Dimensionality Reduction

- PCA(Principal Component Analysis)

Dimensionality Reduction

- Dimensions are represented as features or columns in dataset.
- Reduction of number of features in our datset
  1. Feature Elimination
  2. Feature Extraction

In [1]:
```
# import the required packages
import pandas as pd
```

In [3]:
```
from sklearn.datasets import load_breast_cancer
```

In [4]:
```
cancer = load_breast_cancer()
cancer
```

Out[4]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([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, 0, 1, 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, 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, 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, 0, 0, 1, 0, 1, 0, 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, 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, 1, 0, 0, 0, 0, 0, 0, 1]),
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic)
dataset\n--------------------------------------------\n\n**Data Set Character
istics:**\n\n    :Number of Instances: 569\n\n    :Number of Attributes: 30 n
umeric, predictive attributes and the class\n\n    :Attribute Information:\n
- radius (mean of distances from center to points on the perimeter)\n
- texture (standard deviation of gray-scale values)\n        - perimeter\n
- area\n        - smoothness (local variation in radius lengths)\n        - c
ompactness (perimeter^2 / area - 1.0)\n        - concavity (severity of conca
ve portions of the contour)\n        - concave points (number of concave port
ions of the contour)\n        - symmetry \n        - fractal dimension ("coas
tline approximation" - 1)\n\n        The mean, standard error, and "worst" or
largest (mean of the three\n        largest values) of these features were co
```

```
mputed for each image,\n        resulting in 30 features.  For instance, fiel
d 3 is Mean Radius, field\n        13 is Radius SE, field 23 is Worst Radiu
s.\n\n        - class:\n                - WDBC-Malignant\n                - W
DBC-Benign\n\n    :Summary Statistics:\n\n    ==============================
====== ====== ======\n                                           Min    Max\n
===================================== ====== ======\n    radius (mean):
6.981  28.11\n    texture (mean):                       9.71   39.28\n    per
imeter (mean):                     43.79  188.5\n    area (mean):
143.5  2501.0\n    smoothness (mean):                    0.053  0.163\n    co
mpactness (mean):                    0.019  0.345\n    concavity (mean):
0.0    0.427\n    concave points (mean):                0.0    0.201\n    sym
metry (mean):                      0.106  0.304\n    fractal dimension (mea
n):             0.05   0.097\n    radius (standard error):              0.112
2.873\n    texture (standard error):             0.36   4.885\n    perimeter
(standard error):           0.757  21.98\n    area (standard error):
6.802  542.2\n    smoothness (standard error):          0.002  0.031\n    com
pactness (standard error):         0.002  0.135\n    concavity (standard erro
r):           0.0    0.396\n    concave points (standard error):      0.0
0.053\n    symmetry (standard error):            0.008  0.079\n    fractal di
mension (standard error):   0.001  0.03\n    radius (worst):
7.93   36.04\n    texture (worst):                      12.02  49.54\n    per
imeter (worst):                    50.41  251.2\n    area (worst):
185.2  4254.0\n    smoothness (worst):                   0.071  0.223\n    co
mpactness (worst):                   0.027  1.058\n    concavity (worst):
0.0    1.252\n    concave points (worst):               0.0    0.291\n    sym
metry (worst):                     0.156  0.664\n    fractal dimension (wors
t):            0.055  0.208\n    ===================================== ======
======\n\n    :Missing Attribute Values: None\n\n    :Class Distribution: 212
- Malignant, 357 - Benign\n\n    :Creator:  Dr. William H. Wolberg, W. Nick S
treet, Olvi L. Mangasarian\n\n    :Donor: Nick Street\n\n    :Date: November,
1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) dataset
s.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of
a fine needle\naspirate (FNA) of a breast mass.  They describe\ncharacteristi
cs of the cell nuclei present in the image.\n\nSeparating plane described abo
ve was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Deci
sion Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidw
est Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 199
2], a classification method which uses linear\nprogramming to construct a dec
ision tree.  Relevant features\nwere selected using an exhaustive search in t
he space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear pro
gram used to obtain the separating plane\nin the 3-dimensional space is that
described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramm
ing Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods a
nd Software 1, 1992, 23-34].\n\nThis database is also available through the U
W CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-lea
rn/WDBC/\n\n.. topic:: References\n\n   - W.N. Street, W.H. Wolberg and O.L.
Mangasarian. Nuclear feature extraction \n     for breast tumor diagnosis. IS
&T/SPIE 1993 International Symposium on \n     Electronic Imaging: Science an
d Technology, volume 1905, pages 861-870,\n     San Jose, CA, 1993.\n   - O.
L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and \n
prognosis via linear programming. Operations Research, 43(4), pages 570-577,
\n     July-August 1995.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasaria
n. Machine learning techniques\n     to diagnose breast cancer from fine-need
le aspirates. Cancer Letters 77 (1994) \n     163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'me
an 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'], dtype='<U23'),
 'filename': 'C:\\Users\\Alekhya\\Anaconda3\\lib\\site-packages\\sklearn\\dat
asets\\data\\breast_cancer.csv'}
```

In [5]:

```
cancer["data"]
```

Out[5]:

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

In [7]:

```
df = pd.DataFrame(cancer["data"],columns=['mean radius', 'mean texture', 'me
        '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'])
df.head()
```

Out[7]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	d
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 30 columns

In [8]:
```
df["target"] = cancer["target"]
df.head()
```

Out[8]:

mean ymmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points
0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654
0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860
0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430
0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575
0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625

In [9]:
```
df.shape
```

Out[9]: (569, 31)

In [10]:
```
# checking for null values
df.isnull().sum()
```

Out[10]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
mean radius                569 non-null float64
mean texture               569 non-null float64
mean perimeter             569 non-null float64
mean area                  569 non-null float64
mean smoothness            569 non-null float64
mean compactness           569 non-null float64
mean concavity             569 non-null float64
mean concave points        569 non-null float64
mean symmetry              569 non-null float64
mean fractal dimension     569 non-null float64
radius error               569 non-null float64
texture error              569 non-null float64
perimeter error            569 non-null float64
area error                 569 non-null float64
smoothness error           569 non-null float64
compactness error          569 non-null float64
concavity error            569 non-null float64
concave points error       569 non-null float64
symmetry error             569 non-null float64
fractal dimension error    569 non-null float64
worst radius               569 non-null float64
worst texture              569 non-null float64
worst perimeter            569 non-null float64
worst area                 569 non-null float64
worst smoothness           569 non-null float64
worst compactness          569 non-null float64
worst concavity            569 non-null float64
worst concave points       569 non-null float64
worst symmetry             569 non-null float64
worst fractal dimension    569 non-null float64
target                     569 non-null int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

In [12]:
```
df.describe()
```

Out[12]:

mean fractal nension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points
.000000	...	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
.062798	...	25.677223	107.261213	880.583128	0.132369	0.254265	0.272188	0.114606
.007060	...	6.146258	33.602542	569.356993	0.022832	0.157336	0.208624	0.065732
.049960	...	12.020000	50.410000	185.200000	0.071170	0.027290	0.000000	0.000000
.057700	...	21.080000	84.110000	515.300000	0.116600	0.147200	0.114500	0.064930
.061540	...	25.410000	97.660000	686.500000	0.131300	0.211900	0.226700	0.099930
.066120	...	29.720000	125.400000	1084.000000	0.146000	0.339100	0.382900	0.161400
.097440	...	49.540000	251.200000	4254.000000	0.222600	1.058000	1.252000	0.291000

In [13]:
```
df.columns
```

Out[13]:
```
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 error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension',
       'target'],
      dtype='object')
```

In [14]:
```
# extracting features from original datset
x = df.drop("target",axis=1)
```

In [15]:
```
x.head()
```

Out[15]:

mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	cc
0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	
0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	
0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	
0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	
0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	

In [16]:
```
# selecting the target
y = df["target"]
```

In [17]:
```
y.head()
```

Out[17]:
```
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int32
```

In [19]:
```
# Applying scaling technique
from sklearn.preprocessing import StandardScaler
```

In [20]:
```
scaler = StandardScaler()
```

In [22]:
```
x_transformed = scaler.fit_transform(x)
```

In [23]:
```
x_transformed
```

Out[23]:
```
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 [26]:
```
s = pd.DataFrame(x_transformed)
```

In [27]:
```
s.describe()
```

Out[27]:

	0	1	2	3	4	5
count	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02
mean	-3.162867e-15	-6.530609e-15	-7.078891e-16	-8.799835e-16	6.132177e-15	-1.120369e-15
std	1.000880e+00	1.000880e+00	1.000880e+00	1.000880e+00	1.000880e+00	1.000880e+00
min	-2.029648e+00	-2.229249e+00	-1.984504e+00	-1.454443e+00	-3.112085e+00	-1.610136e+00
25%	-6.893853e-01	-7.259631e-01	-6.919555e-01	-6.671955e-01	-7.109628e-01	-7.470860e-01
50%	-2.150816e-01	-1.046362e-01	-2.359800e-01	-2.951869e-01	-3.489108e-02	-2.219405e-01
75%	4.693926e-01	5.841756e-01	4.996769e-01	3.635073e-01	6.361990e-01	4.938569e-01
max	3.971288e+00	4.651889e+00	3.976130e+00	5.250529e+00	4.770911e+00	4.568425e+00

8 rows × 30 columns

In [28]:
```
# Apply PCA Model
from sklearn.decomposition import PCA
```

In [29]:
```
1 help(PCA)
```

```
Help on class PCA in module sklearn.decomposition.pca:

class PCA(sklearn.decomposition.base._BasePCA)
 |  PCA(n_components=None, copy=True, whiten=False, svd_solver='auto', tol=0.0,
iterated_power='auto', random_state=None)
 |
 |  Principal component analysis (PCA)
 |
 |  Linear dimensionality reduction using Singular Value Decomposition of the
 |  data to project it to a lower dimensional space.
 |
 |  It uses the LAPACK implementation of the full SVD or a randomized truncated
 |  SVD by the method of Halko et al. 2009, depending on the shape of the input
 |  data and the number of components to extract.
 |
 |  It can also use the scipy.sparse.linalg ARPACK implementation of the
 |  truncated SVD.
 |
 |  Notice that this class does not support sparse input. See
 |  :class:`TruncatedSVD` for an alternative with sparse data.
 |
 |  Read more in the :ref:`User Guide <PCA>`.
 |
 |  Parameters
 |  ----------
 |  n_components : int, float, None or string
 |      Number of components to keep.
 |      if n_components is not set all components are kept::
 |
 |          n_components == min(n_samples, n_features)
 |
 |      If ``n_components == 'mle'`` and ``svd_solver == 'full'``, Minka's
 |      MLE is used to guess the dimension. Use of ``n_components == 'mle'``
 |      will interpret ``svd_solver == 'auto'`` as ``svd_solver == 'full'``.
 |
 |      If ``0 < n_components < 1`` and ``svd_solver == 'full'``, select the
 |      number of components such that the amount of variance that needs to be
 |      explained is greater than the percentage specified by n_components.
 |
 |      If ``svd_solver == 'arpack'``, the number of components must be
 |      strictly less than the minimum of n_features and n_samples.
 |
 |      Hence, the None case results in::
 |
 |          n_components == min(n_samples, n_features) - 1
 |
 |  copy : bool (default True)
 |      If False, data passed to fit are overwritten and running
 |      fit(X).transform(X) will not yield the expected results,
 |      use fit_transform(X) instead.
 |
 |  whiten : bool, optional (default False)
 |      When True (False by default) the `components_` vectors are multiplied
 |      by the square root of n_samples and then divided by the singular values
```

```
 |      to ensure uncorrelated outputs with unit component-wise variances.
 |
 |      Whitening will remove some information from the transformed signal
 |      (the relative variance scales of the components) but can sometime
 |      improve the predictive accuracy of the downstream estimators by
 |      making their data respect some hard-wired assumptions.
 |
 |  svd_solver : string {'auto', 'full', 'arpack', 'randomized'}
 |      auto :
 |          the solver is selected by a default policy based on `X.shape` and
 |          `n_components`: if the input data is larger than 500x500 and the
 |          number of components to extract is lower than 80% of the smallest
 |          dimension of the data, then the more efficient 'randomized'
 |          method is enabled. Otherwise the exact full SVD is computed and
 |          optionally truncated afterwards.
 |      full :
 |          run exact full SVD calling the standard LAPACK solver via
 |          `scipy.linalg.svd` and select the components by postprocessing
 |      arpack :
 |          run SVD truncated to n_components calling ARPACK solver via
 |          `scipy.sparse.linalg.svds`. It requires strictly
 |          0 < n_components < min(X.shape)
 |      randomized :
 |          run randomized SVD by the method of Halko et al.
 |
 |      .. versionadded:: 0.18.0
 |
 |  tol : float >= 0, optional (default .0)
 |      Tolerance for singular values computed by svd_solver == 'arpack'.
 |
 |      .. versionadded:: 0.18.0
 |
 |  iterated_power : int >= 0, or 'auto', (default 'auto')
 |      Number of iterations for the power method computed by
 |      svd_solver == 'randomized'.
 |
 |      .. versionadded:: 0.18.0
 |
 |  random_state : int, RandomState instance or None, optional (default None)
 |      If int, random_state is the seed used by the random number generator;
 |      If RandomState instance, random_state is the random number generator;
 |      If None, the random number generator is the RandomState instance used
 |      by `np.random`. Used when ``svd_solver`` == 'arpack' or 'randomized'.
 |
 |      .. versionadded:: 0.18.0
 |
 |  Attributes
 |  ----------
 |  components_ : array, shape (n_components, n_features)
 |      Principal axes in feature space, representing the directions of
 |      maximum variance in the data. The components are sorted by
 |      ``explained_variance_``.
 |
 |  explained_variance_ : array, shape (n_components,)
 |      The amount of variance explained by each of the selected components.
 |
 |      Equal to n_components largest eigenvalues
```

```
 |      of the covariance matrix of X.
 |
 |      .. versionadded:: 0.18
 |
 |  explained_variance_ratio_ : array, shape (n_components,)
 |      Percentage of variance explained by each of the selected components.
 |
 |      If ``n_components`` is not set then all components are stored and the
 |      sum of the ratios is equal to 1.0.
 |
 |  singular_values_ : array, shape (n_components,)
 |      The singular values corresponding to each of the selected components.
 |      The singular values are equal to the 2-norms of the ``n_components``
 |      variables in the lower-dimensional space.
 |
 |  mean_ : array, shape (n_features,)
 |      Per-feature empirical mean, estimated from the training set.
 |
 |      Equal to `X.mean(axis=0)`.
 |
 |  n_components_ : int
 |      The estimated number of components. When n_components is set
 |      to 'mle' or a number between 0 and 1 (with svd_solver == 'full') this
 |      number is estimated from input data. Otherwise it equals the parameter
 |      n_components, or the lesser value of n_features and n_samples
 |      if n_components is None.
 |
 |  noise_variance_ : float
 |      The estimated noise covariance following the Probabilistic PCA model
 |      from Tipping and Bishop 1999. See "Pattern Recognition and
 |      Machine Learning" by C. Bishop, 12.2.1 p. 574 or
 |      http://www.miketipping.com/papers/met-mppca.pdf. (http://www.miketippin
g.com/papers/met-mppca.pdf.) It is required to
 |      compute the estimated data covariance and score samples.
 |
 |      Equal to the average of (min(n_features, n_samples) - n_components)
 |      smallest eigenvalues of the covariance matrix of X.
 |
 |  References
 |  ----------
 |  For n_components == 'mle', this class uses the method of `Minka, T. P.
 |  "Automatic choice of dimensionality for PCA". In NIPS, pp. 598-604`
 |
 |  Implements the probabilistic PCA model from:
 |  `Tipping, M. E., and Bishop, C. M. (1999). "Probabilistic principal
 |  component analysis". Journal of the Royal Statistical Society:
 |  Series B (Statistical Methodology), 61(3), 611-622.
 |  via the score and score_samples methods.
 |  See http://www.miketipping.com/papers/met-mppca.pdf (http://www.miketippin
g.com/papers/met-mppca.pdf)
 |
 |  For svd_solver == 'arpack', refer to `scipy.sparse.linalg.svds`.
 |
 |  For svd_solver == 'randomized', see:
 |  `Halko, N., Martinsson, P. G., and Tropp, J. A. (2011).
 |  "Finding structure with randomness: Probabilistic algorithms for
 |  constructing approximate matrix decompositions".
```

```
 |  SIAM review, 53(2), 217-288.` and also
 |  `Martinsson, P. G., Rokhlin, V., and Tygert, M. (2011).
 |  "A randomized algorithm for the decomposition of matrices".
 |  Applied and Computational Harmonic Analysis, 30(1), 47-68.`
 |
 |
 |  Examples
 |  --------
 |  >>> import numpy as np
 |  >>> from sklearn.decomposition import PCA
 |  >>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
 |  >>> pca = PCA(n_components=2)
 |  >>> pca.fit(X)
 |  PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
 |    svd_solver='auto', tol=0.0, whiten=False)
 |  >>> print(pca.explained_variance_ratio_)  # doctest: +ELLIPSIS
 |  [0.9924... 0.0075...]
 |  >>> print(pca.singular_values_)  # doctest: +ELLIPSIS
 |  [6.30061... 0.54980...]
 |
 |  >>> pca = PCA(n_components=2, svd_solver='full')
 |  >>> pca.fit(X)                 # doctest: +ELLIPSIS +NORMALIZE_WHITESPACE
 |  PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
 |    svd_solver='full', tol=0.0, whiten=False)
 |  >>> print(pca.explained_variance_ratio_)  # doctest: +ELLIPSIS
 |  [0.9924... 0.00755...]
 |  >>> print(pca.singular_values_)  # doctest: +ELLIPSIS
 |  [6.30061... 0.54980...]
 |
 |  >>> pca = PCA(n_components=1, svd_solver='arpack')
 |  >>> pca.fit(X)
 |  PCA(copy=True, iterated_power='auto', n_components=1, random_state=None,
 |    svd_solver='arpack', tol=0.0, whiten=False)
 |  >>> print(pca.explained_variance_ratio_)  # doctest: +ELLIPSIS
 |  [0.99244...]
 |  >>> print(pca.singular_values_)  # doctest: +ELLIPSIS
 |  [6.30061...]
 |
 |  See also
 |  --------
 |  KernelPCA
 |  SparsePCA
 |  TruncatedSVD
 |  IncrementalPCA
 |
 |  Method resolution order:
 |      PCA
 |      sklearn.decomposition.base._BasePCA
 |      abc.NewBase
 |      sklearn.base.BaseEstimator
 |      sklearn.base.TransformerMixin
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, n_components=None, copy=True, whiten=False, svd_solver='aut
o', tol=0.0, iterated_power='auto', random_state=None)
```

```
 |      Initialize self.  See help(type(self)) for accurate signature.
 |  
 |  fit(self, X, y=None)
 |      Fit the model with X.
 |      
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_samples, n_features)
 |          Training data, where n_samples is the number of samples
 |          and n_features is the number of features.
 |      
 |      y : Ignored
 |      
 |      Returns
 |      -------
 |      self : object
 |          Returns the instance itself.
 |  
 |  fit_transform(self, X, y=None)
 |      Fit the model with X and apply the dimensionality reduction on X.
 |      
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_samples, n_features)
 |          Training data, where n_samples is the number of samples
 |          and n_features is the number of features.
 |      
 |      y : Ignored
 |      
 |      Returns
 |      -------
 |      X_new : array-like, shape (n_samples, n_components)
 |  
 |  score(self, X, y=None)
 |      Return the average log-likelihood of all samples.
 |      
 |      See. "Pattern Recognition and Machine Learning"
 |      by C. Bishop, 12.2.1 p. 574
 |      or http://www.miketipping.com/papers/met-mppca.pdf (http://www.miketipp
ing.com/papers/met-mppca.pdf)
 |      
 |      Parameters
 |      ----------
 |      X : array, shape(n_samples, n_features)
 |          The data.
 |      
 |      y : Ignored
 |      
 |      Returns
 |      -------
 |      ll : float
 |          Average log-likelihood of the samples under the current model
 |  
 |  score_samples(self, X)
 |      Return the log-likelihood of each sample.
 |      
 |      See. "Pattern Recognition and Machine Learning"
```

```
 |      by C. Bishop, 12.2.1 p. 574
 |      or http://www.miketipping.com/papers/met-mppca.pdf (http://www.miketipp
ing.com/papers/met-mppca.pdf)
 |
 |      Parameters
 |      ----------
 |      X : array, shape(n_samples, n_features)
 |          The data.
 |
 |      Returns
 |      -------
 |      ll : array, shape (n_samples,)
 |          Log-likelihood of each sample under the current model
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.decomposition.base._BasePCA:
 |
 |  get_covariance(self)
 |      Compute data covariance with the generative model.
 |
 |      ``cov = components_.T * S**2 * components_ + sigma2 * eye(n_features)``
 |      where  S**2 contains the explained variances, and sigma2 contains the
 |      noise variances.
 |
 |      Returns
 |      -------
 |      cov : array, shape=(n_features, n_features)
 |          Estimated covariance of data.
 |
 |  get_precision(self)
 |      Compute data precision matrix with the generative model.
 |
 |      Equals the inverse of the covariance but computed with
 |      the matrix inversion lemma for efficiency.
 |
 |      Returns
 |      -------
 |      precision : array, shape=(n_features, n_features)
 |          Estimated precision of data.
 |
 |  inverse_transform(self, X)
 |      Transform data back to its original space.
 |
 |      In other words, return an input X_original whose transform would be X.
 |
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_samples, n_components)
 |          New data, where n_samples is the number of samples
 |          and n_components is the number of components.
 |
 |      Returns
```

```
 |      -------
 |      X_original array-like, shape (n_samples, n_features)
 |
 |      Notes
 |      -----
 |      If whitening is enabled, inverse_transform will compute the
 |      exact inverse operation, which includes reversing whitening.
 |
 |  transform(self, X)
 |      Apply dimensionality reduction to X.
 |
 |      X is projected on the first principal components previously extracted
 |      from a training set.
 |
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_samples, n_features)
 |          New data, where n_samples is the number of samples
 |          and n_features is the number of features.
 |
 |      Returns
 |      -------
 |      X_new : array-like, shape (n_samples, n_components)
 |
 |      Examples
 |      --------
 |
 |      >>> import numpy as np
 |      >>> from sklearn.decomposition import IncrementalPCA
 |      >>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3,
2]])
 |      >>> ipca = IncrementalPCA(n_components=2, batch_size=3)
 |      >>> ipca.fit(X)
 |      IncrementalPCA(batch_size=3, copy=True, n_components=2, whiten=False)
 |      >>> ipca.transform(X) # doctest: +SKIP
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |
 |      Parameters
 |      ----------
 |      deep : boolean, optional
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |
 |      Returns
 |      -------
```

```
 |      params : mapping of string to any
 |          Parameter names mapped to their values.
 |
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |
 |      The method works on simple estimators as well as on nested objects
 |      (such as pipelines). The latter have parameters of the form
 |      ``<component>__<parameter>`` so that it's possible to update each
 |      component of a nested object.
 |
 |      Returns
 |      -------
 |      self
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.BaseEstimator:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [30]:
```
pca = PCA(n_components=2)
```

In [31]:
```
newfeatures = pca.fit_transform(x_transformed)
```

In [32]:
```
newfeatures
```

Out[32]:
```
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 [35]:
```
pca_df = pd.DataFrame(newfeatures,columns=["pca1","pca2"])
pca_df.head()
```

Out[35]:

	pca1	pca2
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

In [36]:
```
pca.explained_variance_ratio_
```

Out[36]:
```
array([0.44272026, 0.18971182])
```

In [37]:
```
sum(pca.explained_variance_ratio_)
```

Out[37]:
```
0.6324320765155947
```

In [ ]:
```

```