuous features into bins or buckets to reduce the number of unique values and, consequently, the dimensionality.
- **Random Projection**: This is a mathematical technique where the original high-dimensional data is projected onto a lower-dimensional subspace using random linear transformations.

- **Manifold Learning**: Techniques like Isomap, Locally Linear Embedding (LLE), and others assume that the data lies on a lower-dimensional manifold within the higher-dimensional space. These methods aim to uncover the manifold structure to reduce dimensionality.
- **Increasing Sample Size**: While not always feasible, increasing the number of samples can help mitigate the sparsity caused by high dimensionality.
- **Using Domain Knowledge**: Expert knowledge about the domain can help in identifying and removing irrelevant features from the data.

Each of these methods has its own advantages and is suitable for different types of data and problems. The choice of method depends on the specific characteristics of the data and the goals of the analysis or modeling effort[1][2]. It's often beneficial to combine multiple techniques

Feature Selection vs Feature Extraction:

Feature Selection and Feature Extraction are two fundamental techniques used in machine learning for dimensionality reduction, but they serve different purposes and are used in different scenarios. Here's a comparison of the two:

Feature Selection:

- **Purpose**: The goal of feature selection is to select a subset of the most relevant features from the original set of features. It aims to reduce the dimensionality by eliminating redundant or irrelevant features that do not contribute to the predictive power of the model[1].
- **Method**: Feature selection methods can be filter-based, wrapper-based, or embedded. Filter methods rank features based on statistical tests, wrapper methods use a predictive model to evaluate feature subsets, and embedded methods perform feature selection during the model training process[1].
- **Outcome**: The outcome of feature selection is a reduced feature set that retains the original meaning of the features. The selected features are expected to provide the most significant information needed for the modeling task[1].

Feature Extraction:

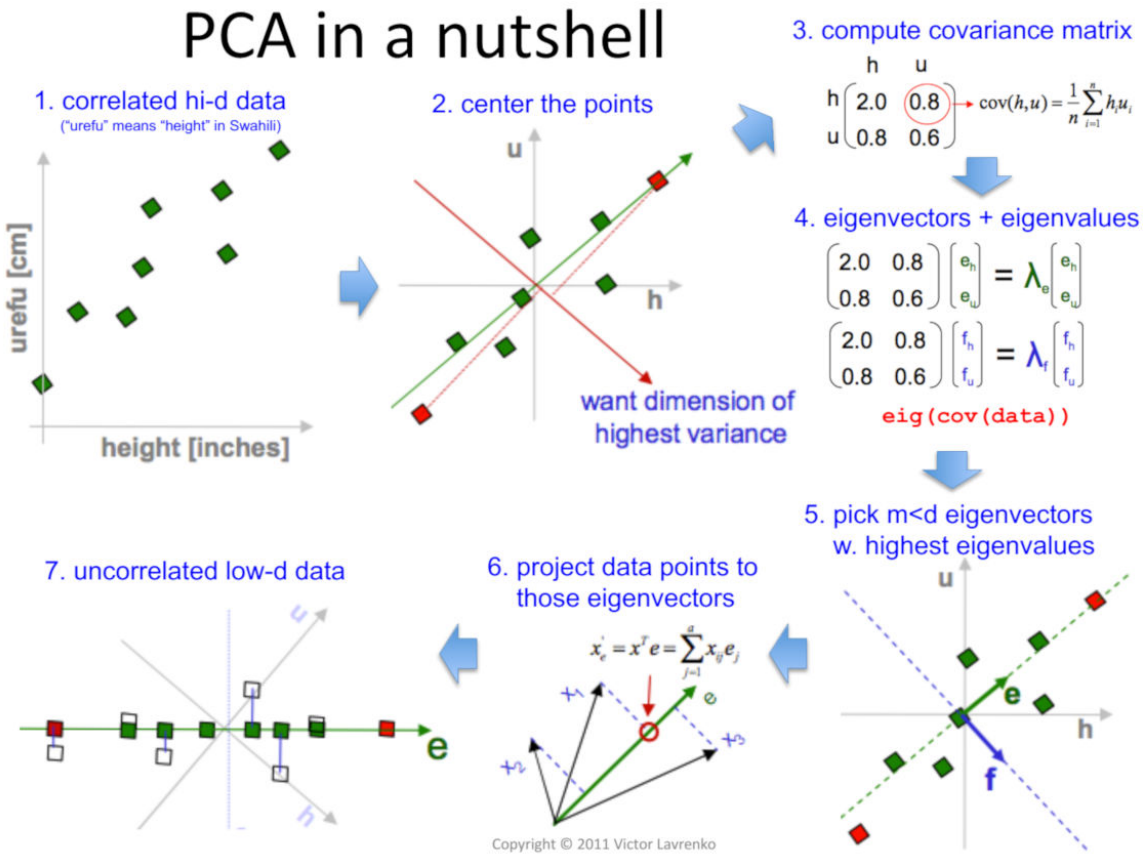
- **Purpose**: Feature extraction transforms the original data into a lower-dimensional space. The goal is to create new features that capture the most important information in the data, often in a way that enhances the separability of classes or clusters[2].
- **Method**: This involves mathematical transformations of the data, such as PCA, which creates new features (principal components) that are linear combinations of the original features[2].
- **Outcome**: The outcome of feature extraction is a new set of features that are different from the original features. These new features are designed to reduce the dimensionality while preserving as much of the relevant information as possible[2].

In summary, feature selection is about choosing a subset of the original features, while feature extraction is about creating new features from the original data. The choice between the two depends on the specific problem, the nature of the data, and the goals of the analysis. Feature selection is typically used when it's important to maintain the original features, whereas feature extraction is used when transforming the data into a new space could lead to better insights or improved model performance.

What is Principal Component Analysis? (Feature Extraction):

Principal Component Analysis (PCA) is a statistical procedure that involves an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This technique is

widely used in exploratory data analysis and for making predictive models. The main goals of PCA are to reduce the dimensionality of a data set and to identify new meaningful underlying variables.



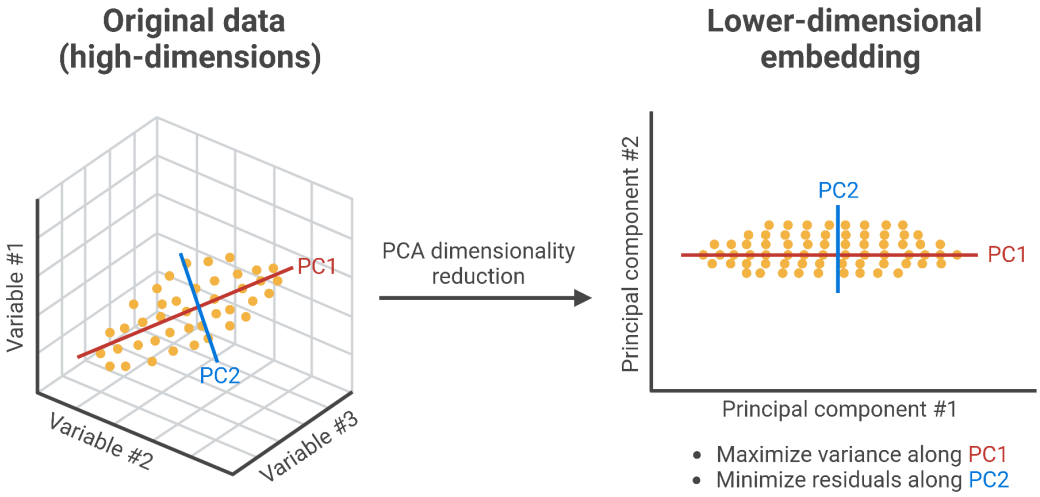
Why we use PCA(Principal Component Analysis)?

We use Principal Component Analysis (PCA) for several reasons, primarily related to the simplification of complex data. Here are some of the key reasons why PCA is widely used:

- **Reduction of Complexity**: PCA reduces the dimensionality of the data, making it easier to visualize and understand, especially when dealing with high-dimensional datasets.
- **Retention of Important Information**: Despite reducing the number of variables, PCA retains the most important information in the data by transforming the original variables into a new set of variables (principal components) that summarize the essential patterns.
- **Removal of Correlation**: PCA helps to remove multicollinearity among the variables by transforming them into orthogonal (uncorrelated) principal components, which simplifies the modeling process.
- **Improvement of Algorithm Performance**: Many machine learning algorithms perform better with lower-dimensional data. PCA can improve the efficiency and accuracy of these algorithms by reducing the input features without losing significant data variance.
- **Noise Reduction**: PCA can also act as a noise filter, removing the effects of irrelevant or noisy features from the data.
- **Feature Extraction**: PCA can be used to discover hidden patterns in the data by identifying the principal components that capture the most variance, which can lead to new insights about the underlying structure of the data.

In summary, PCA is a powerful tool for data preprocessing, feature extraction, and exploratory data analysis. It helps to simplify complex data, making it more manageable for analysis and visualization, and it can enhance the performance of predictive models.

Principal Component Analysis (PCA) Transformation



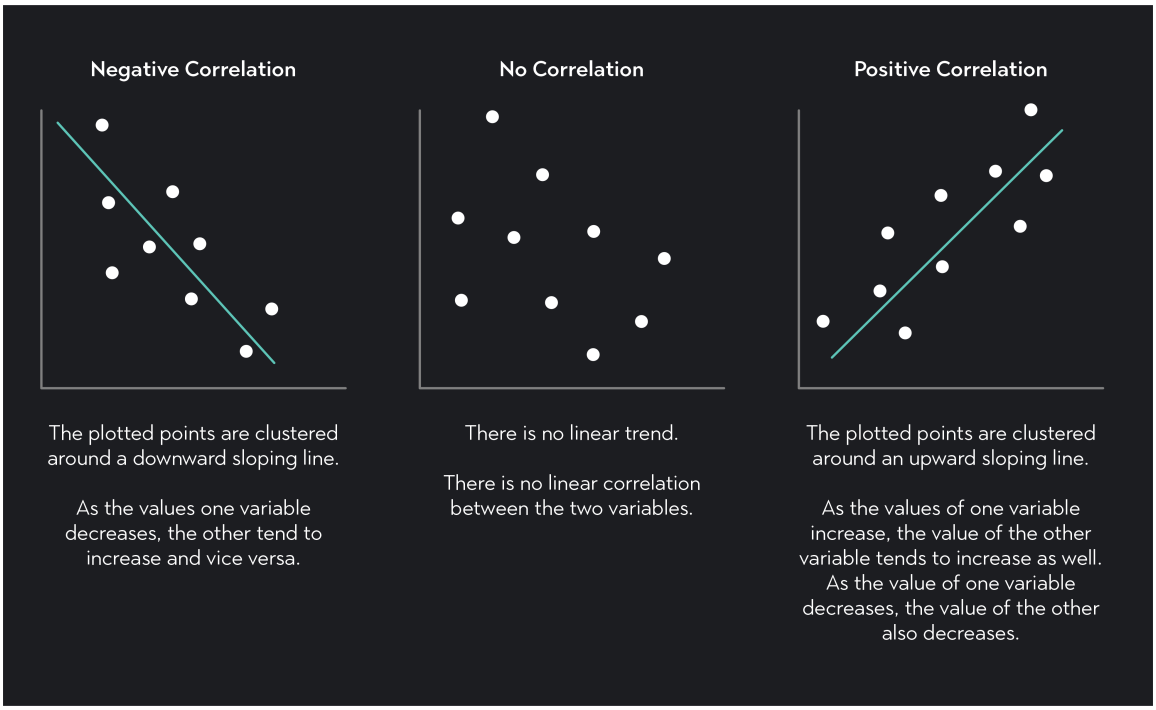
How PCA(Principal Component Analysisa) works:

Principal Component Analysis (PCA) works by following a mathematical procedure to transform the original correlated variables into a new set of uncorrelated variables called principal components. These principal components are ordered so that the first few retain most of the variation present in all of the original variables. Here's a step-by-step breakdown of how PCA works:

1. **Standardization**: The first step in PCA is to standardize the data. This involves scaling the data so that each feature has a mean of 0 and a standard deviation of 1. This is important because PCA is affected by the scale of the variables.
2. **Covariance Matrix Computation**: Next, the covariance matrix is computed. This matrix captures the covariance (a measure of how much two variables change together) between every pair of features in the data. If the data is standardized, the covariance matrix is the same as the correlation matrix.
3. **Eigendecomposition**: The covariance matrix is then decomposed into its eigenvectors and eigenvalues. The eigenvectors represent the directions of the maximum variance in the data (the principal components), and the eigenvalues represent the magnitude of the variance in those directions. In other words, the eigenvalues explain the variance of the data along the new feature axes.
4. **Sort Eigenvectors**: The eigenvectors are sorted by their corresponding eigenvalues in descending order. This ranking is important because it determines the order of the principal components.
5. **Projection**: Finally, the data is projected onto the new feature space using the sorted eigenvectors. This results in a new dataset where the features are the principal components that we calculated. The first principal component has the highest variance followed by the second, and so on.

The result of PCA is a set of principal components that are linearly uncorrelated and ordered by the amount of variance they explain in the original dataset. The first principal component explains the most variance, the second principal component explains the second most, and so on. By selecting the top (k) principal components, where (k) is a number chosen based on the desired level of variance retention, one can reduce the dimensionality of the data while still retaining most of the important information.

This technique is particularly useful in situations where the dimensionality of the data is so high that it becomes difficult to analyze or visualize. PCA provides a means to reduce the number of dimensions without losing significant information, making it easier to explore and understand the data.



Pearson Correlation Coefficient: The Pearson Correlation Coefficient, denoted as (r), is a measure of the linear correlation between two variables (X) and (Y). It ranges from -1 to +1, with -1 indicating a perfect negative linear correlation, +1 indicating a perfect positive linear correlation, and 0 indicating no linear correlation. The coefficient is calculated as the covariance of the two variables divided by the product of their standard deviations.

$$r = \frac{N\sum xy - \sum x \sum y}{\sqrt{[N\sum x^2 - (\sum x)^2]\ [N\sum y^2 - (\sum y)^2]}}$$

Where,

N = Number of pairs of scores

$\sqrt{x}$= Sum of x Scores

$\sqrt{y}$= Sum of y scores

$\sqrt{xy}$= sum of the products of paired scores

$\sqrt{x^2}$= sum of the squared x scores

$\sqrt{y^2}$= sum of the squared y scores

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

Breast Cancer Analysis

In [ ]:
```
# Load the dataset
from sklearn.datasets import load_breast_cancer
```

In [ ]:
```
cancer_data = load_breast_cancer()
```

In [ ]:
```
# Check the datatype of cancer_data
data_type = type(cancer_data)
print(data_type)
```

```
<class 'sklearn.utils._bunch.Bunch'>
```

In [ ]:
```
# Printing the feature column
print(cancer_data['feature_names'])
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

In [ ]:
```
# Printing the target column
print(cancer_data['target'])
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

In [ ]:
```
# Convert the dataset into DataFrame
can_df = pd.DataFrame(cancer_data.data, columns=cancer_data.feature_names)
```

In [ ]:
```
# Viewing the column name and target variable together

# Create a DataFrame with feature data
can_df_tar = pd.DataFrame(cancer_data.data, columns=cancer_data.feature_name

# Add the target column to the DataFrame
can_df_tar['target'] = pd.Series(cancer_data.target)

# Now you can print the DataFrame with the target column included
print(can_df_tar.head(4))
```

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness
\
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597

   mean fractal dimension  ...  worst texture  worst perimeter  worst area
\
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7

   worst smoothness  worst compactness  worst concavity  worst concave poi
nts  \
0            0.1622             0.6656           0.7119                0.2
654
1            0.1238             0.1866           0.2416                0.1
860
2            0.1444             0.4245           0.4504                0.2
430
3            0.2098             0.8663           0.6869                0.2
575

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0

[4 rows x 31 columns]
```

In [ ]:
```
print(can_df.head(4))
```

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness
\
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597

   mean fractal dimension  ...  worst radius  worst texture  worst perimet
er  \
0                 0.07871  ...         25.38          17.33           184.
60
1                 0.05667  ...         24.99          23.41           158.
80
2                 0.05999  ...         23.57          25.53           152.
50
3                 0.09744  ...         14.91          26.50            98.
87

   worst area  worst smoothness  worst compactness  worst concavity  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869

   worst concave points  worst symmetry  worst fractal dimension
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300

[4 rows x 30 columns]
```

In [ ]:
```
# Printing the keys of dataset
can_df.keys()
```

Out[180]:
```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension erro
r',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimensio
n'],
      dtype='object')
```

In [ ]:
```
can_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

Pearson Correlation Matrix / Coefficient

The Pearson Correlation measures the linear dependence between two variables X and Y. The resulting coefficient is a value between -1 and 1 inclusive, where:

1: total positive linear correlation,

0: no linear correlation,the two variables most likely do not affect each other

-1: total negative linear correlation.

P-value: What is this P-value? The P-value is the probability value that the correlation between these two variables is statistically significant. Normally, we choose a significance level of 0.05, which means that we are 95% confident that the correlation between the variables is significant.

By convention, when the p-value is < 0.001 we say there is strong evidence that the correlation is significant,

p-value is < 0.05, there is moderate evidence that the correlation is significant,

p-value is < 0.1, there is weak evidence that the correlation is significant,and

p-value is > 0.1, there is no evidence that the correlation is significant.

Out[182]:

```
can_df.corr(method='pearson') # By default, it will take the 'pearson' method
```

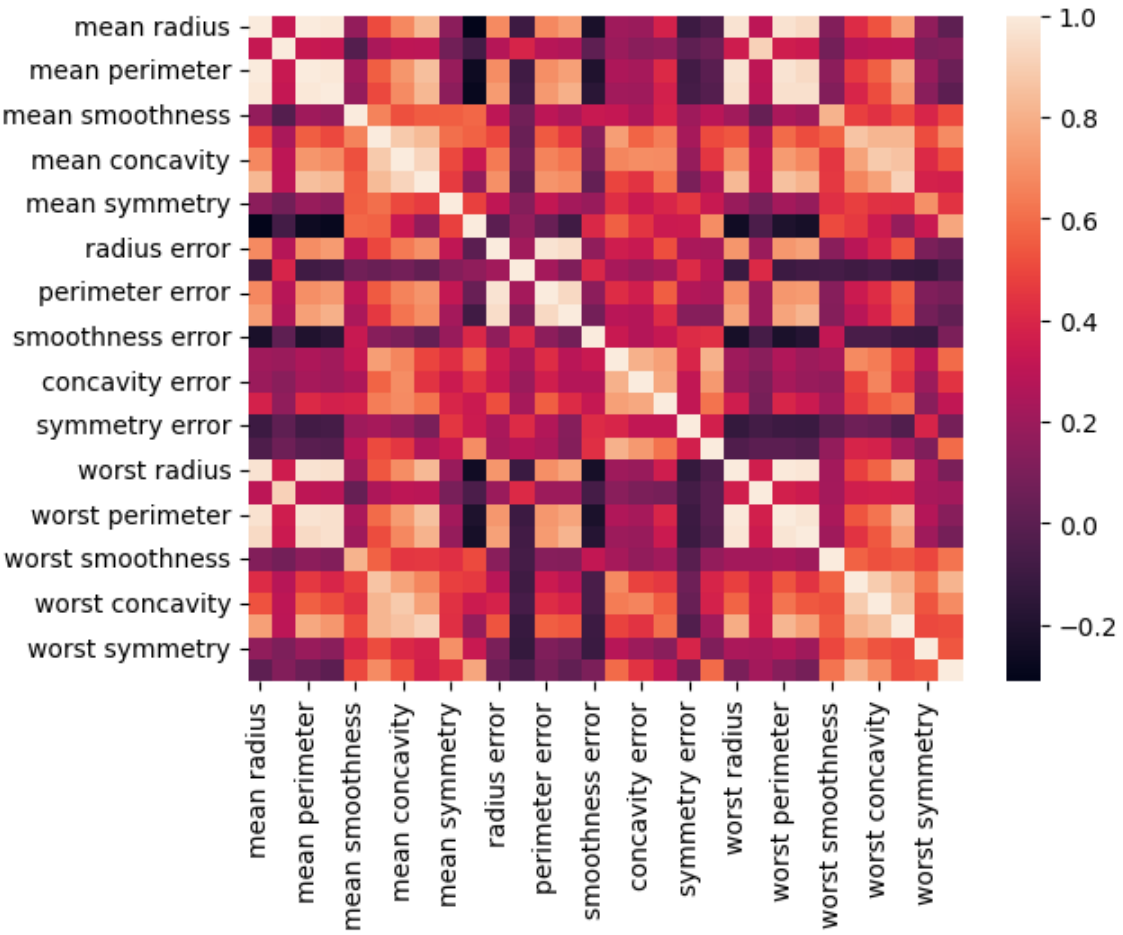
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity
mean radius	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.67676
mean texture	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.30241
mean perimeter	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.71613
mean area	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.68598
mean smoothness	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.52198
mean compactness	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.88312
mean concavity	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.00000
mean concave points	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.92139
mean symmetry	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.50066
mean fractal dimension	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.33678
radius error	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.63192
texture error	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.07621
perimeter error	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.66039
area error	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.61742
smoothness error	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.09856
compactness error	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.67027
concavity error	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.69127
concave points error	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.68326
symmetry error	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.17800
fractal dimension error	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.44930
worst radius	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.68823
worst texture	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.29987
worst perimeter	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.72956
worst area	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.67598
worst smoothness	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.44882
worst compactness	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.75496
worst concavity	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.88410
worst concave points	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.86132

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity
worst symmetry	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.40946
worst fractal dimension	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.51493

30 rows × 30 columns

In [ ]:
```
sns.heatmap(can_df.corr()) # More closer to 1, it denotes more connection or
```

Out[183]: <Axes: >



Standard Scaler

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [ ]:
```
scaler.fit(can_df)
```

Out[185]: StandardScaler()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```
scaled_can_df = scaler.transform(can_df) # Also we can use fit_transorm inst
scaled_can_df
```

Out[186]:
```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

In [ ]:
```
# Checking the shape
scaled_can_df.shape
```

Out[187]:
```
(569, 30)
```

Apply PCA Algo

In [ ]:
```
from sklearn.decomposition import PCA
```

In [ ]:
```
pca = PCA(n_components = 2) # n_components = 2 means that the PCA will reduc
# This is useful when you want to visualize high-dimensional data in a 2D pl
```

In [ ]:
```
# Fit and transform the scaled data into PCA
can_pca = pca.fit_transform(scaled_can_df)
can_pca
```

Out[190]:
```
array([[ 9.19283683,  1.94858307],
       [ 2.3878018 , -3.76817174],
       [ 5.73389628, -1.0751738 ],
       ...,
       [ 1.25617928, -1.90229671],
       [10.37479406,  1.67201011],
       [-5.4752433 , -0.67063679]])
```

In [ ]:
```
# We can use it only after the usage of fit and transform
pca.explained_variance_ # amount of variance explained by each of the select
```

Out[191]:
```
array([13.30499079,  5.7013746 ])
```

In [ ]:
```
# Checking the shape
can_pca.shape # Feature reduced to 2 from 30
```

Out[192]:
```
(569, 2)
```

In [ ]:
```
# Viwing the dtype of can_pca
data_type = type(can_pca)
print(data_type)
```

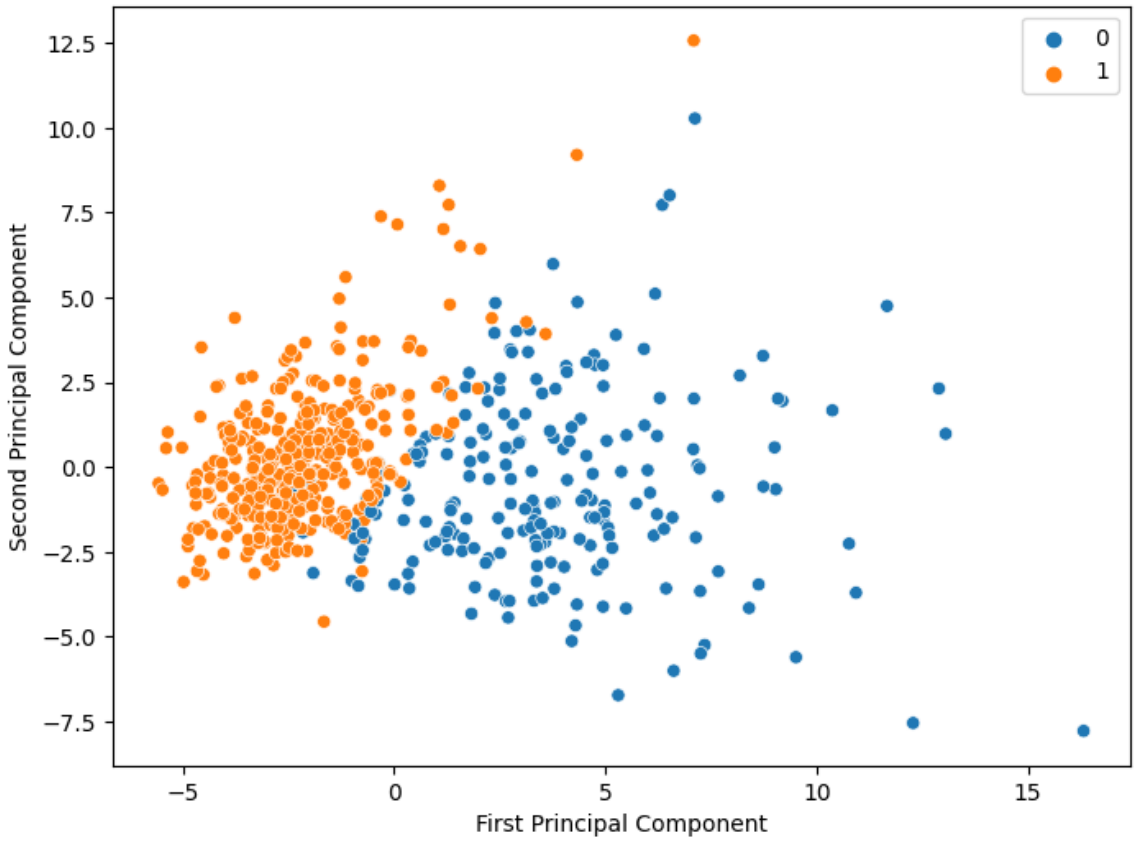
```
<class 'numpy.ndarray'>
```

In [ ]:
```
# Viewing the columns and giving the name which is generated by PCA
# Convert can_pca to a DataFrame
can_pca_df = pd.DataFrame(can_pca, columns=['PC1', 'PC2'])

# Now you can view the DataFrame with named columns
print(can_pca_df.head())
```

```
        PC1        PC2
0  9.192837   1.948583
1  2.387802  -3.768172
2  5.733896  -1.075174
3  7.122953  10.275589
4  3.935302  -1.948072
```

Viewing the PCA in visualization

In [ ]:
```
plt.figure(figsize = (8,6))

sns.scatterplot(x = can_pca[:,0],
                y = can_pca[:,1],
                hue = cancer_data['target']) #  'target' array contains the labe

plt.xlabel("First Principal Component")
plt.ylabel("Second Principal Component")
plt.show()
```

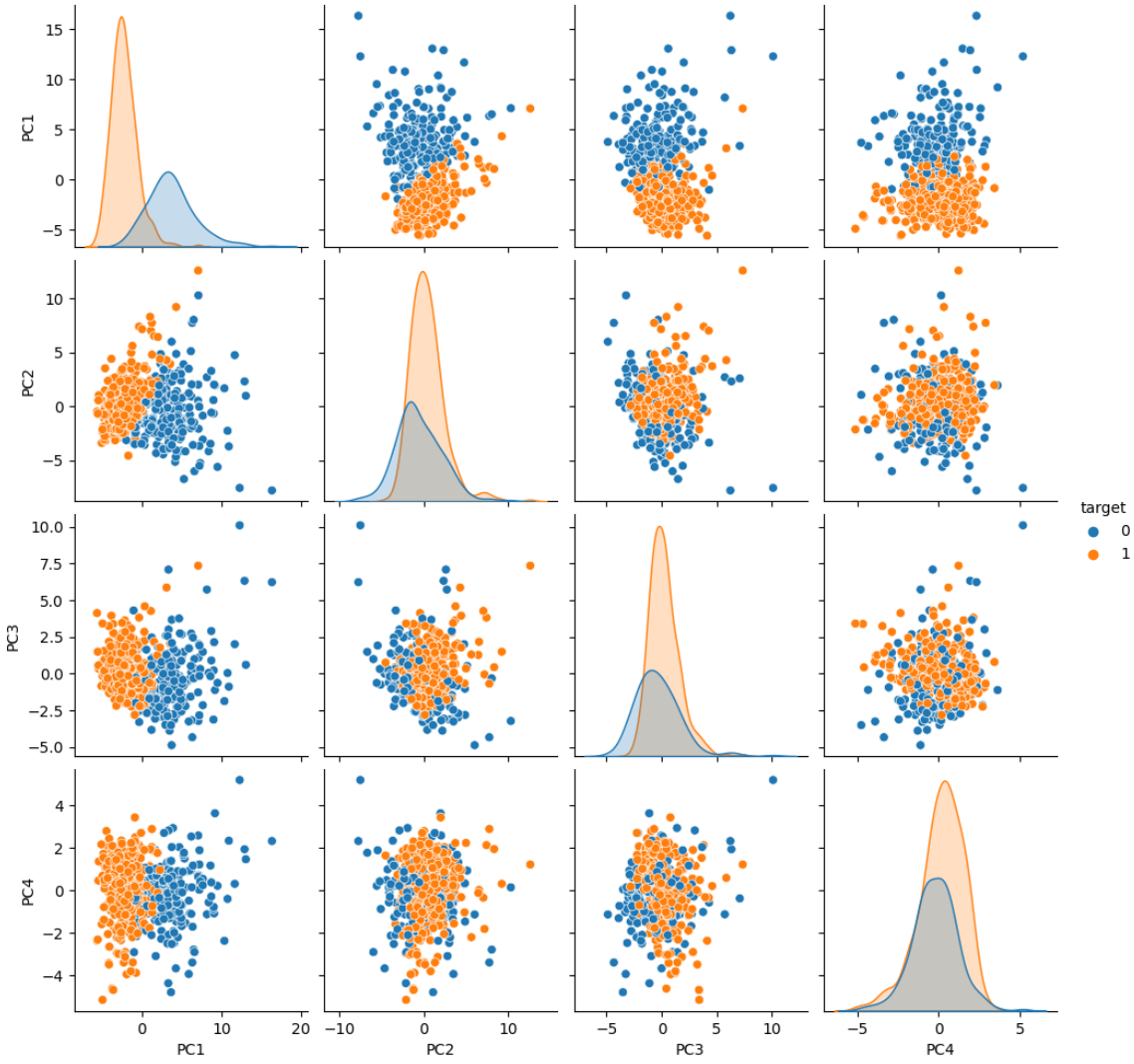


Viewing in n_components = 4

In [ ]:
```
pca = PCA(n_components = 4)

can_pca_4 = pca.fit_transform(scaled_can_df)
can_pca_4
```

Out[196]:
```
array([[ 9.19283685,  1.94858381, -1.12317221,  3.63371043],
       [ 2.38780179, -3.76817181, -0.52929196,  1.11826546],
       [ 5.73389628, -1.07517394, -0.55174664,  0.91208694],
       ...,
       [ 1.25617927, -1.90229689,  0.56273196, -2.08922211],
       [10.37479406,  1.67200994, -1.8770281 , -2.35602646],
       [-5.47524329, -0.67063632,  1.49043886, -2.2991701 ]])
```

In [ ]:
```
# Convert the PCA components into a DataFrame
df_pca = pd.DataFrame(can_pca_4, columns=['PC1', 'PC2', 'PC3', 'PC4'])

# Add the target column to the DataFrame
df_pca['target'] = cancer_data.target # hue parameter in pair plot expects t

# Create a pair plot
sns.pairplot(df_pca, hue='target')
plt.show()
```



Implementing Logistic Regression

In [ ]:
```
x = can_df_tar.drop('target', axis=1)
x
```

Out[198]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	r symn
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.

569 rows × 30 columns

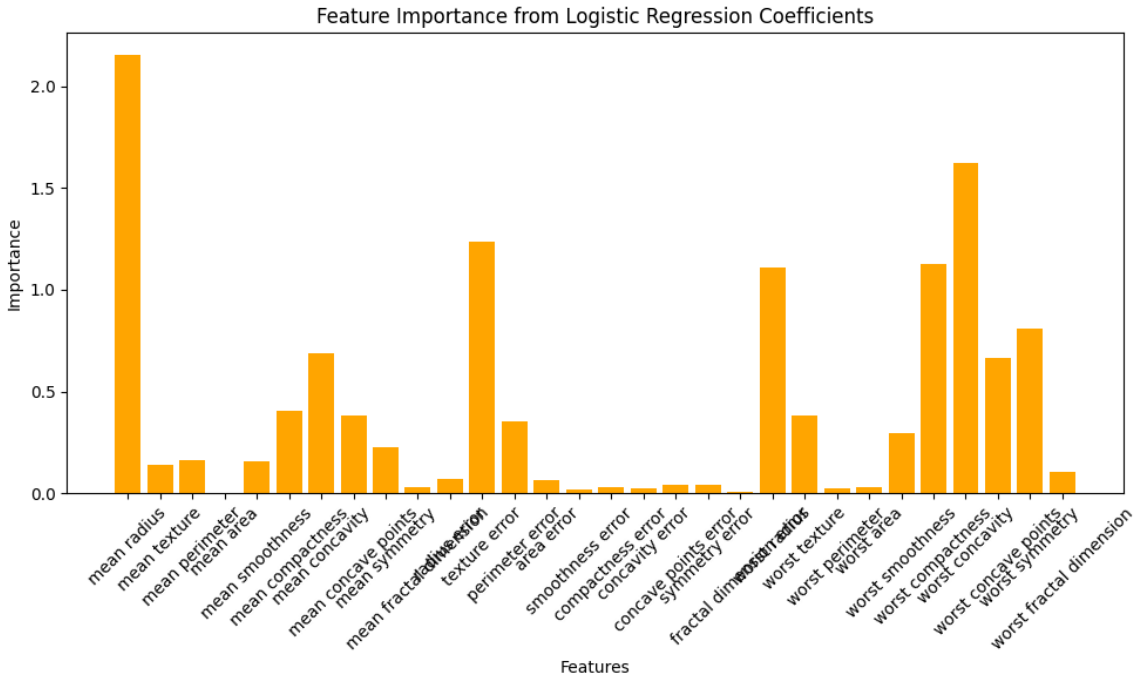
In [ ]:
```
y = can_df_tar['target']
y
```

Out[199]:
```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: target, Length: 569, dtype: int64
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, ran

# Initialize and train the classifier
log = LogisticRegression(max_iter=800)
log.fit(X_train, y_train)

# Evaluate the model
print("Training accuracy:", log.score(X_train, y_train))
print("Testing accuracy:", log.score(X_test, y_test))
```

```
Training accuracy: 0.9582417582417583
Testing accuracy: 0.956140350877193

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:
458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [ ]:
```
# Printing the feature importance

feature_importance = abs(log.coef_[0]) # absolute value of the coefficients

# Assuming 'feature_names' is a list of feature names corresponding to the c
feature_names = x.columns.tolist()

# Create a bar plot of feature importance
plt.figure(figsize=(10, 6))  # Optional: Adjust the figure size as needed
plt.bar(feature_names, feature_importance, color='orange')

# Add labels and title
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance from Logistic Regression Coefficients')

# Rotate the feature names on the x-axis for better readability
plt.xticks(rotation=45)

# Display the plot
plt.tight_layout()  # Adjust the layout to fit the rotated x-tick labels
plt.show()
```



Implementing PCA in Logistic Regression

In [ ]:
```
x_pca = can_pca
x
```

Out[202]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	r symn
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.

569 rows × 30 columns

In [ ]:
```
y = can_df_tar['target']
y
```

Out[203]:
```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: target, Length: 569, dtype: int64
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x_pca, y, test_size=0.2,

# Initialize and train the classifier
log_pca = LogisticRegression(max_iter=800)
log_pca.fit(X_train, y_train)

# Evaluate the model
print("Training accuracy:", log_pca.score(X_train, y_train))
print("Testing accuracy:", log_pca.score(X_test, y_test))
```

```
Training accuracy: 0.945054945054945
Testing accuracy: 0.9912280701754386
```

In [ ]:
```
# Printing the most important features

# Get the absolute values of the coefficients
feature_importance = abs(log_pca.coef_[0])

# Create a bar plot of feature importance
plt.bar(range(len(feature_importance)), feature_importance)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance from PCA Logistic Regression Coefficients')
plt.show()
```