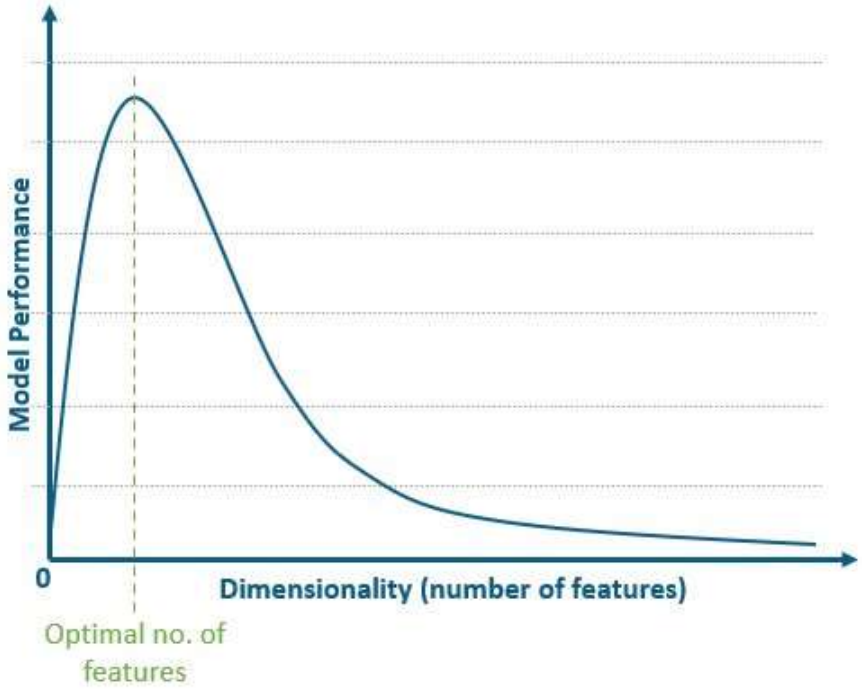
What is the Curse of Dimensionality?

It refers to the difficulties a machine learning algorithm faces when working with data in the higher dimensions(features), that did not exist in the lower dimensions.

This means, that you are required to have an `Optimal Set` of features to predict better accuracy of the model.



With the increase in the data dimensions, your model –

- would also increase in `Computation Cost`
- would decrease the `Performance of the Model`
- would become increasingly dependent on the training data for production.

This leads to overfitting of the model, so even though the model performs really well on training data, it may fails on test data.

Dimensionality Reduction

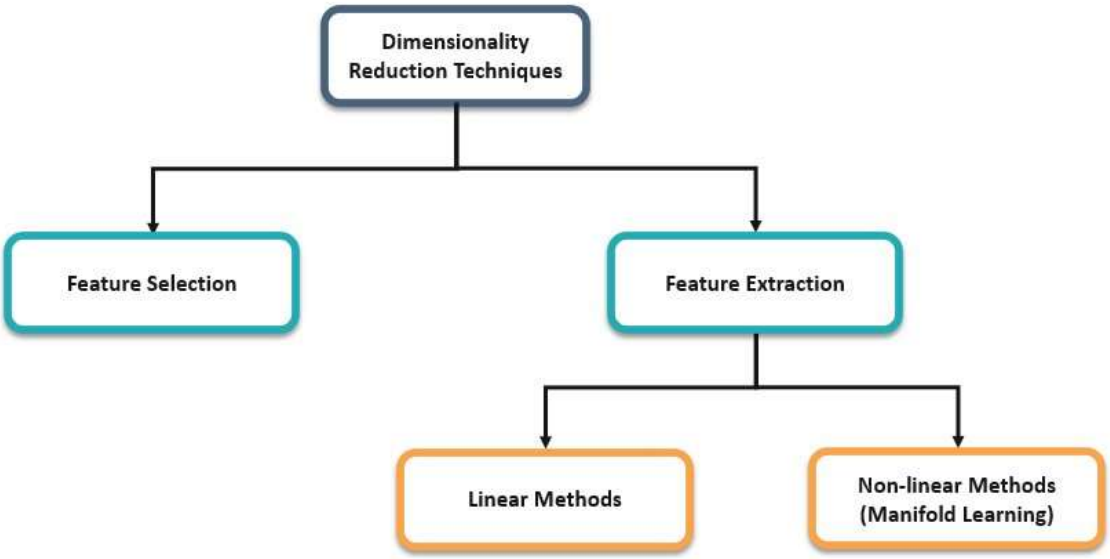
It is a process of reducing the dimension of your data to a few principal features in the original dataset, to remove the curse of dimensionality.

Why is Dimensionality Reduction necessary?

- **Avoids overfitting** – the lesser assumptions a model makes, the simpler it will be.
- **Easier computation** – the lesser the dimensions, the faster the model trains.
- **Improved model performance** – removes redundant features and noise, lesser misleading data improves model accuracy.
- **Lower dimensional data** - requires less storage space.

How is Dimensionality Reduction done?

These techniques are divided into two broad categories for dimensionality reduction depending on the problem and the data:



a. **Feature Selection**: By only keeping the most relevant variables from the original dataset

i. Correlation

ii. Forward Selection

iii. Backward Elimination

iv. Select K Best

v. Missing value Ratio

b. **Feature Extraction**: By finding a smaller set of new variables, each being a combination of the input variables, containing basically the same information as the input variables.

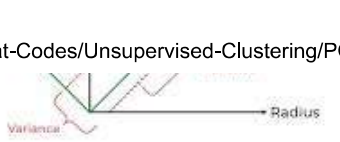
i. PCA(Principal Component Analysis)

ii. LDA(Linear Discriminant Analysis)

iii. t-SNE (t-distributed Stochastic Neighbor Embedding)

PCA(Principal Component Analysis)

is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables.



It tends to find the direction of maximum variation (spread) in data. PCA is more useful when dealing with 3 or higher-dimensional data.

PCA can be used for anomaly detection and outlier detection because they will not be part of the data as it would be considered noise by PCA.

Moreover, Principal Component Analysis (PCA) is an unsupervised learning algorithm technique used to examine the interrelations among a set of variables.

Step-By-Step Explanation of PCA (Principal Component Analysis)

Step 1: Standardization

First, we need to standardize our dataset to ensure that each variable has a mean of 0 and a standard deviation of 1.

$$Z = \frac{X-\mu}{\sigma}$$

Here,

- **mu** - is the mean of independent features
- **sigma** - is the standard deviation of independent features

Step2: Covariance Matrix Computation

Covariance is a statistical measure that quantifies how changes in one variable correspond to changes in another. To find the covariance we can use the formula:

$$cov(x1, x2) = \frac{\sum_{i=1}^{n}(x1_i - \bar{x1})(x2_i - \bar{x2})}{n-1}$$

Covariance can be positive, negative, or zero.

- Positive: As the x1 increases x2 also increases.
- Negative: As the x1 increases x2 also decreases.
- Zeros: No direct relation

Covariance Matrix

In a covariance matrix, each element represents the covariance between two variables. The main diagonal of the matrix contains the variances of individual variables.

Step 3: Compute Eigenvalues and Eigenvectors of Covariance Matrix to Identify Principal Components

Let A be a square nXn matrix and X be a non-zero vector for which

$$AX = \lambda X$$

for some scalar values \lambda, then \lambda is known as the eigenvalue of matrix A and X is known as the eigenvector of matrix A for the corresponding eigenvalue.

From the above equation, we can find the eigenvalues \lambda, and therefore corresponding eigenvector can be found using the equation AX = \lambda X.

Implementing PCA for Dimensionality Reduction

Problem Statement:

A large number of input dimensions can cause a model to slow down during execution. So, we perform Principal Component Analysis (PCA) on the model to speed up the fitting of the ML algorithm.

PCA projects data in the direction of increasing variance. The features having the highest variance are the principal components. Let's see how to implement PCA using Python.

Importing Libraries

In [2]:
```
import numpy as np
import pandas as pd
import plotly.express as px
```

In [20]:
```
# Setting a seed for reproducibility
np.random.seed(23)

# Generate data for class 1
mu_vec1 = np.array([0, 0, 0])
cov_mat1 = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
class1_sample = np.random.multivariate_normal(mu_vec1, cov_mat1, 20)

# Create a DataFrame for class 1
df_class1 = pd.DataFrame(class1_sample, columns=['feature1', 'feature2', 'feature3'])
df_class1['target'] = 1

# Generate data for class 2
mu_vec2 = np.array([1, 1, 1])
cov_mat2 = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
class2_sample = np.random.multivariate_normal(mu_vec2, cov_mat2, 20)

# Create a DataFrame for class 2
df_class2 = pd.DataFrame(class2_sample, columns=['feature1', 'feature2', 'feature3'])
df_class2['target'] = 0

# Concatenate both classes into a single DataFrame
df = pd.concat([df_class1, df_class2], ignore_index=True)

# Randomly sample 40 rows from the combined DataFrame
df = df.sample(40, random_state=23)

# Display the resulting DataFrame
df.head()
```

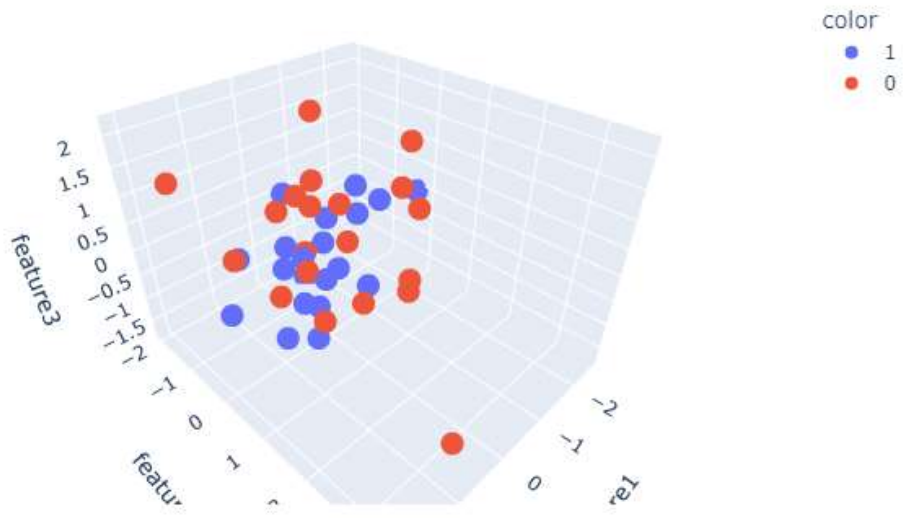
Out[20]:

	feature1	feature2	feature3	target

18	-0.331617	-1.632386	0.619114	1
23	1.010229	1.437830	2.327788	0
8	0.241106	-0.952510	-0.136267	1
22	1.676860	4.187503	-0.080565	0
33	2.823378	-0.332863	2.637391	0

In [21]:
```
# 3D Plot of data
fig = px.scatter_3d(df, x=df['feature1'], y=df['feature2'], z=df['feature3'],
                    color=df['target'].astype('str'))

fig.show()
```



Step 1 - Apply standard scaling

In [22]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

df.iloc[:,0:3] = scaler.fit_transform(df.iloc[:,0:3])
```

Step 2 - Find Covariance Matrix

In [23]:
```
# Creating a Covariance Matrix
covariance_matrix = np.cov([df.iloc[:,0],df.iloc[:,1],df.iloc[:,2]])
print('Covariance Matrix:\n', covariance_matrix)
```

```
Covariance Matrix:
 [[1.02564103 0.20478114 0.080118  ]
 [0.20478114 1.02564103 0.19838882]
```

```
[0.080118   0.19838882 1.02564103]]
```

Step 3 - Finding Eigen Vectors and Eigen Values for the Covariance-matrix

In [24]:
```
# Using Linear Algebra for extrating Eigen Vectores and Values
eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)
```

In [25]:
```
# Checking Eigen Vectors
eigen_vectors
```

Out[25]:
```
array([[-0.53875915, -0.69363291,  0.47813384],
       [-0.65608325, -0.01057596, -0.75461442],
       [-0.52848211,  0.72025103,  0.44938304]])
```

In [10]:
```
# Checking Eigen Values
eigen_values
```

Out[10]:
```
array([1.3536065 , 0.94557084, 0.77774573])
```

Step 4 – Concatenating the Principal Components with the target variable

In [26]:
```
# Selecting PC1 and PC2
pc = eigen_vectors[0:2]
pc
```

Out[26]:
```
array([[-0.53875915, -0.69363291,  0.47813384],
       [-0.65608325, -0.01057596, -0.75461442]])
```

In [27]:
```
# Transforming Dataset into two PCs
transformed_df = np.dot(df.iloc[:,0:3],pc.T)
# 40,3 - 3,2
new_df = pd.DataFrame(transformed_df,columns=['PC1','PC2'])
new_df['target'] = df['target'].values
new_df.head()
```

Out[27]:

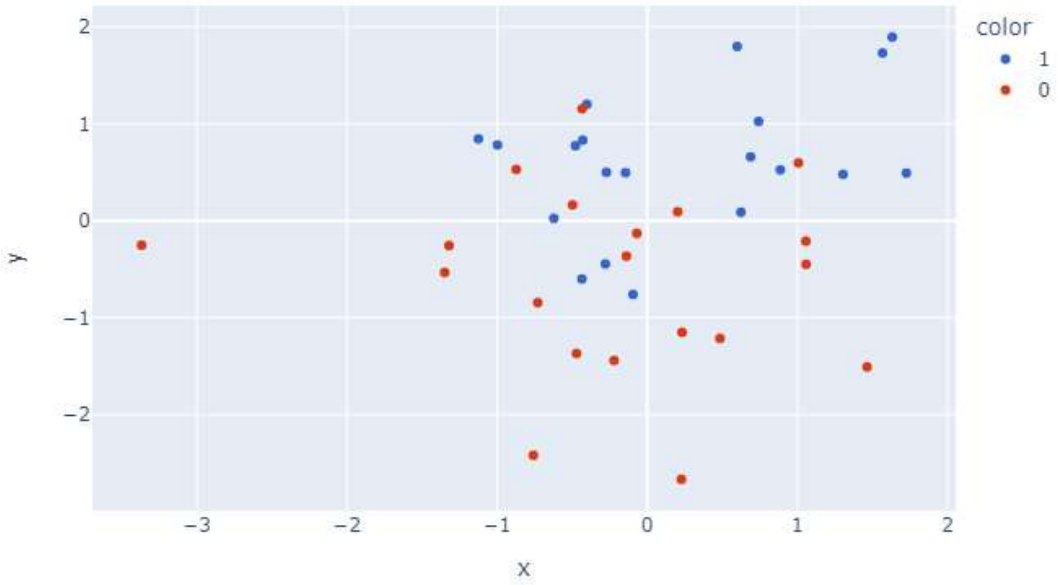
	PC1	PC2	target
0	1.726114	0.492511	1
1	-0.220797	-1.441911	0
2	0.688605	0.658084	1
3	-3.367715	-0.254627	0
4	0.227326	-2.669841	0

Step 5 – Visualizing the 2D Projection

In [28]:
```
new_df['target'] = new_df['target'].astype('str')
```

```
new_df['target'] = new_df['target'].astype('str')
fig = px.scatter(x=new_df['PC1'],
                 y=new_df['PC2'],
                 color=new_df['target'],
                 color_discrete_sequence=px.colors.qualitative.G10
                )

fig.show()
```



Hence dataset is successfully transformed.

Disadvantages of PCA:

PCA does not guarantee class separability which is why it is an unsupervised algorithm. In other words, PCA does not know whether the problem which we are solving is a regression or classification task.

Note:

I have performed this Step-by-Step PCA implementation for understanding purposes.

Use `scikit-learn` to implement PCA in Python:

In scikit-learn, the `fit_transform` method performs both fitting the PCA model on the data and transforming the data to the new subspace with reduced dimensions.