Feature Selection

Agenda

In this lesson, we will cover the following concepts with the help of a business use case:

- Feature Selection
- Dimensionality Reduction:
  - Dimensionality Reduction Techniques
  - Pros and Cons of Dimensionality Reduction
- Prominent Methods Used for Feature Selection:
  - Factor Analysis
  - Principal Component Analysis
  - LDA

What Is Feature Selection?

Feature selection is a method that helps in the inclusion of the significant variables that help form a model with good predictive power.

Features or variables that are redundant or irrelevant can negatively impact the performance of the model, thus it becomes necessary to remove them.

Benefits of Feature Selection

- It reduces overfitting as the unwanted variables are removed, and the focus is on the significant variables.
- It removes irrelevant information, which helps to improve the accuracy of the model's predictions.
- It reduces the computation time involved to get the model's predictions.

Dimensionality Reduction

Dimensionality reduction is the method of transforming a collection of data having large dimensions into data of smaller dimensions while ensuring that identical information is conveyed concisely.

Dimensionality Reduction Techniques

Some of the techniques used for dimensionality reduction are:

1. Imputing missing values
2. Dropping low-variance variables
3. Decision trees (DT)
4. Random forest (RF)
5. Reducing highly correlated variables
6. Backward feature elimination
7. Factor analysis
8. Principal component analysis (PCA)

Pros of Dimensionality Reduction

- It helps to compress data, reducing the storage space needed.
- It cuts down on computing time.
- It also aids in the removal of redundant features.

Cons of Dimensionality Reduction

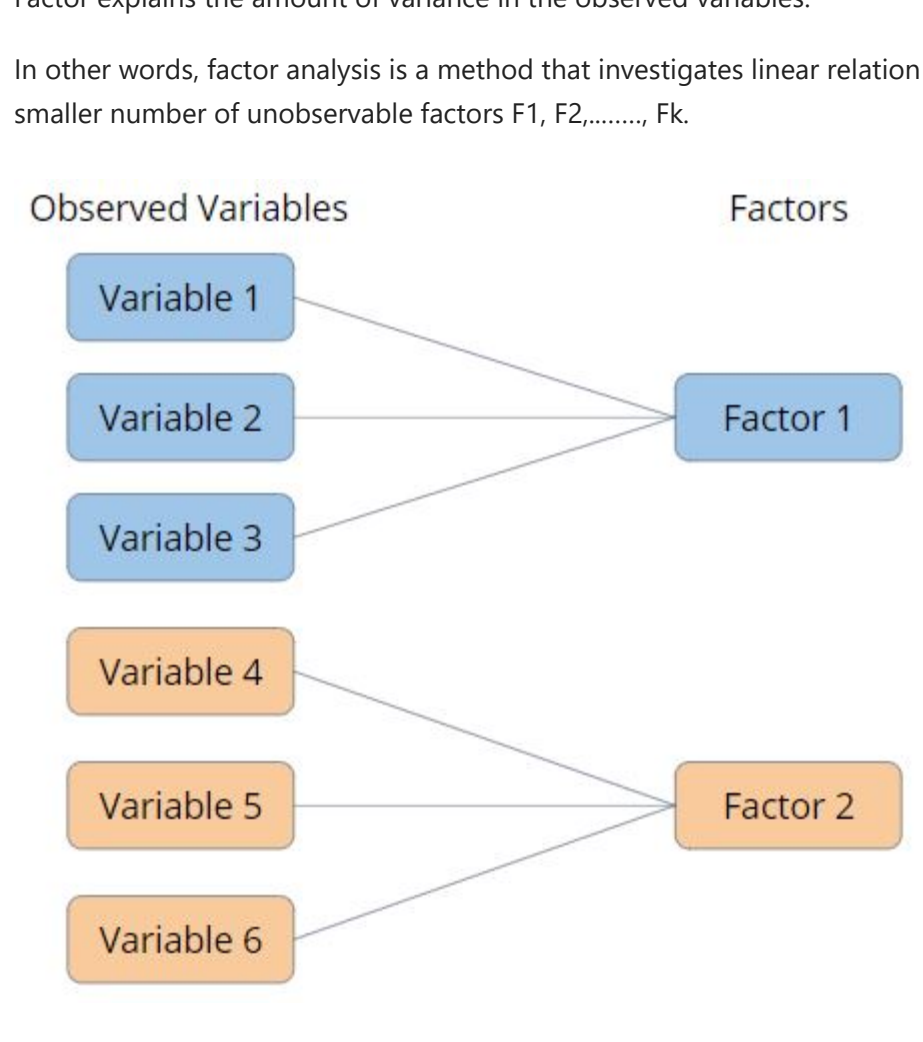
- Some data may will be lost as a result.
- PCA tends to find linear relationships between variables, which is not always ideal.
- When mean and covariance are insufficient to describe datasets, PCA fails.
- We use certain thumb rules when we do not know how many principal components to keep in practice.

Gist of Factor Analysis

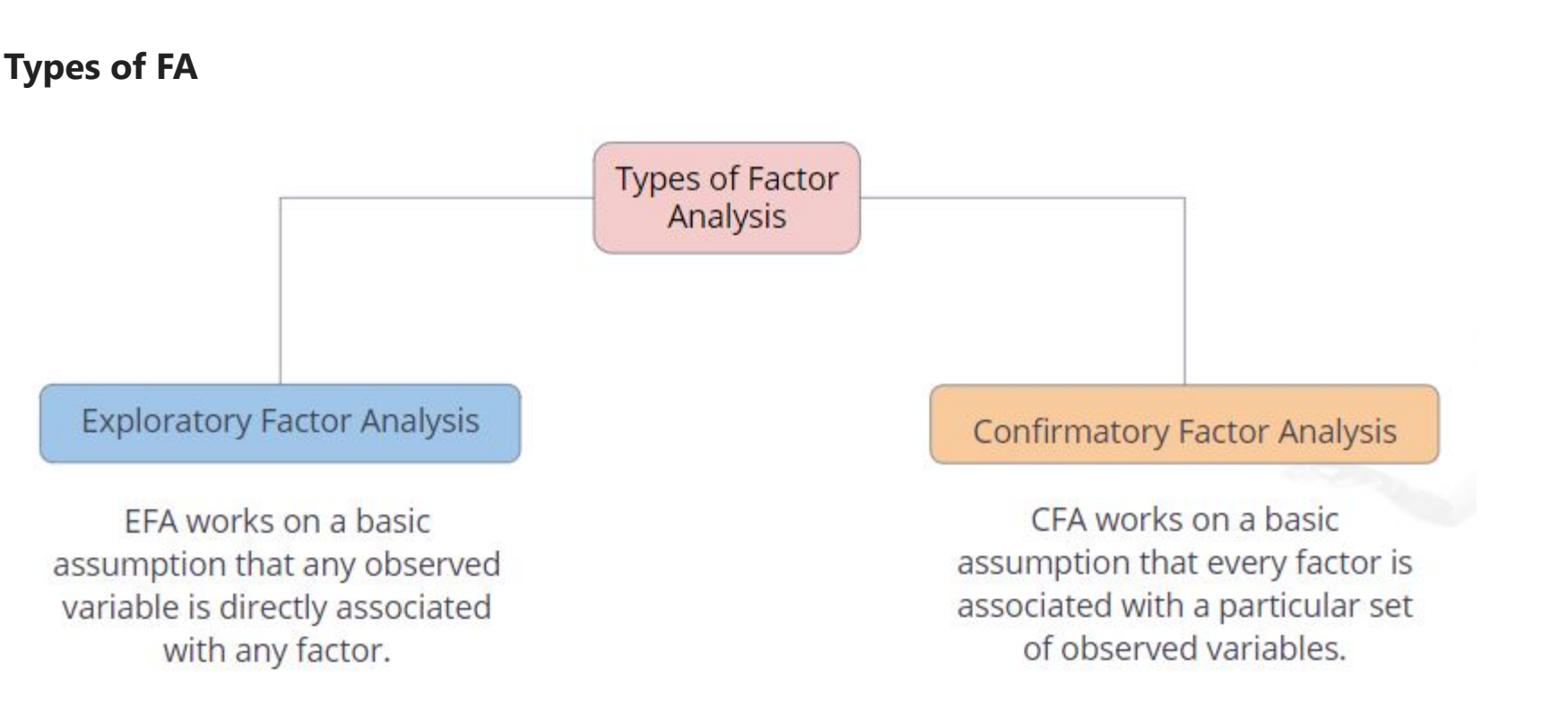
- Factor analysis is used to:
  - Explain variance among the observed variables
  - Condense the set of observed variables into the factors

$$Y_i = \beta_{i0} + \beta_{i1}F_1 + \beta_{i2}F_2 + (1)e_i$$

- Factor explains the amount of variance in the observed variables.
- In other words, factor analysis is a method that investigates linear relation of a number of variables of interest V1, V2,......., Vl, with a smaller number of unobservable factors F1, F2,........, Fk.



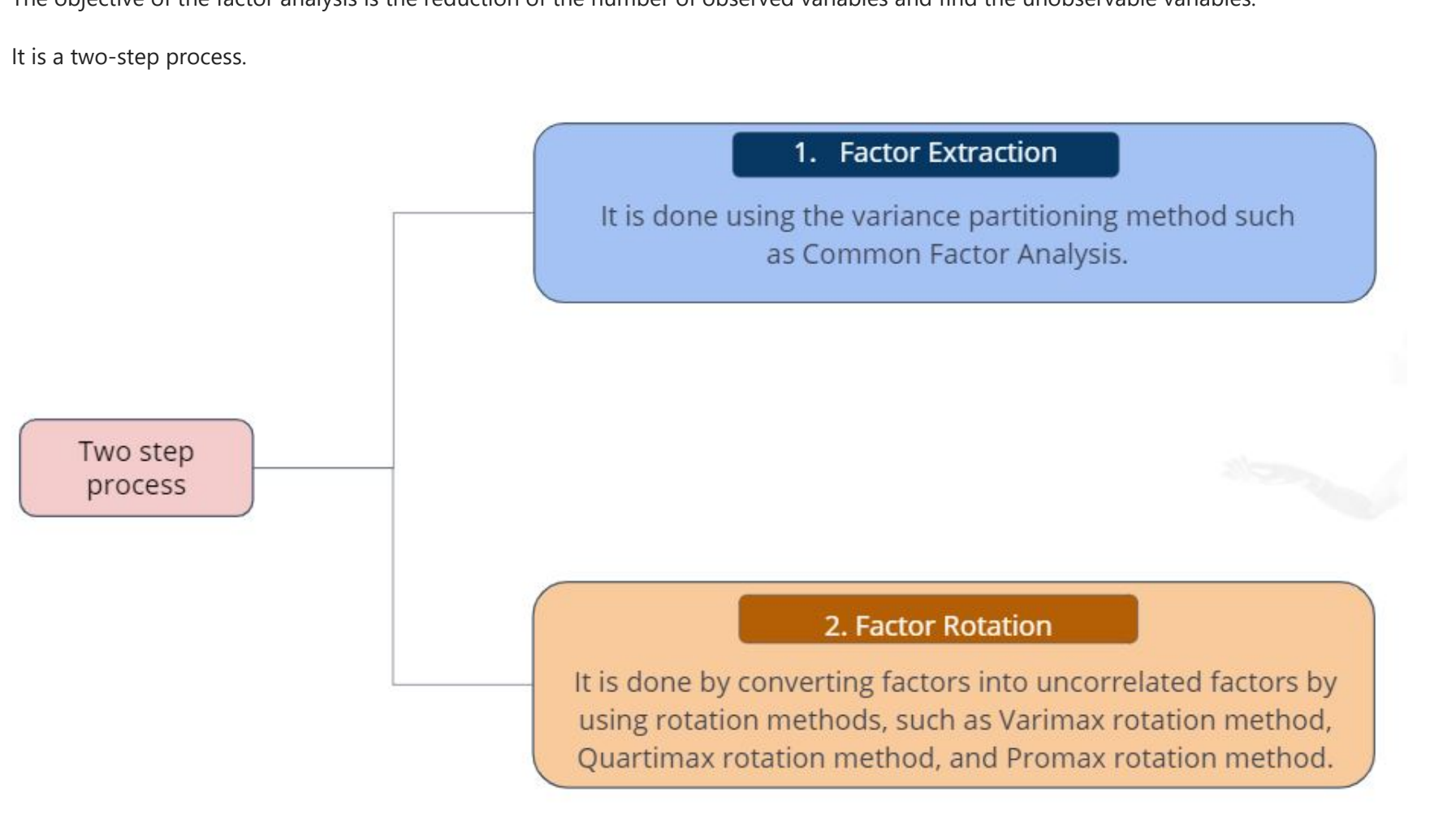
Types of FA



Work Process of FA

The objective of the factor analysis is the reduction of the number of observed variables and find the unobservable variables.

It is a two-step process.



Choosing Factors

Note: Let us get accustomed to the term eigenvalue before moving to the selecting the number of factors.

Eigenvalues:

It represents the explained variance of each factor from the total variance and is also known as the characteristic roots.

Ways to Choose Factors:

- The eigenvalues are a good measure for identifying the significant factors. An eigenvalue greater than 1 is considered for the selection criteria of the feature.
- Apart from observing values, the graphical approach is used that visually represents the factors' eigenvalues. This visualization is known as the scree plot. A Scree plot helps in determining the number of factors where the curve makes an elbow.

Use Case: Feature Selection in Cancer Dataset Using FA

Problem Statement

John Cancer Hospital (JCH) is a leading cancer hospital in the USA. It specializes in preventing breast cancer.

Over the last few years, JCH has collected breast cancer data from patients who came for screening or treatment.

However, this data has 32 attributes and is difficult to run and interpret the result. As an ML expert,

you have to reduce the number of attributes so that the results are meaningful and accurate.

Use FA for feature selection.

Dataset

Features of the dataset are computed from a digitized image of a Fine-Needle Aspirate (FNA) of a breast mass.

They describe the characteristics of the cell nuclei present in the image.

Data Dictionary

Dimensions:

- 32 variables
- 569 observations

Attribute Information:

1. ID number
2. Diagnosis (M = malignant, B = benign)
3. Attributes with mean values:
   10 real-valued features are computed for each cell nucleus:
   - radius_mean (mean of distances from center to points on the perimeter)
   - texture_mean (standard deviation of gray-scale values)
   - perimeter_mean
   - area_mean
   - smoothness_mean (local variation in radius lengths)
   - compactness_mean ($perimeter^2$ / area - 1.0)
   - concavity_mean (severity of concave portions of the contour)
   - concave points_mean (number of concave portions of the contour)
   - symmetry_mean
   - fractal dimension_mean ("coastline approximation" - 1)
4. Attributes with standard error and worst/largest:
   - radius_se
   - texture_se
   - perimeter_se
   - area_se
   - smoothness_se
   - compactness_se
   - concavity_se
   - concave points_se
   - symmetry_se
   - fractal_dimension_se
   - radius_worst
   - texture_worst
   - perimeter_worst
   - area_worst
   - smoothness_worst
   - compactness_worst
   - concavity_worst
   - concave points_worst
   - symmetry_worst
   - fractal_dimension_worst

Solution

Import Libraries

In Python, Numpy is a package that includes multidimensional array objects as well as a number of derived objects. Matplotlib is an amazing visualization library in Python for 2D plots of arrays.\n Pandas is used for data manipulation and analysis\n So these are the core libraries that are used for the EDA process.

These libraries are written with an import keyword.

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
%matplotlib inline
```

Import and Check the Data

Before reading data from a csv file, you need to download the "breast-cancer-data.csv" dataset from the resource section and upload it into the Lab. We will use the Up arrow icon, which is shown on the left side under the View icon. Click on the Up arrow icon and upload the file from wherever it has downloaded into your system.

After this, you will see the downloaded file will be visible on the left side of your lab along with all the .pynb files.

In [2]:
```
df = pd.read_csv('breast-cancer-data.csv')
df.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 32 columns

pd.read_csv function is used to read the "breast-cancer-data.csv" file and df.head() will show the top 5 rows of the dataset.

- dataframe or df is a variable that will store the data read by the csv file.
- head will show the rows and () default take the 5 top rows as output.
- one more example - df.head(3) will show the top 3 rows.

shape function

In [3]:
```
df.shape
```

Out[3]:
```
(569, 32)
```

df.shape will show the number of rows and columns in the dataframe.

info Function

In [4]:
```
# Check the data , there should be no missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

- The dataframe's information is printed using the info() function.
- The number of columns, column labels, column data types, memory use, range index, and the number of cells in each column are all included in the data (non-null values).

In [5]:
```
# defining the array as np.array
feature_names = np.array(['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 [6]:
```
#### Convert diagnosis column to 1/0 and store in new column target
from sklearn.preprocessing import LabelEncoder
```

- The sklearn.preprocessing package contains a number of useful utility methods and transformer classes for converting raw feature vectors into a format that is suitable for downstream estimators.
- LabelEncoder encodes labels with a value between 0 and n_classes-1 where n is the number of distinct labels.
- These libraries are written with an import keyword.

In [7]:
```
# # Encode label diagnosis
#M -> 1
#B -> 0

#Converting diagnosis to numerical variable in df
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0}).astype(int)
```

In the above code, we are encoding the column diagnosis in which we are encoding M as 1 and B as 0.

Factor Analysis

- A linear statistical model is a factor analysis.
- It is used to condense a group of observable variables into an unobserved variable termed factors and to explain the variance among the observed variables.

Adequacy Test

Before you perform factor analysis, you need to evaluate the "factorability" of our dataset. Can we find the factors in the dataset? Checking factorability or sampling adequacy can be done in two ways:

1- The Bartlett's Test

2- Test of Kaiser-Meyer-Olkin

In [8]:
```
#Install factor analyzer
#!pip install factor_analyzer
```

Bartlett's Test

Bartlett's test of sphericity checks whether or not the observed variables intercorrelate at all using the observed correlation matrix against the identity matrix. If the test is found to be statistically insignificant, you should not employ a factor analysis.

Note: This test checks for the intercorrelation of observed variables by comparing the observed correlation matrix against the identity matrix.

In [9]:
```
!pip install factor_analyzer
```

```
Requirement already satisfied: factor_analyzer in c:\users\alpika.gupta\anaconda3\lib\site-packages (0.4.0)
Requirement already satisfied: scikit-learn in c:\users\alpika.gupta\anaconda3\lib\site-packages (from factor_analyzer) (1.1.2)
Requirement already satisfied: scipy in c:\users\alpika.gupta\anaconda3\lib\site-packages (from factor_analyzer) (1.7.1)
Requirement already satisfied: numpy in c:\users\alpika.gupta\anaconda3\lib\site-packages (from factor_analyzer) (1.20.3)
Requirement already satisfied: pandas in c:\users\alpika.gupta\anaconda3\lib\site-packages (from factor_analyzer) (1.3.4)
Requirement already satisfied: pytz>=2017.3 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from pandas->factor_analyzer) (2021.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from pandas->factor_analyzer) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas->factor_analyzer) (1.16.0)
Requirement already satisfied: joblib>=1.0.0 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from scikit-learn->factor_analyzer) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\alpika.gupta\anaconda3\lib\site-packages (from scikit-learn->factor_analyzer) (2.2.0)
```

- In the above code, we are installing the factor analyzer.
- pip is used to install the packages.
- Factor analysis is an exploratory data analysis method used to search for influential underlying factors or latent variables from a set of observed variables.

Now, you are trying to perform factor analysis by using the factor analyzer module. Use the below code for calculating_bartlett_sphericity.

In [10]:
```
from factor_analyzer import FactorAnalyzer
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity
chi_square_value,p_value=calculate_bartlett_sphericity(df)
chi_square_value, p_value
```

Out[10]:
```
(40227.48570934462, 0.0)
```

- In the above code, you are importing the factor analyzer and calculate_bartlett_sphericity.
- In this Bartlett's test, the p-value is 0. The test was statistically significant, indicating that the observed correlation matrix is not an identity matrix.

Inference:

The p-value is 0, and this indicates that the test is statistically significant and highlights that the correlation matrix is not an identity matrix.

Kaiser-Meyer-Olkin Test

- The Kaiser-Meyer-Olkin (KMO) test determines if data is suitable for factor analysis.
- It assesses the suitability of each observed variable as well as the entire model.

In [11]:
```
from factor_analyzer.factor_analyzer import calculate_kmo
kmo_all,kmo_model=calculate_kmo(df)
```

```
C:\Users\alpika.gupta\Anaconda3\lib\site-packages\factor_analyzer\utils.py:249: UserWarning: The inverse of the variance-covariance matrix was calculated using the Moore-Penrose generalized matrix inversion, due to its determinant being at or very close to zero.
  warnings.warn('The inverse of the variance-covariance matrix '
```

In [12]:
```
kmo_model
```

Out[12]:
```
0.2599517032832366
```

- In the above code, we are calculating the KMO. KMO estimates the proportion of variance among all the observed variables.
- The overall KMO for the data is 0.25, which is excellent. This value indicates that you can proceed with your planned factor analysis.

In [13]:
```
df.head(2)
```

Out[13]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	1	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001
1	842517	1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869

2 rows × 32 columns

Inference:

The KMO value is less than 0.5, and this indicates that we need to delete the insignificant variables.

Finding Significant Variables

In [14]:
```
corr = df.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[14]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
id	1.000000	0.039769	0.074626	0.099770	0.073159	0.096893	-0.012968	0.000096
diagnosis	0.039769	1.000000	0.730029	0.415185	0.742636	0.708984	0.358560	0.596534
radius_mean	0.074626	0.730029	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124
texture_mean	0.099770	0.415185	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702
perimeter_mean	0.073159	0.742636	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936
area_mean	0.096893	0.708984	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502
smoothness_mean	-0.012968	0.358560	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123
compactness_mean	0.000096	0.596534	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000
concavity_mean	0.050080	0.696360	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121
concave points_mean	0.044158	0.776614	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135
symmetry_mean	-0.022114	0.330499	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641
fractal_dimension_mean	-0.052511	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369
radius_se	0.143048	0.567134	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473
texture_se	-0.007526	-0.008303	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205
perimeter_se	0.137331	0.556141	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905
area_se	0.177742	0.548236	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653
smoothness_se	0.096781	-0.067016	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299
compactness_se	0.033961	0.292999	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722
concavity_se	0.055239	0.253730	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517
concave points_se	0.078768	0.408042	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262
symmetry_se	-0.017306	-0.006522	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977
fractal_dimension_se	0.025725	0.077972	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318
radius_worst	0.082405	0.776454	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315
texture_worst	0.064720	0.456903	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133
perimeter_worst	0.079986	0.782914	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210
area_worst	0.107187	0.733825	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604
smoothness_worst	0.010338	0.421465	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541
compactness_worst	-0.002968	0.590998	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809
concavity_worst	0.023203	0.659610	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275
concave points_worst	0.035174	0.793566	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573
symmetry_worst	-0.044224	0.416294	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223
fractal_dimension_worst	-0.029866	0.323872	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382

If your main goal is to visualize the correlation matrix rather than creating a plot per se, the convenient pandas styling options are a viable built-in solution, as shown in the above code.

Creating Dataset of Significant Variables

In [16]:
```
df_corr = df[['radius_mean','perimeter_mean', 'area_mean','radius_worst','perimeter_worst',
             'area_worst','concavity_mean','concave points_mean', 'concavity_worst',
             'concave points_worst','diagnosis']]
```

Running KMO Test

In [17]:
```
from factor_analyzer.factor_analyzer import calculate_kmo
kmo_all,kmo_model=calculate_kmo(df_corr)
kmo_model
```

Out[17]:
```
0.8255727175653487
```

- In the above code, we are calculating the KMO. KMO estimates the proportion of variance among all the observed variables.
- The overall KMO for the data is 0.82, which is excellent. This value indicates that you can proceed with your planned factor analysis.

In [ ]:

Inference:

The value of the KMO model is greater than 0.5. Therefore, the dataset is good enough for factor analysis.

Factor Analysis

In [18]:
```
from factor_analyzer import FactorAnalyzer
fa = FactorAnalyzer(rotation='varimax')
fa.fit(df_corr,11)
```

Out[18]:
```
FactorAnalyzer
FactorAnalyzer(rotation='varimax', rotation_kwargs={})
```

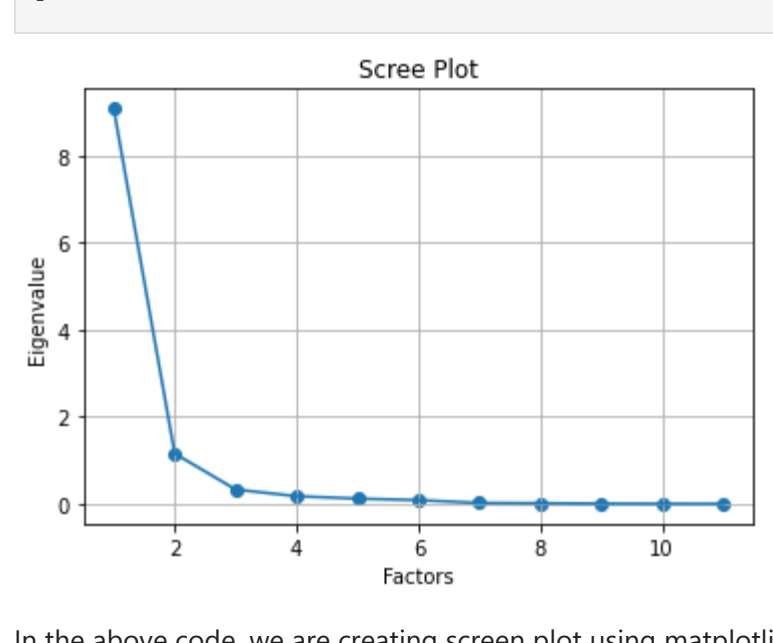
In the above code, we create a factor analysis object and perform factor analysis.

In [19]:
```
ev, v = fa.get_eigenvalues()
ev
```

Out[19]:
```
array([9.09178108e+00, 1.15744917e+00, 3.30050451e-01, 1.75328755e-01,
       1.20114615e-01, 8.46813558e-02, 2.04348144e-02, 1.33778025e-02,
       4.91643818e-03, 1.55811996e-03, 3.07397281e-04])
```

- Here, you can see only for 6-factors eigenvalues are greater than one. It means we only need to choose 6 factors (or unobserved variables).
- In the above code, we are checking the Eigenvalues.
- It measures how much of the variance of the variables a factor explains.
- An eigenvalue of more than one means that the factor explains more variance than a unique variable.

In [20]:
```
plt.scatter(range(1,df_corr.shape[1]+1),ev)
plt.plot(range(1,df_corr.shape[1]+1),ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()
```



In the above code, we are creating screen plot using matplotlib module.

Inference:

- Referring eigenvalues: There are only two values above 1. Therefore, we will choose only 2 factors (or unobserved variables).
- Referring Scree plot: There are only two values after the elbow. Therefore, we will choose only 2 factors (or unobserved variables).

In [21]:
```
fa1 = FactorAnalyzer(rotation="varimax", n_factors=2)
fa1.fit(df, 2)
```

Out[21]:
```
FactorAnalyzer
FactorAnalyzer(n_factors=2, rotation='varimax', rotation_kwargs={})
```

In [22]:
```
fa1.loadings_
```

Out[22]:
```
array([[ 0.09949822, -0.0175289 ],
       [ 0.76990049,  0.2623415 ],
       [ 0.97577565, -0.05180551],
       [ 0.37374525,  0.08592885],
       [ 0.97834607,  0.00385959],
       [ 0.97757834, -0.04299327],
       [ 0.21011043,  0.61739299],
       [ 0.54025265,  0.78340654],
       [ 0.71664403,  0.63252347],
       [ 0.84801319,  0.44148548],
       [ 0.19103234,  0.61468442],
       [-0.26532102,  0.86590234],
       [ 0.7431245 ,  0.18542383],
       [-0.04891222,  0.18458444],
       [ 0.7388625 ,  0.2275271 ],
       [ 0.79163254,  0.08879413],
       [-0.18452397,  0.38701833],
       [ 0.22456327,  0.78529353],
       [ 0.2181755 ,  0.66739594],
       [ 0.3904135 ,  0.59545102],
       [-0.08217263,  0.40098444],
       [-0.03298699,  0.73180422],
       [ 0.98993515, -0.00563542],
       [ 0.36256059,  0.11401422],
       [ 0.99041424,  0.05315552],
       [ 0.97481138, -0.01017728],
       [ 0.18911152,  0.56071661],
       [ 0.46178052,  0.69523831],
       [ 0.5755513 ,  0.64392357],
       [ 0.78203771,  0.47681393],
       [ 0.20855172,  0.49045164],
       [ 0.06575214,  0.79707143]])
```

Confirmatory Factor Analysis

- It determines the factors and factor loadings of measured variables.
- The general assumption of CFA is that every factor has an association with a specific subset of the measured variables.

A ConfirmatoryFactorAnalyzer class, which fits a confirmatory factor analysis model using maximum likelihood.

Parameters:

- specification (ModelSpecificaition object or None, optional): It is a model specification that must be a ModelSpecificaiton object or None. If None, the ModelSpecification will be generated assuming that n_factors == n_variables, and that all variables load on all factors.

  Note that this could mean the factor model is not identified, and the optimization could fail. Hence, the factor value will be taken as none by default.

- n_obs (int or None, optional): It indicates the number of observations in the original data set. If this is not passed and is_cov_matrix=True, then an error will be raised. Hence, the value of n_obs will be default to None.

- is_cov_matrix (bool, optional): It denotes whether the input X is a covariance matrix. If False, assume it is the full data set. Defaults to False.

- bounds (list of tuples or None, optional): It is a list of minimum and maximum boundaries for each element of the input array. This must equal x0, which is the input array from your parsed and combined model specification. The length is:

((n_factors *n_variables) + n_variables + n_factors + (((n_factors* n_factors) - n_factors) // 2)

If None, nothing will be bounded. Defaults to None.

- max_iter (int, optional): It is the maximum number of iterations for the optimization routine. Defaults to 200.
- tol (float or None, optional): It is the tolerance for convergence. Defaults to None.
- disp (bool, optional): It denotes whether to print the scipy optimization fmin message to standard output. Defaults to True.

Raises: ValueError – If is_cov_matrix is True, and n_obs is not provided.

model

```
The model specification object.
Type: ModelSpecification
```

loadings_

```
The factor loadings matrix.
Type: numpy array
```

error*vars*

```
The error variance matrix
Type: numpy array
```

factor*varcovs*

```
The factor covariance matrix.
Type: numpy array
```

log*likelihood*

```
The log likelihood from the optimization routine.
Type: float
```

In [25]:
```
#  CFA stands for Confirmatory Factor Analysis.This line imports CFA function
from factor_analyzer import (ConfirmatoryFactorAnalyzer, ModelSpecificationParser)
from factor_analyzer import (ConfirmatoryFactorAnalyzer, ModelSpecificationParser)
```

Here, 'radius_mean', 'perimeter_mean', 'area_mean', 'radius_worst', 'perimeter_worst', 'diagnosis', 'area_worst', 'concavity_mean', 'concave points_mean', 'concavity_worst', 'concave points_worst' refers to the name of columns in the data frame that you'd like to allocate to each factor (F1 and F2).

In [27]:
```
model_dict = {"F1": ['radius_mean','perimeter_mean', 'area_mean','radius_worst',
                     'perimeter_worst','diagnosis'], "F2": ['area_worst','concavity_mean',
                    'concave points_mean', 'concavity_worst','concave points_worst']}
```

class ModelSpecificationParser:

```
A class to generate the model specification for CFA.
This class includes two static methods to generate the
"ModelSpecification" object from either a dictionary
or a numpy array.
```

def parse_model_specification_from_dict(X, specification=None):

```
    Generate the model specification from a
    dictionary. The keys in the dictionary
    should be the factor names, and the values
    should be the feature names. If this method
    is used to create the "ModelSpecification",
    then factor names and variable names will
    be added as properties to that object.
```

Parameters

```
    ----------
    X : array-like
        The data set that will be used for CFA.
    specification : dict or None
        A dictionary with the loading details. If None, the matrix will
        be created assuming all variables load on all factors.
        Defaults to None.

    Returns
    -------
    ModelSpecification: A model specification object

    Raises
    ------
    ValueError: If 'specification' is not in the expected format.
```

In [28]:
```python
model_spec = ModelSpecificationParser.parse_model_specification_from_dict(df_corr,model_dict)
# df_corr:Pandas DataFrame and model_dict: dictionary with factors
```

```
def fit(self, X, y=None):
    """
    Perform confirmatory factor analysis.

    Parameters
    ----------
    X : array-like
        The data to use for confirmatory
        factor analysis. If this is just a
        covariance matrix, make sure 'is_cov_matrix'
        was set to True.
    y : ignored

    Raises
    ------
    ValueError:If the specification is not None or a "ModelSpecification" object
    AssertionError:If "is_cov_matrix=True" and the matrix is not square.
    AssertionError:If len(bounds) != len(x0)
    """
```

In [29]:
```python
# Performs confirmatory factor analysis
cfa1 = ConfirmatoryFactorAnalyzer(model_spec, disp=False)
cfa1.fit(df_corr.values)
```

```
C:\Users\alpika.gupta\Anaconda3\lib\site-packages\factor_analyzer\confirmatory_factor_analyzer.py:732: UserWarn
ing: The optimization routine failed to converge: ABNORMAL_TERMINATION_IN_LNSRCH
  warnings.warn('The optimization routine failed '
```

Out[29]:
```
ConfirmatoryFactorAnalyzer

ConfirmatoryFactorAnalyzer(disp=False, n_obs=569,
                           specification=<factor_analyzer.confirmatory_factor_analyzer.ModelSpecification obj
ect at 0x0000016DF76C4A60>)
```

In [30]:
```python
# cfa1.loadings_ will gave you the factor loading matrix
# The factor loading is a matrix which shows the relationship of each variable to the underlying factor.
# It shows the correlation coefficient for observed variable and factor.
# It shows the variance explained by the observed variables.
cfa1.loadings_
```

Out[30]:
```
array([[ 9.52388865e+00,  0.00000000e+00],
       [ 3.47315903e+02,  0.00000000e+00],
       [ 7.86489967e+03,  0.00000000e+00],
       [-1.49873855e+02,  0.00000000e+00],
       [ 9.11336000e+02,  0.00000000e+00],
       [ 1.32123059e+04,  0.00000000e+00],
       [ 0.00000000e+00, -1.98153073e+03],
       [ 0.00000000e+00, -1.45499332e+03],
       [ 0.00000000e+00, -3.13842115e+02],
       [ 0.00000000e+00,  4.12173570e+03],
       [ 0.00000000e+00,  1.11725801e+02]])
```

In [31]:
```python
#This will give you the factor covariance matrix and the type of this will be numpy array
cfa1.factor_varcovs_
```

Out[31]:
```
array([[1.        , 0.33443073],
       [0.33443073, 1.        ]])
```

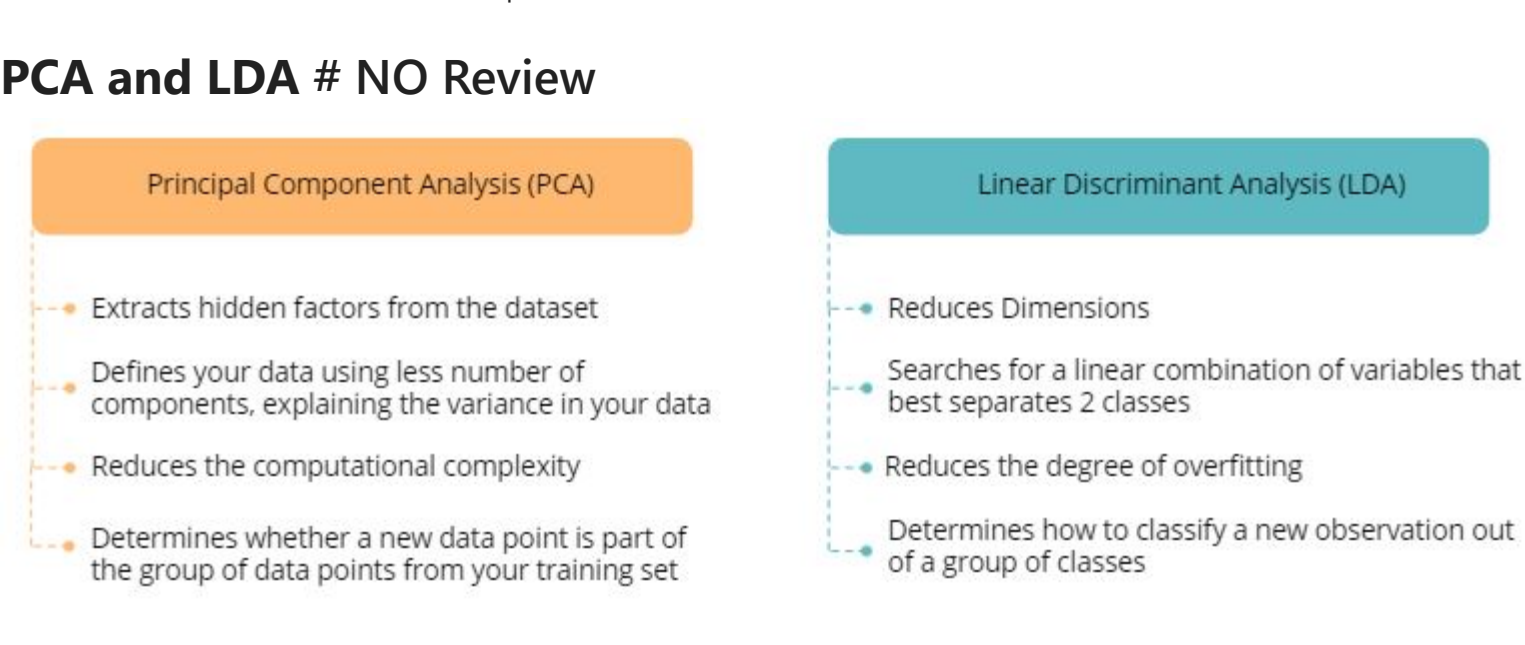
In [32]:
```python
# transform(X) used to get the factor scores for new data set.

# Parameters:X (array-like, shape (n_samples, n_features)) - The data to score using the fitted factor model.
# Returns: scores - The latent variables of X.
# Return type:numpy array, shape (n_samples, n_components)
```

In [33]:
```python
cfa1.transform(df_corr.values)
```

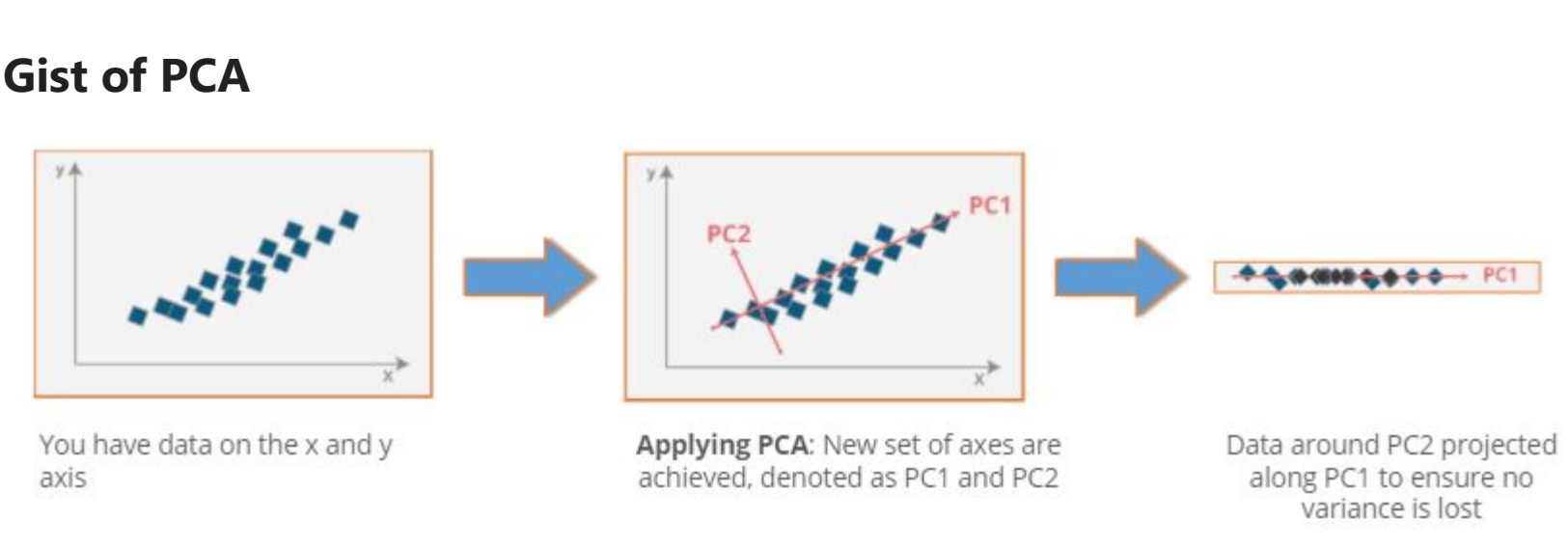
Out[33]:
```
array([[ 0.08030754,  0.00103054],
       [ 0.08199898,  0.00108067],
       [ 0.06382796,  0.0010036 ],
       ...,
       [ 0.01961916,  0.0012    ],
       [ 0.0725346 ,  0.00114118],
       [-0.04865471, -0.00067256]])
```

Now, let us take a look at PCA and LDA-specific benefits.

PCA and LDA # NO Review



Gist of PCA



What Is Principal Component?

A principal component is a normalized linear combination of the original predictors in a dataset.

First Principal Component (Z^1)

- The aim of PCA is to find components that account for **maximum variance in the data** that includes the error and within-variable variance.
- It finds the direction of the highest variability in the data. Greater the variability captured in the first component, the greater the information captured by the component.
- The first principal component develops a line that is closest to the data points.

  In other words, it minimizes the sum of squared distance between a data point and the line.

Equation for first principal component:

$Z^1 = \Phi^{11}X^1 + \Phi^{21}X^2 + \Phi^{31}X^3 + ... + \Phi p^1 Xp$

- Z^1 = First principal component
- Φp^1 = Loading vector
- X^1 to Xp = Normalized predictors that have mean equals to **0** and standard deviation equals to **1**.

Second Principal Component (Z^2)

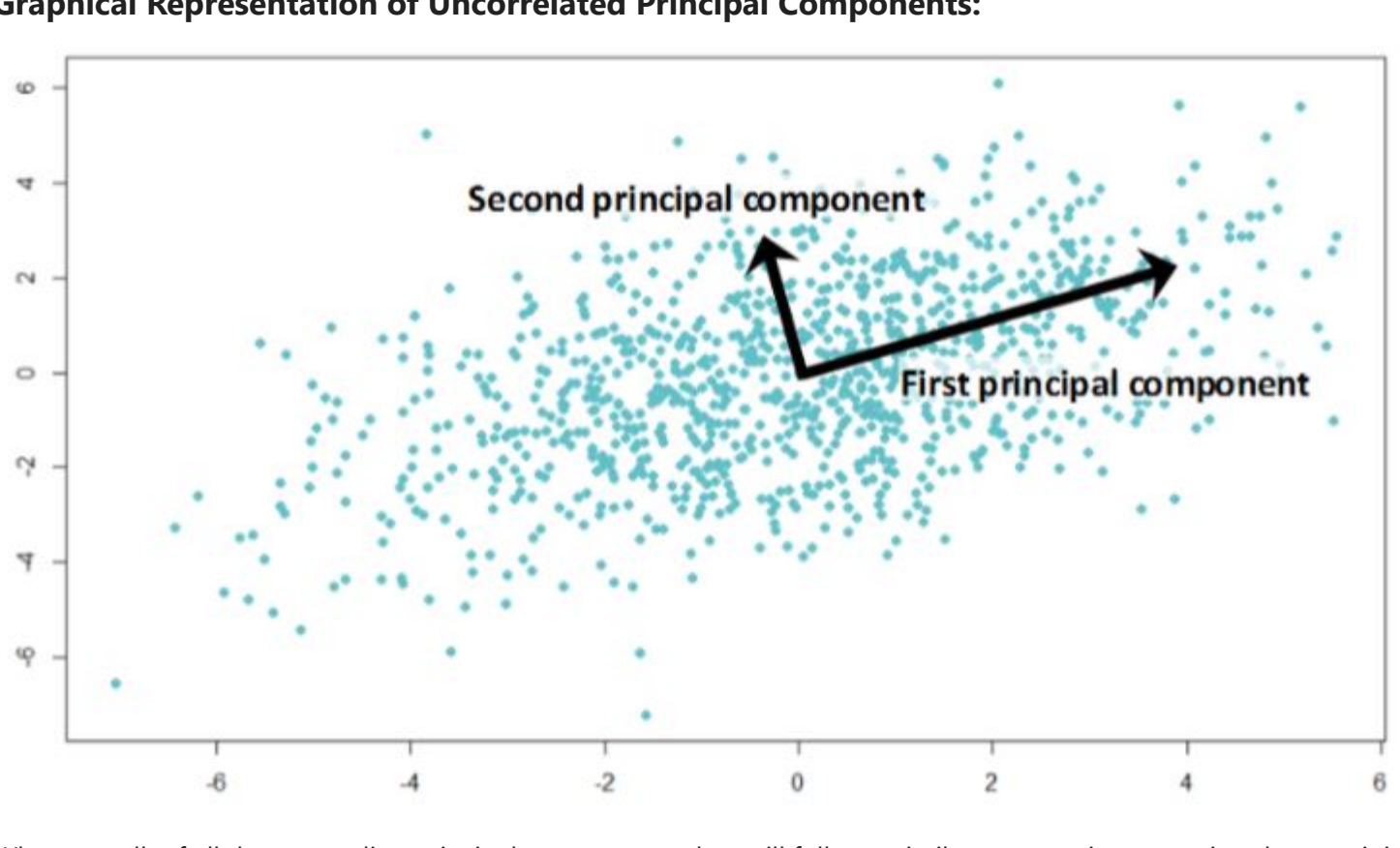
The aim of the linear combination of original predictors is to capture the remaining variance in the dataset where Z^2 is uncorrelated to Z^1.

In other words, the correlation between the first and second components should be zero.

Equation for first principal component:

$Z^2 = \Phi^{12}X^1 + \Phi^{22}X^2 + \Phi^{32}X^3 + ... + \Phi p2Xp$

Graphical Representation of Uncorrelated Principal Components:



When we talk of all the succeeding principal components, they will follow a similar concept, i.e., capturing the remaining variation without being correlated to the previous component.

How Does PCA work?

PCA is built on the concept of Eigen vectors and Eigen values. So, let us first understand the Eigen vector and value in brief.

Eigenvectors and Eigenvalues

Eigenvectors

When the vectors are plotted on a two-dimensional plane they showcase both the magnitude and direction.

Usually, the direction of a vector is along its span but there are special vectors that on linear transformation get squished or stretched instead of falling off their span.

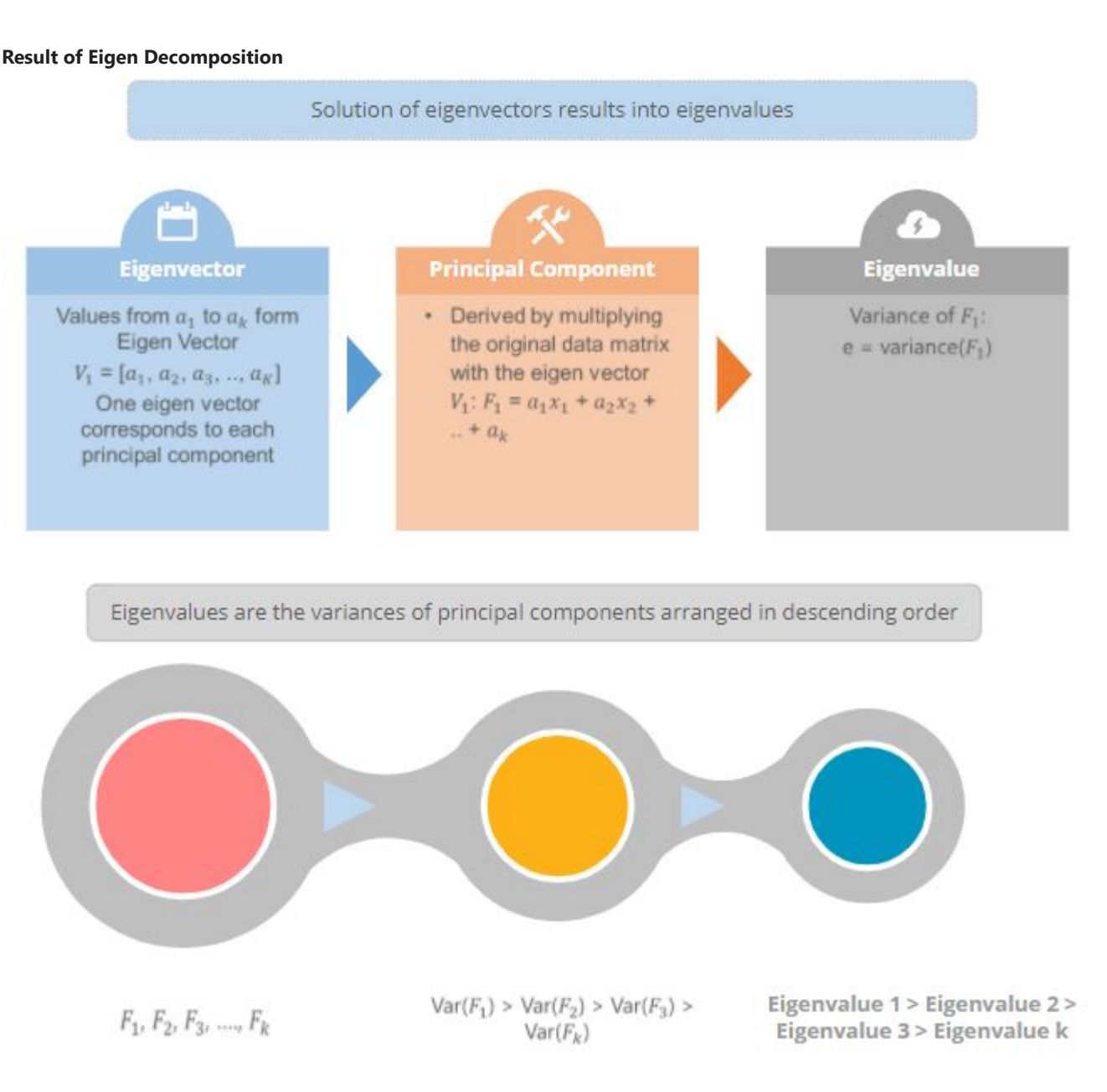
Eigenvalues

These are the constant values that increase or decrease the Eigenvectors along their span when transformed linearly.

PCA Process

Finding PC1

Result of Eigen Decomposition



Use Case: Feature Selection in Cancer Dataset Using PCA

Problem Statement

John Cancer Hospital (JCH) is a leading cancer hospital in the USA. It specializes in preventing breast cancer.

Over the last few years, JCH has collected breast cancer data from patients who came for screening or treatment.

However, this data has 32 attributes and is difficult to run and interpret the result. As an ML expert,

you have to reduce the number of attributes so that the results are meaningful and accurate.

Use PCA for feature selection.

Dataset

Features of the dataset are computed from a digitized image of a Fine-Needle Aspirate (FNA) of a breast mass.

They describe the characteristics of the cell nuclei present in the image.

Links to the Dataset

This database is available at:

- The UCI Machine Learning Repository:
  https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

Data Dictionary

Dimensions:

- 32 variables
- 569 observations

Attribute Information:

1. ID number
2. Diagnosis (M = malignant, B = benign)
3. Attributes with mean values:
   Ten real-valued features are computed for each cell nucleus:
   - radius_mean (mean of distances from center to points on the perimeter)
   - texture_mean (standard deviation of gray-scale values)
   - perimeter_mean
   - area_mean
   - smoothness_mean (local variation in radius lengths)
   - compactness_mean (perimeter^2 / area - 1.0)
   - concavity_mean (severity of concave portions of the contour)
   - concave points_mean (number of concave portions of the contour)
   - symmetry_mean
   - fractal dimension_mean ("coastline approximation" - 1)
4. Attributes with standard error and worst/largest:
   - radius_se
   - texture_se
   - perimeter_se
   - area_se
   - smoothness_se
   - compactness_se
   - concavity_se
   - concave points_se
   - symmetry_se
   - fractal_dimension_se
   - radius_worst
   - texture_worst
   - perimeter_worst
   - area_worst
   - smoothness_worst
   - compactness_worst
   - concavity_worst
   - concave points_worst
   - symmetry_worst
   - fractal_dimension_worst

Solution

Import Libraries

> Indented block # CE Review

In [34]:
```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

Import and Check the Data

Before reading data from a csv file, you need to download the "breast-cancer-data.csv" dataset from the resource section and upload it into the lab.
We will use the Up arrow icon which is shown on the left side under the View icon. Click on the Up arrow icon and upload the filewherever it has been downloaded into your system.

After this, you will see the downloaded file will be visible on the left side of your lab along with all the .pynb files.

In [35]:
```python
df = pd.read_csv('breast-cancer-data.csv')
df.head()
```

Out[35]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 32 columns

pd.read_csv function is used to read the "breast-cancer-data.csv" file and df.head() will show the top 5 rows of the dataset.

- dataframe or df is a variable that will store the data read by the csv file.
- head will show the rows and () default take the top 5 rows as output.

In [36]:
```python
df.shape
```

Out[36]:
```
(569, 32)
```

In [37]:
```python
# Check the data , there should be no missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

- The dataframe's information is printed using the info() function.
- The number of columns, column labels, column data types, memory use, range index, and the number of cells in each column are all included in the data (non-null values).

In [38]:
```python
feature_names = np.array(['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 [39]:
```python
#### Convert diagnosis column to 1/0 and store in new column target
from sklearn.preprocessing import LabelEncoder
```

In [40]:
```python
# # Encode label diagnosis
#M -> 1
#B -> 0
```

In [41]:
```python
target_data=df["diagnosis"]
encoder = LabelEncoder()
target_data = encoder.fit_transform(target_data)
```

In the above code, in output we are getting all the rows, but only with the last column.

In [42]:
```python
df.drop(["diagnosis"],axis = 1, inplace = True)
```

In the above code, you store the encoded column in a dataframe and drop the diagnosis column for simpilcity.

Principal Component Analysis

Let's use PCA to find the first two principal components and visualize the data in this new, two-dimensional space with a single scatter-plot.

In [43]:
```python
from sklearn.preprocessing import StandardScaler
```

In the above code, you will scale data so that each feature has a single unit variance.

In [44]:
```python
scaler = StandardScaler()
scaler.fit(df)
```

Out[44]:
```
StandardScaler
StandardScaler()
```

In [45]:
```python
scaled_data = scaler.transform(df)
```

In the above code, you will initially create an object of the StandardScaler() function. Further, you will use fit() along with the object assigned to df and standardize it.

Two Principal Components

In [46]:
```python
from sklearn.decomposition import PCA
```

In the above code, you can transform this data into its first 2 principal components.

In [47]:
```python
pca = PCA(n_components=2)
```

Now, you will pass the number of components (n_components=2).

In [48]:
```python
pca.fit(scaled_data)
```

Out[48]:
```
PCA
PCA(n_components=2)
```

In [49]:
```python
x_pca = pca.transform(scaled_data)
```

Finally, call fit function to aggregate the data. Here, several components represent the lower dimension in which you will project your higher dimension data.

In [50]:
```python
#shape of data
scaled_data.shape
```

Out[50]:
```
(569, 31)
```

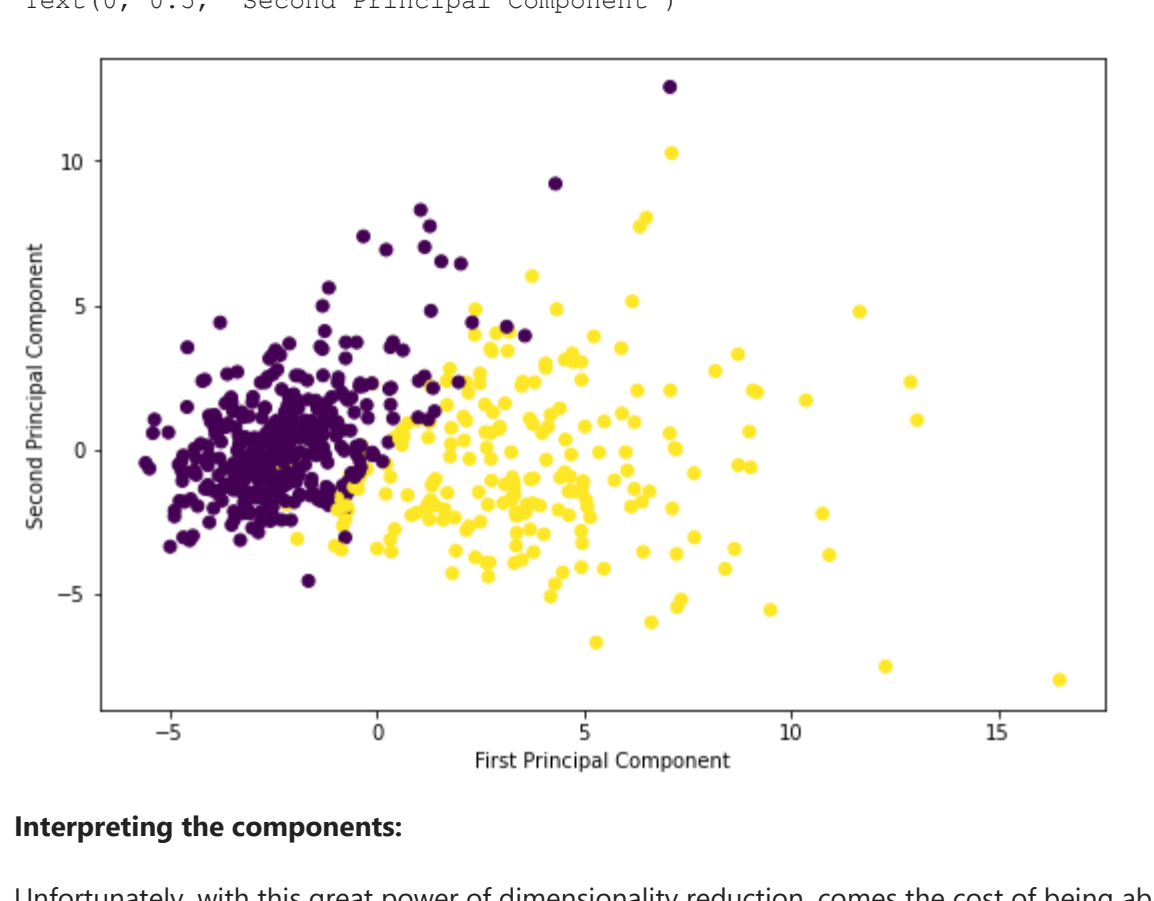
In [51]:
```python
x_pca.shape
```

Out[51]:
```
(569, 2)
```

Plot # NO Review

In [52]:
```python
# Reduced 30 dimensions to just 2! Let's plot these two dimensions out!
# Draw inference from the plot?
plt.figure(figsize=(9,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=target_data,cmap='viridis')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
```

Out[52]:
```
Text(0, 0.5, 'Second Principal Component')
```



Interpreting the components:

Unfortunately, with this great power of dimensionality reduction, comes the cost of being able to easily understand what these components represent.

The components correspond to combinations of the original features. The components themselves are stored as an attribute of the fitted PCA object:

In [53]:
```python
pca.components_
```

Out[53]:
```
array([[ 0.02291216,  0.21891302,  0.10384388,  0.22753491,  0.22104577,
         0.14241471,  0.2390673 ,  0.25828025,  0.26073811,  0.13797774,
         0.06414779,  0.20611747,  0.01741339,  0.21144652,  0.20307642,
         0.01467821,  0.1702884 ,  0.15354367,  0.18340675,  0.04241552,
         0.10249607,  0.22800935,  0.10451545,  0.23663734,  0.22493214,
         0.12782441,  0.20988456,  0.22860218,  0.2507462 ,  0.12267993,
         0.13156024],
       [-0.03406849, -0.2332714 , -0.0600442 , -0.214589  , -0.23066882,
         0.18642221,  0.15245473,  0.06054163, -0.03416739,  0.19068498,
         0.36653106, -0.1059357 ,  0.08954779, -0.08980704, -0.15277129,
         0.20318988,  0.23250336,  0.19684608,  0.12996518,  0.18355863,
         0.27958414, -0.21929604, -0.04550122, -0.19929599, -0.21898546,
         0.17256296,  0.14425364,  0.09852652, -0.00753437,  0.14261944,
         0.27570208]])
```

Explained Variance:

The explained variance tells you how much information (variance) can be attributed to each of the principal components. This is important as you can convert n-dimensional space to 2-dimensional space; you lose some of the variance (information).

In [54]:
```python
pca.explained_variance_ratio_
```

Out[54]:
```
array([0.42864701, 0.18376792])
```

Three Principal Components

In [55]:
```python
pca_3 = PCA(n_components=3)
pca_3.fit(scaled_data)
x_pca_3 = pca_3.transform(scaled_data)
```

In this Numpy matrix array, each row represents a principal component, and each column relates back to the original features. You can visualize this relationship with a heatmap:

In [56]:
```python
x_pca_3.shape
```

Out[56]:
```
(569, 3)
```

What is the total variance attributed by three Components?

In [57]:
```python
pca_3.explained_variance_ratio_
```

Out[57]:
```
array([0.42864701, 0.18376792, 0.09146436])
```

Gist of LDA

LDA identifies the linear combination of the observed variables that maximize the class separation on the availability of the prior information of the classes.

For example:

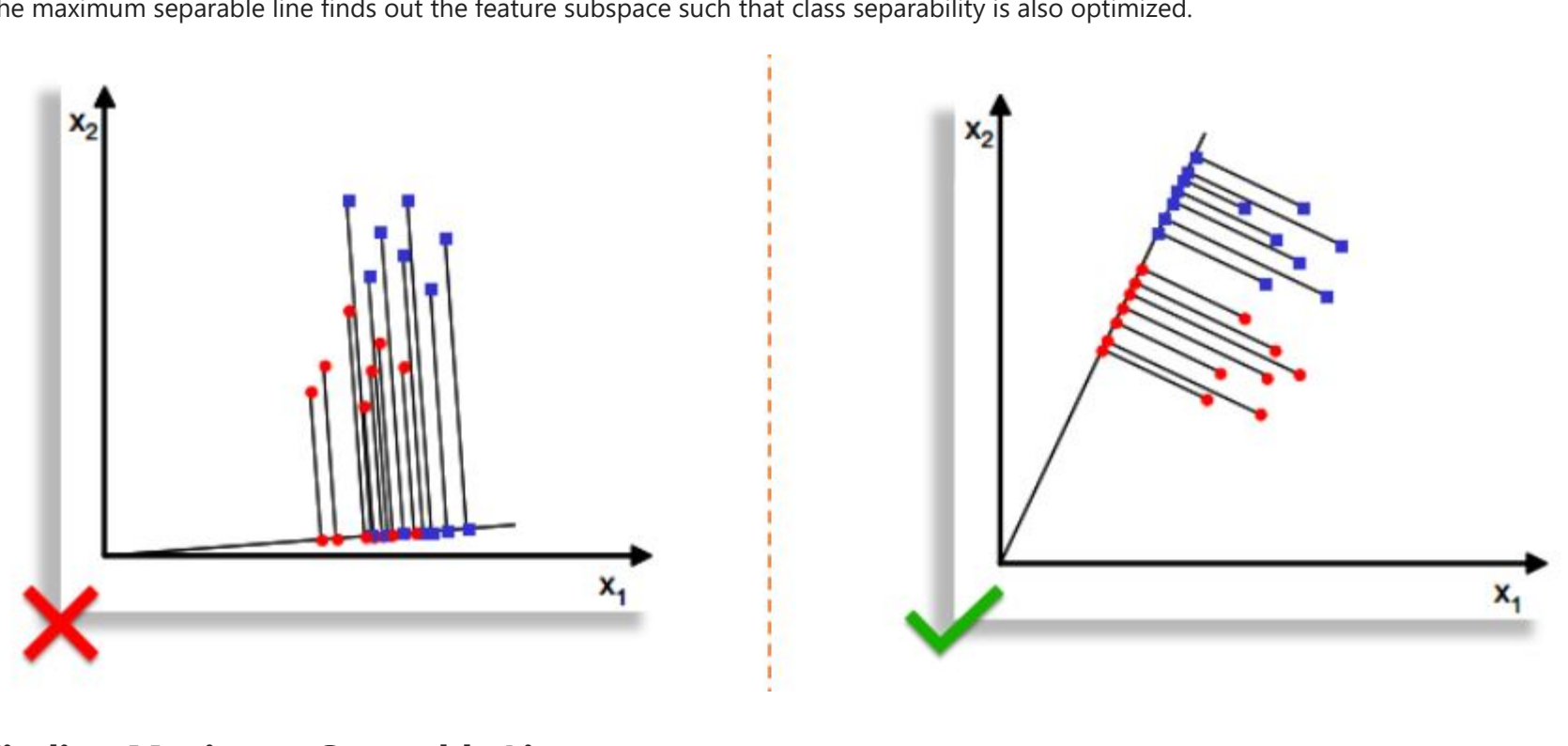
A variable in the training dataset that specifies the class of each observation

LDA Process

- Assume a set of D - dimensional samples (X(1, X(2, ..., X(N), N1 of which belong to class ω1 and N2 to class ω2
- Obtain a scalar **y** by projecting the samples **x** onto a line: **Y = W^T X**
- Of all the possible lines, select the one that maximizes the separability of the scalars

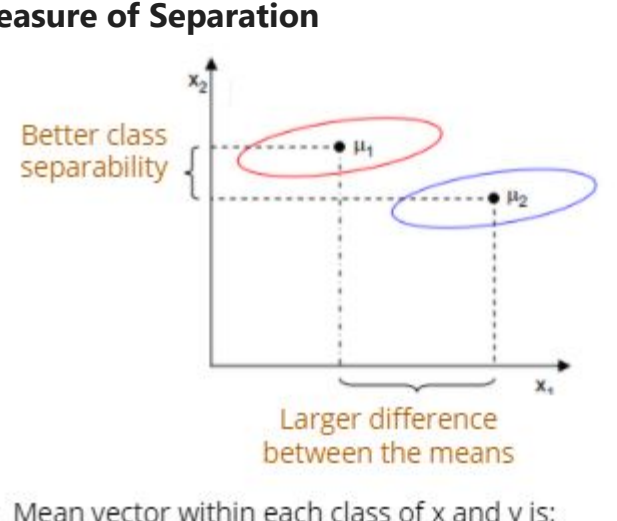
Line of Maximum Separability

The maximum separable line finds out the feature subspace such that class separability is also optimized.



Finding Maximum Separable Line

1. Measure of Separation



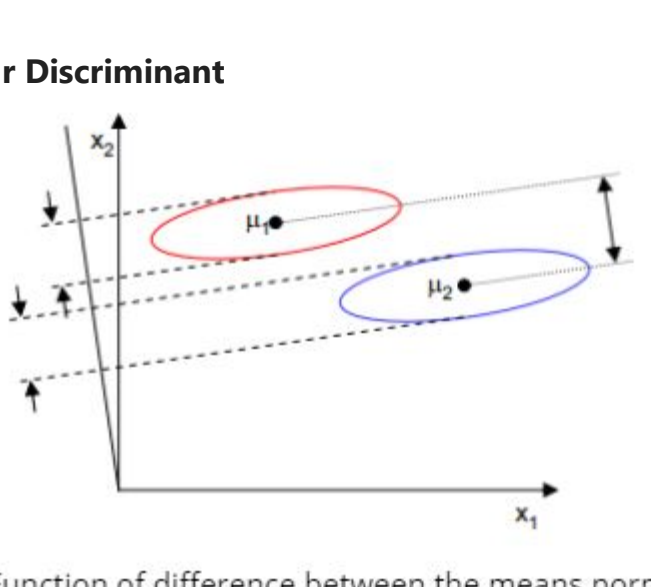
- Mean vector within each class of x and y is:

$$\mu_i = \frac{1}{N_i}\sum_{x \in \omega_i} x \quad \text{and} \quad \tilde{\mu}_i = \frac{1}{N_i}\sum_{y \in \omega_i} y = \frac{1}{N_i}\sum_{x \in \omega_i} w^T x = w^T \mu_i$$

- Objective function is the distance between the projected means:

$$J(w) = |\tilde{\mu}_1 - \tilde{\mu}_2| = |w^T(\mu_1 - \mu_2)|$$

2. Linear Discriminant



- Function of difference between the means normalized by measure of scatter (an equivalent variance):

$$J(w) = \frac{|\tilde{\mu}_1 - \tilde{\mu}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

Variables from same class are projected very close to each other and at the same time, the projected means are as farther apart as possible

Note: Variables from same class are projected very close to each other and at the same time, the projected means are as farther apart as possible

3. Optimum Projection

- A measure of the scatter in multivariate feature space x:

$$S_i = \sum_{x \in \omega_i}(x - \mu_i)(x - \mu_i)^T$$

$$S_1 + S_2 = S_W$$

S_W is the within class scatter matrix

- Scatter of the projection y can be expressed as a function of the scatter matrix in feature space x

$$\tilde{s}_i^2 = \sum_{y \in \omega_i}(y - \tilde{\mu}_i)^2 = \sum_{x \in \omega_i}(w^T x - w^T \mu_i)^2 = \sum_{x \in \omega_i} w^T (x - \mu_i)(x - \mu_i)^T w = w^T S_i w$$

$$\tilde{s}_1^2 + \tilde{s}_2^2 = w^T S_W w$$

S_W is the within class scatter

- Difference between the projected means

$$(\tilde{\mu}_1 - \tilde{\mu}_2)^2 = (w^T\mu_1 - w^T\mu_2)^2 = w^T \underbrace{(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T}_{S_B} w = w^T S_B w$$

4. Obtain the Maxima

- Express the linear discriminant in terms of S_W and S_B:

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

- Find the maxima of J(w) by differentiating and equating to zero

$$w^* = \underset{w}{\operatorname{argmax}}\left\{\frac{w^T S_B w}{w^T S_W w}\right\} = S_W^{-1}(\mu_1 - \mu_2)$$

Above is the optimal direction for projection

Let us perform LDA using the same dataset that we used for PCA.

Use Case: Feature Selection in Cancer Dataset Using LDA

Note: We are leveraging the problem statement, dataset, and data dictionary from the previous use case.

Problem Statement

John Cancer Hospital (JCH) is a leading cancer hospital in the USA. It specializes in preventing breast cancer.

Over the last few years, JCH has collected breast cancer data from patients who came for screening or treatment.

However, this data has 32 attributes and is difficult to run and interpret the result. As an ML expert,

you have to reduce the number of attributes so that the results are meaningful and accurate.

Use LDA for feature selection.

Dataset

Features of the dataset are computed from a digitized image of a Fine-Needle Aspirate (FNA) of a breast mass.

They describe the characteristics of the cell nuclei present in the image.

Data Dictionary

Dimensions:

- 32 variables
- 569 observations

Attribute Information:

1. ID number
2. Diagnosis (M = malignant, B = benign)
3. Attributes with mean values:
   Ten real-valued features are computed for each cell nucleus:
   - radius_mean (mean of distances from center to points on the perimeter)
   - texture_mean (standard deviation of gray-scale values)
   - perimeter_mean
   - area_mean
   - smoothness_mean (local variation in radius lengths)
   - compactness_mean (perimeter^2 / area - 1.0)
   - concavity_mean (severity of concave portions of the contour)
   - concave points_mean (number of concave portions of the contour)
   - symmetry_mean
   - fractal dimension_mean ("coastline approximation" - 1)
4. Attributes with standard error and worst/largest:
   - radius_se
   - texture_se
   - perimeter_se
   - area_se
   - smoothness_se
   - compactness_se
   - concavity_se
   - concave points_se
   - symmetry_se
   - fractal_dimension_se
   - radius_worst
   - texture_worst
   - perimeter_worst
   - area_worst
   - smoothness_worst
   - compactness_worst
   - concavity_worst
   - concave points_worst
   - symmetry_worst
   - fractal_dimension_worst

Solution

Import and Check the Data # CE Review

In [58]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
```

In the above code, you are importing different files: pandas, numpy, pyplot, LinearDiscriminantAnalysis, KFold, cross_val_score, GridSearchCV, train_test_split,accuracy_score, classification_report, confusion_matrix and sea to know them better, you can start the notebook from the beginning.

In [59]:
```
# load data

df = pd.read_csv('breast-cancer-data.csv')
```

Before reading data from a csv file, you need to download the "breast-cancer-data.csv" dataset from the resource section and upload it into the lab. We will use the Up arrow icon, which is shown on the left side under the View icon. Click on the Up arrow icon and upload the filewhereever it has been downloaded into your system.

After this, you will see the downloaded file will be visible on the left side of your lab with all the .pynb files.

In [60]:
```
df = df.head(559)
```

In [61]:
```
df.shape
```

Out[61]:
```
(559, 32)
```

df.shape will show the number of rows and columns in the dataframe.

In [62]:
```
# map categorical variable 'diagnosis' into numeric

df.diagnosis = df.diagnosis.map({'M': 1, 'B': 0})
```

In the above code, you are encoding the column diagnosis in which we are encoding M as 1 and B as 0.

In [63]:
```
# head will show the rows and () default take 5 top rows as output.
df.head(5)
```

Out[63]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	c point
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	

5 rows × 32 columns

In [64]:
```
df.columns
```

Out[64]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [65]:
```
# Drop redundant column 'id'

df.drop('id', axis=1, inplace=True)
```

In [66]:
```
# Check for NA values

df.isna().any()
```

Out[66]:
```
diagnosis                  False
radius_mean                False
texture_mean               False
perimeter_mean             False
area_mean                  False
smoothness_mean            False
compactness_mean           False
concavity_mean             False
concave points_mean        False
symmetry_mean              False
fractal_dimension_mean     False
radius_se                  False
texture_se                 False
perimeter_se               False
area_se                    False
smoothness_se              False
compactness_se             False
concavity_se               False
concave points_se          False
symmetry_se                False
fractal_dimension_se       False
radius_worst               False
texture_worst              False
perimeter_worst            False
area_worst                 False
smoothness_worst           False
compactness_worst          False
concavity_worst            False
concave points_worst       False
symmetry_worst             False
fractal_dimension_worst    False
dtype: bool
```

In [68]:
```
X_train, X_val, y_train, y_val = train_test_split(df.iloc[:,:-1], df['diagnosis'],
                                  train_size=0.8, test_size=0.2, random_state=120)
```

In the above code, you split the data into training and validation sets (since we already have a separate test set).

In [69]:
```
from sklearn.preprocessing import Normalizer

norm = Normalizer()
norm.fit(X_train)
X_train_norm = norm.transform(X_train)
X_val_norm = norm.transform(X_val)
```

In above code, you are Normalizing the features.

LDA

Let's throw in linear classifiers, i.e., Linear Discriminant Analysis for good measure and because our dataset is small.

In [70]:
```
lda = LinearDiscriminantAnalysis()
lda.fit(X_train_norm, y_train)
lda_predicted = lda.predict(X_val_norm)

print('LDA Accuracy is: {}'.format(accuracy_score(y_val,lda_predicted)))

print('LDA Classification Report')
print(classification_report(y_val, lda_predicted))
```

```
LDA Accuracy is: 1.0
LDA Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        76
           1       1.00      1.00      1.00        36

    accuracy                           1.00       112
   macro avg       1.00      1.00      1.00       112
weighted avg       1.00      1.00      1.00       112
```

- In the above code, you are doing the Linear Discriminant Analysis, in which you find the LDA Accuracy and LDA Classification Report.
- Analogous to the principle of beam search, let us stick with the three best algorithms, LDA, RFC, and GBC, and tune these models further, p the one that gives the highest accuracy after tuning. Instantiate a new LDA model and check its performance.

In [71]:
```
confusion_matrix_lda = pd.DataFrame(confusion_matrix(y_val, lda_predicted),
                                    index = ['Actual Negative','Actual Positive'],
                          columns = ['Predicted Negative','Predicted Postive'] )
confusion_matrix_lda
```

Out[71]:

	Predicted Negative	Predicted Postive
Actual Negative	76	0
Actual Positive	0	36