Chapter 13: PCA

- Principal Component Analysis (PCA) is a procedure that converts a set of observations from m to n dimensions (m > n), after analyzing the correlated features of the variables. It is used to move the data from high to a low dimension for visualization or dimensionality reduction purposes.

Applying PCA is no different than applying other estimators:

- create an estimator,
- fit it on the model to get a transformer,
- apply the transformer to the data.

PCA - Visualization

- Dataset of handwritten image has 785 columns. The first column represents a label defining the digit class (0 →9), the other 784 columns represent the pixel values of the 28*28 image.
- Use PCA to reduce the dimensions to only 3

In [1]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [2]:
```
import findspark
findspark.init()
```

In [3]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('PCA_visual').getOrCreate()
```

In [4]:
```
data = spark.read.csv('digits.csv', header=True, inferSchema=True)
```

In [5]:
```
# data.printSchema()
```

```
root
 |-- label: integer (nullable = true)
 |-- pixel0: integer (nullable = true)
 |-- pixel1: integer (nullable = true)
 |-- pixel2: integer (nullable = true)
 |-- pixel3: integer (nullable = true)
 |-- pixel4: integer (nullable = true)
 |-- pixel5: integer (nullable = true)
 |-- pixel6: integer (nullable = true)
 |-- pixel7: integer (nullable = true)
 |-- pixel8: integer (nullable = true)
 |-- pixel9: integer (nullable = true)
 |-- pixel10: integer (nullable = true)
 |-- pixel11: integer (nullable = true)
 |-- pixel12: integer (nullable = true)
 |-- pixel13: integer (nullable = true)
 |-- pixel14: integer (nullable = true)
 |-- pixel15: integer (nullable = true)
 |-- pixel16: integer (nullable = true)
```

In [6]:
```
from pyspark.ml.feature import VectorAssembler
```

In [7]:
```
# output: lable
# input: pixel0 => pixel783
assembler = VectorAssembler(inputCols=data.columns[1:], outputCol='features')
data_2 = assembler.transform(data)
```

In [8]:
```
from pyspark.ml.feature import PCA
```

In [9]:
```
pca = PCA(k=3, inputCol='features', outputCol='features_pca')
```

In [10]:
```
pca_model = pca.fit(data_2)
```

In [11]:
```
pca_model.explainedVariance
# from 784 dementions => 3 dementions: We keep only ~23% value of dataset
# we need more dementions than 3.
```

Out[11]:
```
DenseVector([0.0975, 0.0716, 0.0615])
```

In [12]:
```
pca_data = pca_model.transform(data_2).select('features_pca')
```

In [13]:
```
pca_data.show(3, truncate=False)
```

```
+-----------------------------------------------------------+
|features_pca                                               |
+-----------------------------------------------------------+
|[103.73881375798472,699.5124334036453,383.7195856009678]   |
|[2466.786278309416,360.75266138892886,-301.36804795392715]|
|[-121.55984060477854,293.9668873776094,267.5928558304658] |
+-----------------------------------------------------------+
only showing top 3 rows
```

In [14]:
```
# Change data to df: with 3 columns: label, comp1, comp2
temp = pca_data.select("features_pca").rdd.map(lambda x: x[0].toArray().tolist()).toDF()
```

In [15]:
```
temp.show(5)
```

```
+------------------+------------------+------------------+
|                _1|                _2|                _3|
+------------------+------------------+------------------+
| 103.73881375798472|  699.5124334036453|  383.7195856009678|
|  2466.786278309416| 360.75266138892886|-301.36804795392715|
|-121.55984060477854|  293.9668873776094|  267.5928558304658|
|  599.5789910719535|-299.98165533942415| 136.29206078780783|
| 2689.0443094759903|  449.3541744175643| -348.1754772429902|
+------------------+------------------+------------------+
only showing top 5 rows
```

In [16]:
```
import pyspark.sql.functions as f
```

In [17]:
```
data_label= data.select('label')
data_label = data_label.withColumn('row_index', f.monotonically_increasing_id())

temp=temp.withColumn('row_index', f.monotonically_increasing_id())

temp = temp.join(data_label, on=["row_index"]).sort("row_index").drop("row_index")
```

In [18]:
```
temp.show(3)
```

```
+------------------+------------------+------------------+-----+
|                _1|                _2|                _3|label|
+------------------+------------------+------------------+-----+
| 103.73881375798472| 699.5124334036453|  383.7195856009678|    1|
|  2466.786278309416|360.75266138892886|-301.36804795392715|    0|
|-121.55984060477854| 293.9668873776094|  267.5928558304658|    1|
+------------------+------------------+------------------+-----+
only showing top 3 rows
```

In [19]:
```
temp = temp.select(f.col("_1").alias("comp1"),
                   f.col("_2").alias("comp2"),
                   f.col("_3").alias("comp3"),
                   "label")
```

In [20]:
```
temp.show(3)
```

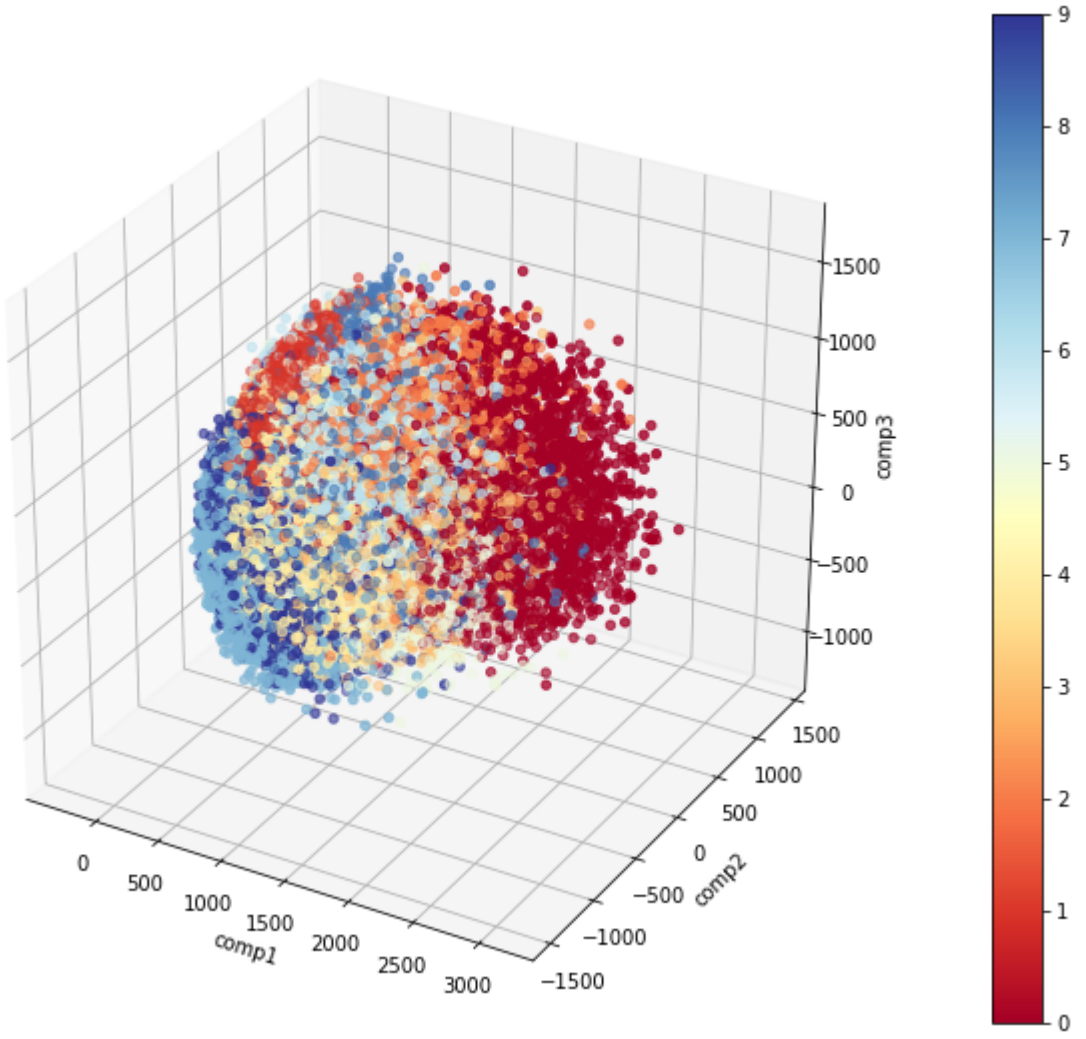
```
+------------------+-----------------+------------------+-----+
|             comp1|            comp2|             comp3|label|
+------------------+-----------------+------------------+-----+
| 103.73881375798472| 699.5124334036453|  383.7195856009678|    1|
|  2466.786278309416|360.75266138892886|-301.36804795392715|    0|
|-121.55984060477854| 293.9668873776094|  267.5928558304658|    1|
+------------------+-----------------+------------------+-----+
only showing top 3 rows
```

In [27]:
```
df = temp.toPandas()
```

In [28]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [30]:
```
from mpl_toolkits.mplot3d import Axes3D
```

In [35]:
```
fig = plt.figure(figsize=(9, 7))
ax = Axes3D(fig) # Method 1
sc = ax.scatter(df['comp1'], df['comp2'], df['comp3'],
                c=df['label'], marker='o',
                cmap= plt.cm.get_cmap('RdYlBu'))
ax.set_xlabel('comp1')
ax.set_ylabel('comp2')
ax.set_zlabel('comp3')
plt.colorbar(sc)
plt.show()
```



In [29]:
```
#Seaborn pair plot
sns.pairplot(df, hue='label')
plt.show()
```