MACHINE LEARNING LAB-7:PCA&LDA

Submitted by:
Name: **Stebin George**
Register Number: **21122061**
Class: **2MSCDS**
Time Taken: **5 hours**

Lab Overview

Objectives

TO get to know more about the PCA and LDA two major dimensionality reduction techniques in ML

LIBRARIES:

In [33]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn import datasets
from sklearn.decomposition import PCA
```

Questions:

- Part A. Perform PCA and LDA on Breast Cancer Dataset, write down your obsevations. While loading, use the toy dataset available in SKLearn (load_breast_cancer)
- Part B. Illustrate the effect of changing various method parameters of PCA and LDA. Compare the accuracies, and provide visualizations and interpretations for the evaluation metrices.
- Part C. "PCA could be used in applications such as Image Processing, to reduce the complexity of data and improve performance or to compress images". Justify this statement with your own findings.

Problem Definition:

The problem tries to give a introducrion about Pca and Lda in dimensionality reduction.

Approach

- Importing the necessary libraries
- analysing the data and doing the basic operations

- doing the EDA and pre-processing steps
- Loading the PCA & LDA function
- Plotting the accuracy
- checking for the accuracy
- usage of images to show compression using PCA

Sections

1. Libraries
2. introduction
3. EDA
4. PCA&LDA
5. Q&A
6. Accuracy Table
7. Conclusion

Loading Dataset:

In [265…

```
df=datasets.load_breast_cancer(as_frame=True)
df1=df.data
df1['diagnosis']=df.target
```

In [3]:

```
df1.head()
```

Out[3]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	di
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 × 31 columns

In [4]:

```
#checking the shape
df1.shape
```

Out[4]:

```
(569, 31)
```

Datset has 569 rows and 33 columns

In [5]:

```
df1.describe()
```

Out[5]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	c
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.

8 rows × 31 columns

GIVES THE SUMMARY OF THE DATASET

CHECKING NULL VALUES:

In [6]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  diagnosis                569 non-null    int32  
```

```
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

In [7]:

```
df1.isna().sum()
```

Out[7]:

```
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
diagnosis                  0
dtype: int64
```

We dont have any null values here

EDA:

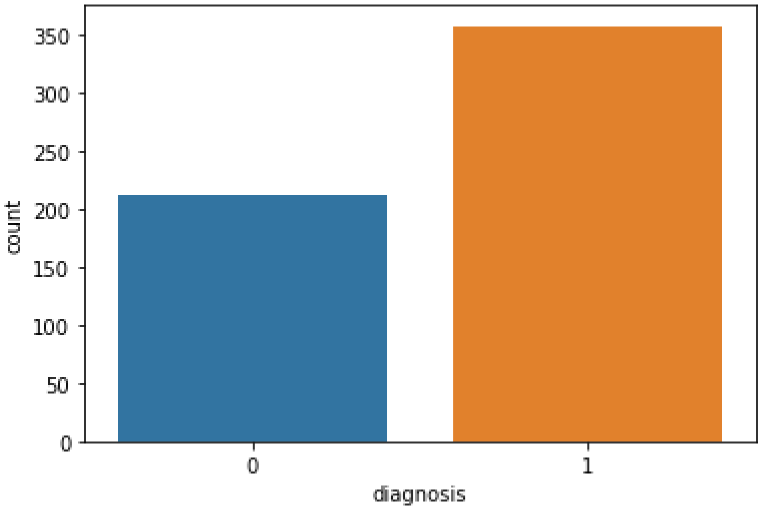
In [8]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [9]:

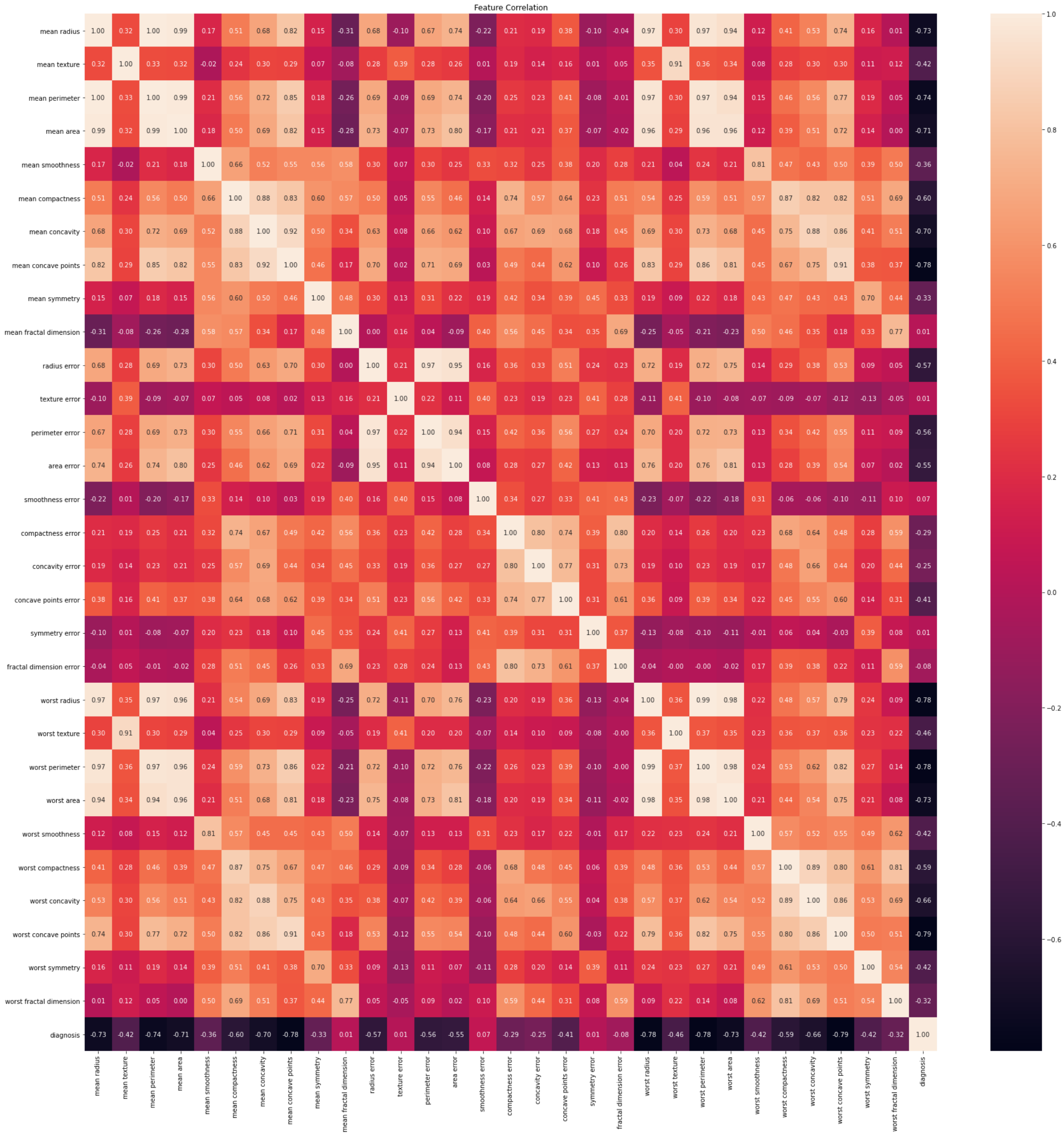
```
ax = sns.countplot(df1.diagnosis,label="Count")
B, M =df1.diagnosis.value_counts()
print('Number of Benign: ',B)
print('Number of Malignant : ',M)
```

```
Number of Benign:  357
Number of Malignant :  212
```



In [10]:

```
# Correlation Matrix
plt.figure(figsize=(30,30))
corr_matrix = df1.corr()
sns.heatmap(corr_matrix, annot = True, fmt = '.2f',)
plt.title("Feature Correlation")
plt.show()
```



The above graph shows the correlation plot of the different variables in the dataset.

In [11]:

```
df1.corr()
```

Out[11]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	r con p
mean radius	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.82
mean texture	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.29
mean perimeter	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.85
mean area	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.82
mean smoothness	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.55
mean compactness	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.83

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	r con p
mean concavity	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.92
mean concave points	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.00
mean symmetry	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.46
mean fractal dimension	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.16
radius error	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925	0.69
texture error	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218	0.02
perimeter error	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391	0.71
area error	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427	0.69
smoothness error	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564	0.02
compactness error	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279	0.49
concavity error	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270	0.43
concave points error	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260	0.61
symmetry error	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009	0.09
fractal dimension error	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301	0.25
worst radius	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688236	0.83
worst texture	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299879	0.29
worst perimeter	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729565	0.85
worst area	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675987	0.80
worst smoothness	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.448822	0.45
worst compactness	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.754968	0.66
worst concavity	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.884103	0.75
worst concave points	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.861323	0.91
worst symmetry	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.409464	0.37

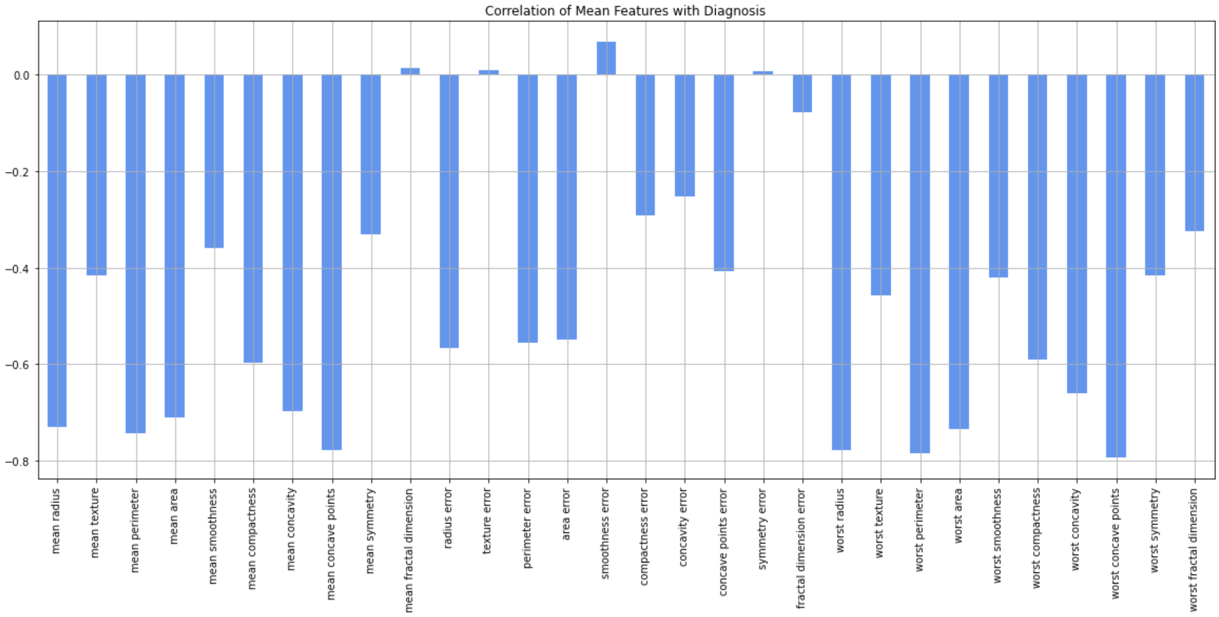
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	r con p
worst fractal dimension	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.514930	0.36
diagnosis	-0.730029	-0.415185	-0.742636	-0.708984	-0.358560	-0.596534	-0.696360	-0.77

31 rows × 31 columns

Feature contributions to the target variable:

In [19]:

```
df_mean = df1[df1.columns[:]]
plt.figure(figsize=(20, 8))
df_mean.drop('diagnosis', axis=1).corrwith(df_mean.diagnosis).plot(kind='bar', grid=
```



Splitting the data

In [28]:

```
X = df1.iloc[:,1:-1]
y = df1.iloc[:,-1]
```

Standardizing the dataset:

In [110…

```
sc = StandardScaler()
X_stand = sc.fit_transform(X)
```

Question1)

Part A. Perform PCA and LDA on Breast Cancer Dataset, write down your obsevations. While loading, use the toy dataset available in SKLearn (load_breast_cancer)

NORMAL DATA:

In [97]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.8,random_state
```

In [98]:

```
classifer = LogisticRegression()
classifer.fit(X_train, y_train)
predictions  = classifer.predict(X_test)
predictions
```

Out[98]:

```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0])
```

In [99]:

```
print(accuracy_score(y_test,predictions))
```

```
0.9407894736842105
```

PCA:

In [275…

```
pca = PCA(n_components=3)
pca.fit(X_stand)
PCA(copy=True, iterated_power='auto', n_components=3, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
X_pca = pca.transform(X_stand)
X_pca.shape
```

Out[275…

```
(569, 3)
```

In [276…

```
explained_variance=pca.explained_variance_ratio_
explained_variance
```

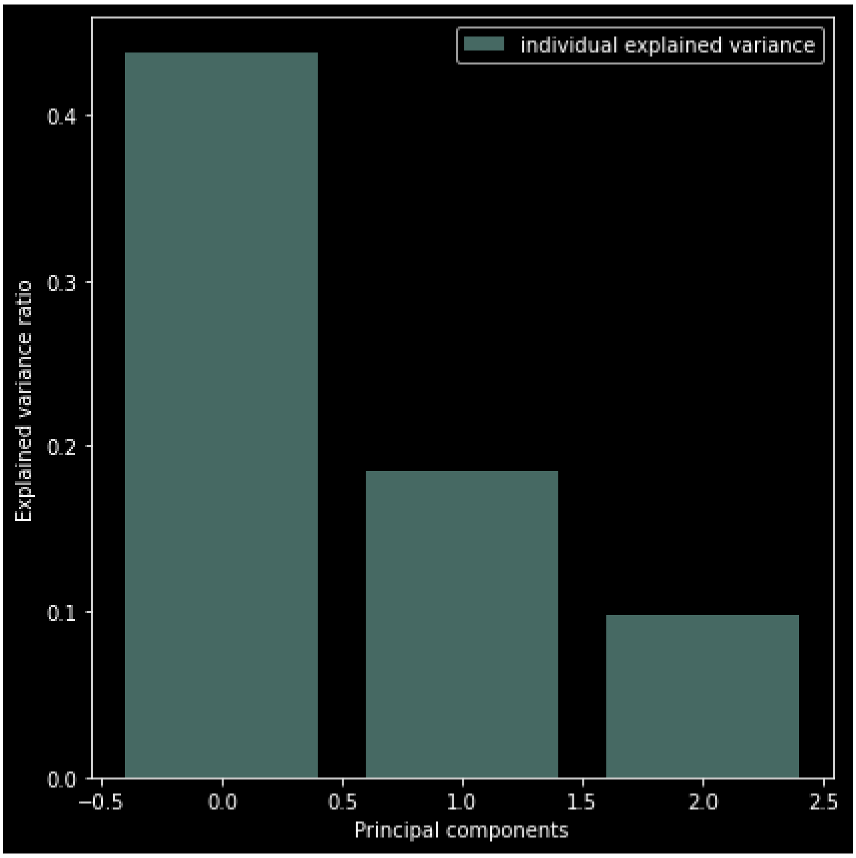
Out[276…

```
array([0.43706363, 0.18472237, 0.09716239])
```

In [277…

```
with plt.style.context('dark_background'):
    plt.figure(figsize=(6,6))

    plt.bar(range(3), explained_variance, alpha=0.5, align='center',
            label='individual explained variance')
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```



```
In [114…  X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.8,random_s
```

```
In [115…  classifer = LogisticRegression()
          classifer.fit(X_train, y_train)
```

```
Out[115… LogisticRegression()
```

```
In [116…  predictions  = classifer.predict(X_test)
          predictions
```

```
Out[116… array([1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
                0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
                0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
                0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
                1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
                0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
                0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
                0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
                0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
                1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
                1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
                1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1,
                1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
                1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
                1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
                1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1,
                1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0])
```

```
In [117…  print(accuracy_score(y_test,predictions))
```

```
0.956140350877193
```

OBSERVATION:

With PCA the logistics regression was giving better accuracy PCA has no concern with the class labels. It summarizes the feature set without considering the output. PCA tries to find the directions of the maximum variance in the dataset. In a high cardinality feature set, there are possibilities of duplicate features which would add redundancy to the dataset, increase the computation cost and add unneccessary model complexity. The role of PCA is to find such highly correlated or duplicate features and to come up with a new feature set where there is minimum correlation between the features or in other words feature set with maximum variance between the features.

LDA:

In [118…

```
X_train, X_test, y_train, y_test = train_test_split(X_stand, y, test_size=0.8,random
```

In [119…

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components = 1)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
```

In [120…

```
classifer = LogisticRegression()
classifer.fit(X_train, y_train)
```

Out[120…

```
LogisticRegression()
```

In [121…

```
predictions  = classifer.predict(X_test)
predictions
```

Out[121…

```
array([1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0])
```

In [122…

```
print(accuracy_score(y_test,predictions))
```

```
0.9320175438596491
```

OBSERVATIONS:

LDA tries to reduce the dimensionality by taking into consideration the information that discriminates the output classes. LDA tries to find the decision boundary around each cluster of class.It projects the data points to new dimension in a way that the clusters are as seperate from each other as possible and individual elements within a class are as close to the centroid as possible.In other words, the inter-class seperability is increased in LDA. Intra-class seperability is reduced. The new dimensions are the linear discriminants of the feature set.

QUESTION-2:

Part B. Illustrate the effect of changing various method parameters of PCA and LDA. Compare the accuracies, and provide visualizations and interpretations for the evaluation metrices.

In [258…

```
def standard_scalar(data):
    sc = StandardScaler()
    X_stand = sc.fit_transform(data)
    return(X_stand)
def pca(X_stand,n_components):
    pca = PCA(n_components)
    pca.fit(X_stand)
    X_pca = pca.transform(X_stand)
    return (X_pca)
def logistic(X_train, X_test, y_train, y_test):
    classifer = LogisticRegression()
    classifer.fit(X_train, y_train)
    predictions  = classifer.predict(X_test)
    acc=accuracy_score(y_test,predictions)
    return(acc)
```

In [261…

```
method=['NORMAL','PCA','LDA']
randomstate=[5,10,15,20,25]
n_components=[1,3,5,7,9]
methods=[]
accu=[]
random=[]
n_component=[]
for i in method:
    if i =='NORMAL':
        for j in randomstate:
            methods.append(i)
            random.append(j)
            n_component.append('NA')
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.8,
            acc=logistic(X_train, X_test, y_train, y_test)
            accu.append(acc)
    elif i=='PCA':
        X_stand=standard_scalar(X)
        for j in randomstate:
            for k in n_components:
                methods.append(i)
                random.append(j)
                n_component.append(k)
                X_pca=pca(X_stand,k)
                X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_s
                acc1=logistic(X_train, X_test, y_train, y_test)
                accu.append(acc1)
    elif i=='LDA':
        for j in randomstate:
            methods.append(i)
            random.append(j)
            n_component.append(1)
            X_train, X_test, y_train, y_test = train_test_split(X_stand, y, test_siz
            lda = LDA(n_components = 1)
            X_train = lda.fit_transform(X_train, y_train)
            X_test = lda.transform(X_test)
            acc1=logistic(X_train, X_test, y_train, y_test)
            accu.append(acc1)
```

ACCURACY DATAFRAME:

In [262…

```
score=pd.DataFrame()
score['method']=methods
score['randomstate']=random
score['no of components']=n_component
score['accuracy']=accu
score
```

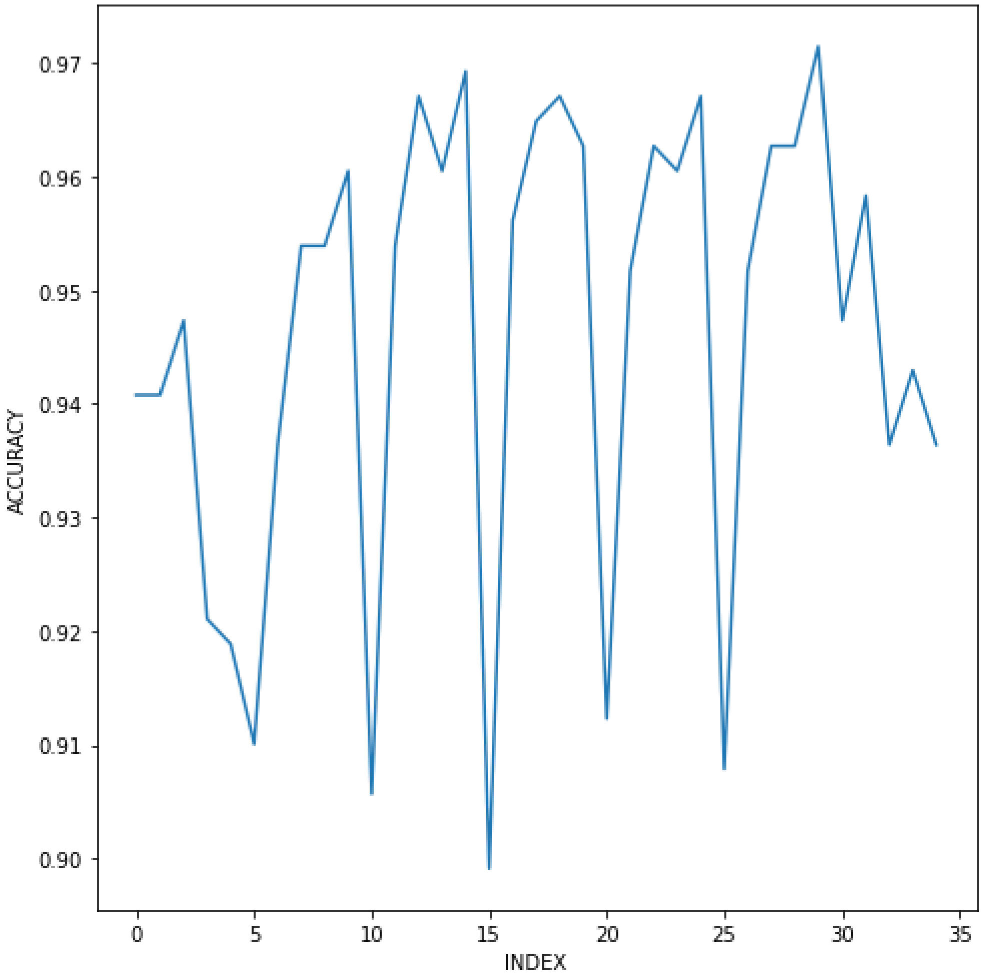
Out[262…

	method	randomstate	no of components	accuracy
0	NORMAL	5	NA	0.940789
1	NORMAL	10	NA	0.940789
2	NORMAL	15	NA	0.947368
3	NORMAL	20	NA	0.921053
4	NORMAL	25	NA	0.918860
5	PCA	5	1	0.910088
6	PCA	5	3	0.936404
7	PCA	5	5	0.953947
8	PCA	5	7	0.953947
9	PCA	5	9	0.960526
10	PCA	10	1	0.905702
11	PCA	10	3	0.953947
12	PCA	10	5	0.967105
13	PCA	10	7	0.960526
14	PCA	10	9	0.969298
15	PCA	15	1	0.899123
16	PCA	15	3	0.956140
17	PCA	15	5	0.964912
18	PCA	15	7	0.967105
19	PCA	15	9	0.962719
20	PCA	20	1	0.912281

	method	randomstate	no of components	accuracy
21	PCA	20	3	0.951754
22	PCA	20	5	0.962719
23	PCA	20	7	0.960526
24	PCA	20	9	0.967105
25	PCA	25	1	0.907895
26	PCA	25	3	0.951754
27	PCA	25	5	0.962719
28	PCA	25	7	0.962719
29	PCA	25	9	0.971491
30	LDA	5	1	0.947368
31	LDA	10	1	0.958333
32	LDA	15	1	0.936404
33	LDA	20	1	0.942982
34	LDA	25	1	0.936404

In [282…

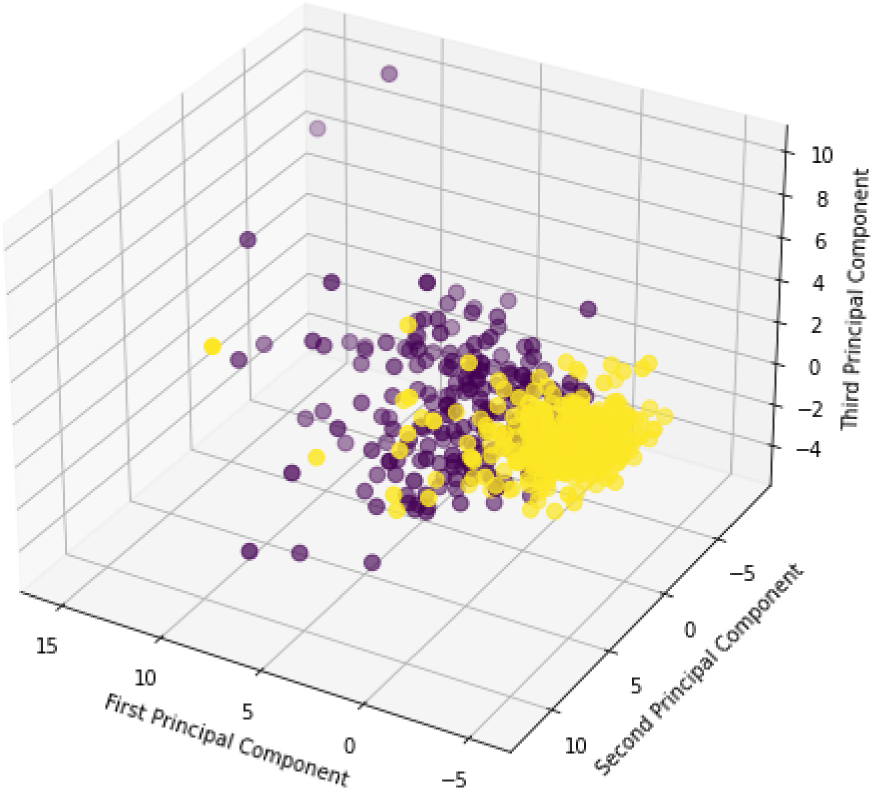
```
plt.figure(figsize=(7,7))
plt.plot(accu)
plt.ylabel('ACCURACY')
plt.xlabel('INDEX')
plt.tight_layout()
```



For the random state of 25 and a pca component of 9 we ge t the maximum accuracy of 97.14%

In [263…

```
fig = plt.figure(figsize=(15, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_pca[:,0], x_pca[:,1], x_pca[:,2], c=df1['diagnosis'], s=60)
ax.set_xlabel('First Principal Component')
ax.set_ylabel('Second Principal Component')
ax.set_zlabel('Third Principal Component')
ax.view_init(30, 120)
```

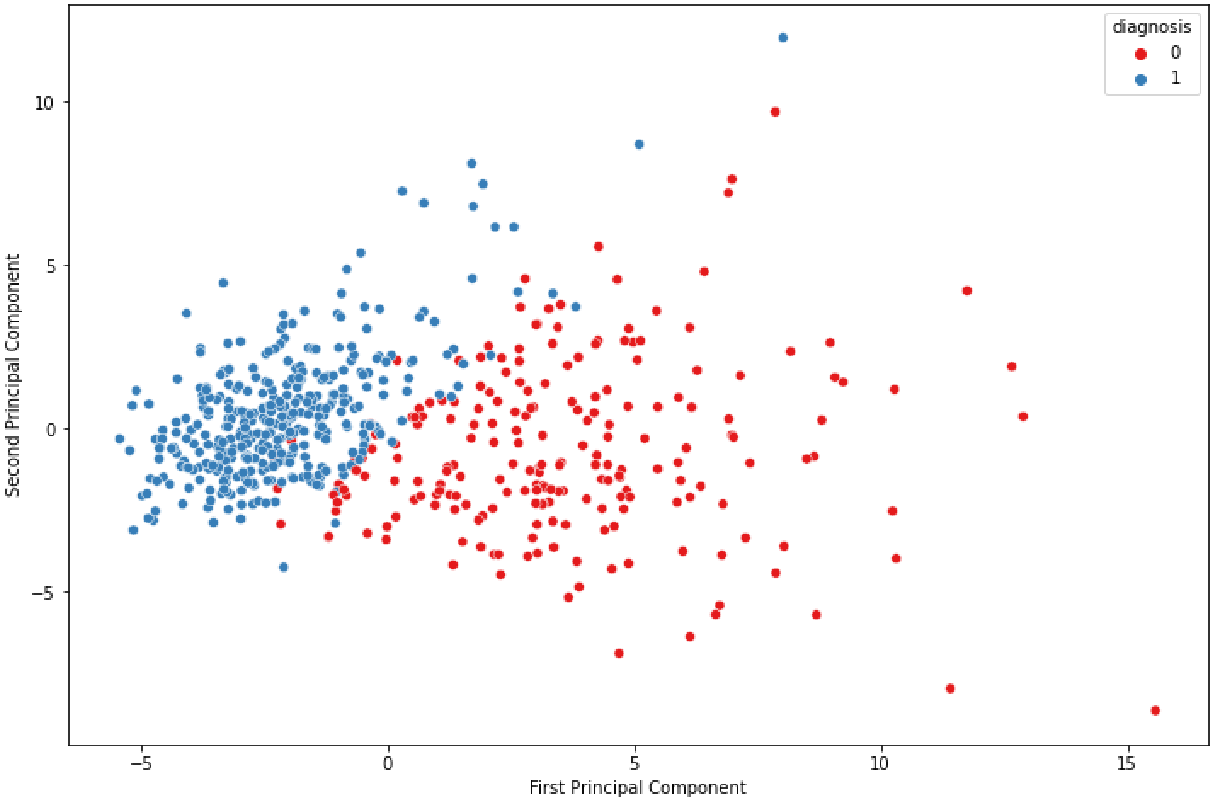


In [264…

```
ax = plt.figure(figsize=(12,8))
sns.scatterplot(x_pca[:,0], x_pca[:,1],hue=df1['diagnosis'], palette ='Set1' )
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
```

Out[264…

```
Text(0, 0.5, 'Second Principal Component')
```



QUESTION 3:

"PCA could be used in applications such as Image Processing, to reduce the complexity of data and improve performance or to compress images". Justify this statement with your own findings.

The MNIST dataset contains the image data of handwritten digits.

In [239…

```
mnist = pd.read_csv(r"C:\Users\stebi\OneDrive\Desktop\mnist.csv")
mnist.head()
```

Out[239…

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x22	28x23
0	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 785 columns

The ‘label’ column contains the values of the digit (0–9). We do not need that column for our analysis because PCA is an unsupervised machine learning task that does not deal with labelled data. So, we can simply drop that column.

In [241…

```
mnist.drop(columns='label', inplace=True)
mnist.head()
```

Out[241…

	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	1x10	...	28x19	28x20	28x21	28x22	28x23
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 784 columns

In [240…

```
mnist.shape
```

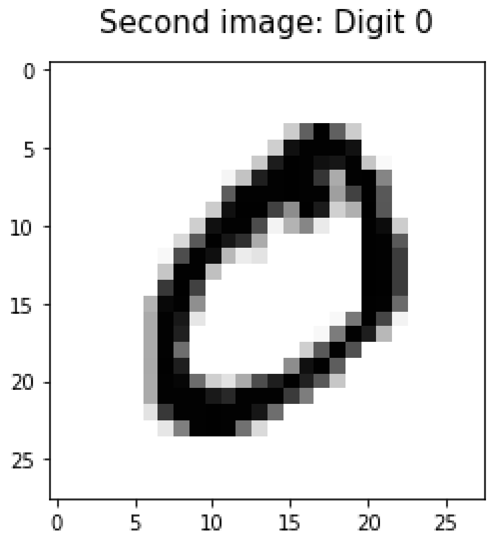
Out[240…

```
(60000, 785)
```

Plotting the image:

In [246…

```
second_image = mnist.iloc[1].values.reshape([28,28])
plt.imshow(second_image, cmap='gray_r')
plt.title('Second image: Digit 0', fontsize=15, pad=15)
plt.savefig("Second image.png")
```



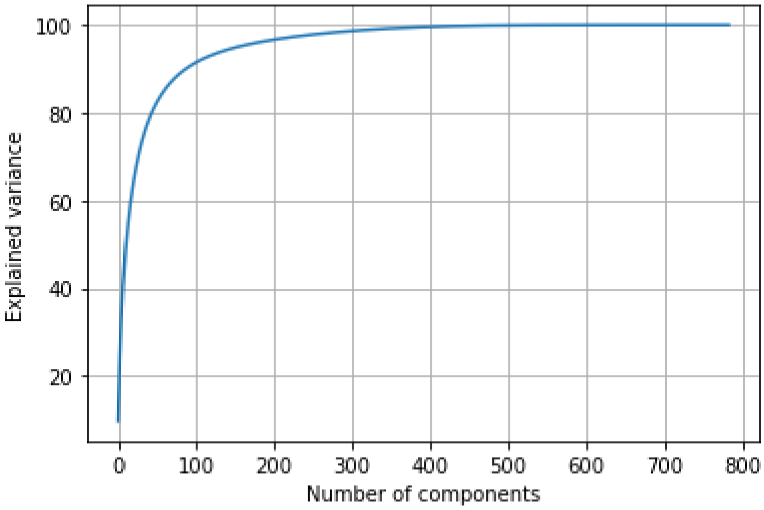
Apply PCA with the original number of dimensions (i.e., 784) and create the scree plot to see how well PCA captures the variance of the data.

In [255…

```
pca_784 = PCA(n_components=784)
pca_784.fit(mnist)
plt.grid()
plt.plot(np.cumsum(pca_784.explained_variance_ratio_ * 100))
plt.xlabel('Number of components')
plt.ylabel('Explained variance')
```

Out[255…

```
Text(0, 0.5, 'Explained variance')
```



Reducing the number of principal components to 10 and obtaining the output

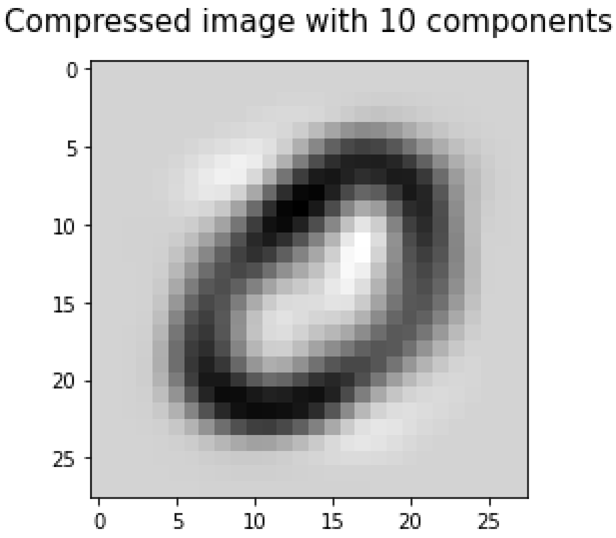
In [248…

```
pca_10 = PCA(n_components=10)
mnist_pca_10_reduced = pca_10.fit_transform(mnist)
mnist_pca_10_recovered = pca_10.inverse_transform(mnist_pca_10_reduced)

image_pca_10 = mnist_pca_10_recovered[1,:].reshape([28,28])
plt.imshow(image_pca_10, cmap='gray_r')
plt.title('Compressed image with 10 components', fontsize=15, pad=15)
```

Out[248…

```
Text(0.5, 1.0, 'Compressed image with 10 components')
```



Increasing the number of components of pca to 184 shows much more clearer image than pca with 10 components.

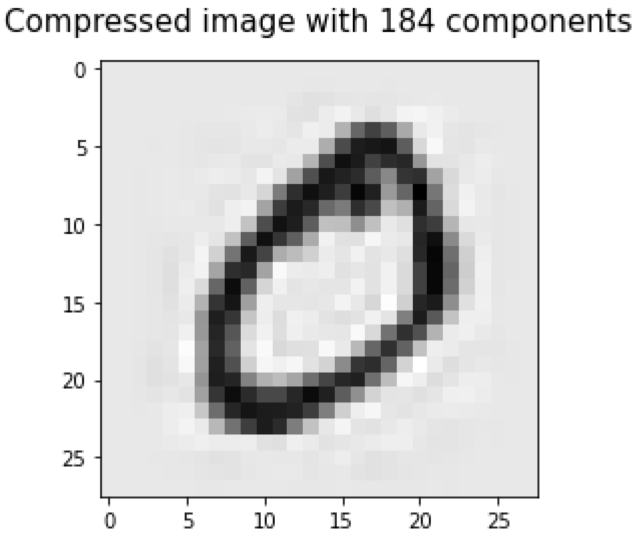
In [253…

```
pca_184 = PCA(n_components=184)
mnist_pca_184_reduced = pca_184.fit_transform(mnist)
mnist_pca_184_recovered = pca_184.inverse_transform(mnist_pca_184_reduced)

image_pca_184 = mnist_pca_184_recovered[1,:].reshape([28,28])
plt.imshow(image_pca_184, cmap='gray_r')
plt.title('Compressed image with 184 components', fontsize=15, pad=15)
```

Out[253…

```
Text(0.5, 1.0, 'Compressed image with 184 components')
```



In [254…

```
np.cumsum(pca_184.explained_variance_ratio_ * 100)[-1]
```

Out[254…

```
96.11902137200546
```

96.12% of the variance in the model is explained by the pca with 184 instead of 784.THus we actually reduced the size of the image without acyally loosing the accuracy of the model.

OBSERVATION:

The image at the left is the original image with 784 dimensions. The image at the right is the compressed image with 184 dimensions. After applying PCA on image data, the dimensionality has been reduced by 600 dimensions while keeping about 96% of the variability in the original image data! By comparing these two images, you can see that there is a slight image quality loss, but the content of the compressed image is still visible.

References:

PCA

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html https://towardsdatascience.com/principal-component-analysis-for-breast-cancer-data-with-r-and-python-b312d28e911f https://www.kaggle.com/jahirmorenoa/pca-to-the-breast-cancer-data-set https://www.youtube.com/watch?v=e2sM7ccaA9c&ab_channel=DigitalSreeni https://www.datacamp.com/community/tutorials/principal-component-analysis-in-python https://towardsdatascience.com/dimensionality-reduction-of-a-color-photo-splitting-into-rgb-channels-using-pca-algorithm-in-python-ba01580a1118 https://www.kaggle.com/mirzarahim/introduction-to-pca-image-compression-example https://github.com/gtraskas/breast_cancer_prediction/blob/master/breast_cancer.ipynb

LDA

http://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html https://machinelearningmastery.com/linear-discriminant-analysis-with-python/ https://towardsdatascience.com/linear-discriminant-analysis-in-python-76b8b17817c2 https://www.mygreatlearning.com/blog/linear-discriminant-analysis-or-lda/ https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/

In [ ]: