APML CA2

Zheng Yimin / NSDAI -01 / P7053148

PART A: UNSUPERVISED LEARNING

```
# Importing the wine dataset and reading it
df = pd.read_csv(
    filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data',
    header=None,
    sep=',')

df.columns=['Class','Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium',
            'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
            'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline']


df.head()
```

	Class	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

IMPORT DATA | EDA | PCA | K-MEANS | GMM | HIERACHICAL CLUSTERING | EVALUATION

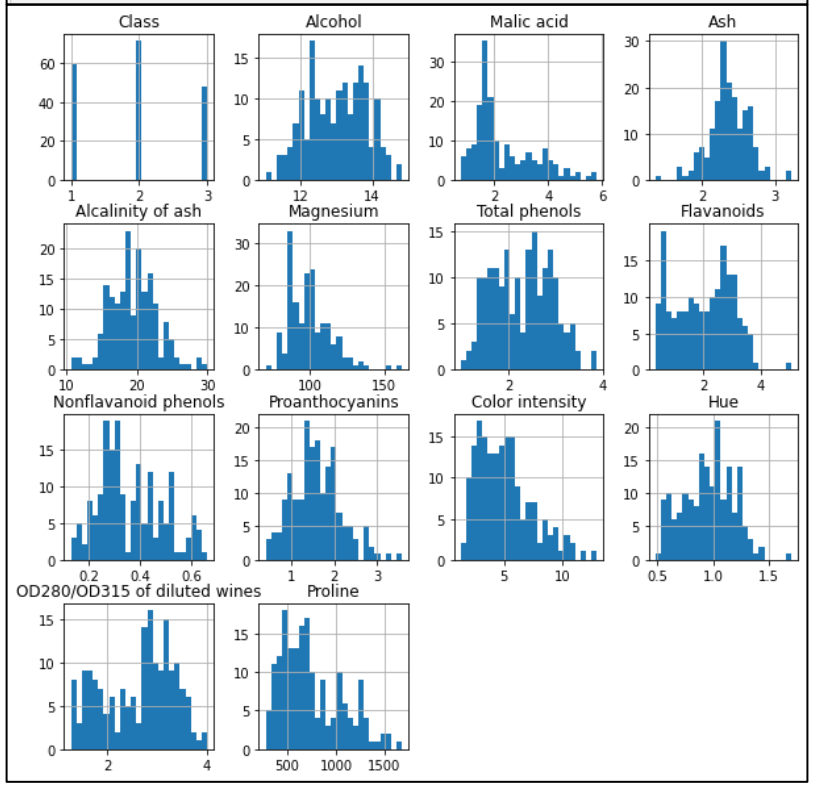
```python
# Check for missing values in dataset
df.isnull().sum()
```

```
Class                           0
Alcohol                         0
Malic acid                      0
Ash                             0
Alcalinity of ash               0
Magnesium                       0
Total phenols                   0
Flavanoids                      0
Nonflavanoid phenols            0
Proanthocyanins                 0
Color intensity                 0
Hue                             0
OD280/OD315 of diluted wines    0
Proline                         0
dtype: int64
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Class                         178 non-null    int64
 1   Alcohol                       178 non-null    float64
 2   Malic acid                    178 non-null    float64
 3   Ash                           178 non-null    float64
 4   Alcalinity of ash             178 non-null    float64
 5   Magnesium                     178 non-null    int64
 6   Total phenols                 178 non-null    float64
 7   Flavanoids                    178 non-null    float64
 8   Nonflavanoid phenols          178 non-null    float64
 9   Proanthocyanins               178 non-null    float64
 10  Color intensity               178 non-null    float64
 11  Hue                           178 non-null    float64
 12  OD280/OD315 of diluted wines  178 non-null    float64
 13  Proline                       178 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

```python
# display histogram
df.hist(bins=25,figsize=(10,10))

plt.show()
```



IMPORT DATA | EDA | PCA | K-MEANS | GMM | HIERACHICAL CLUSTERING | EVALUATION

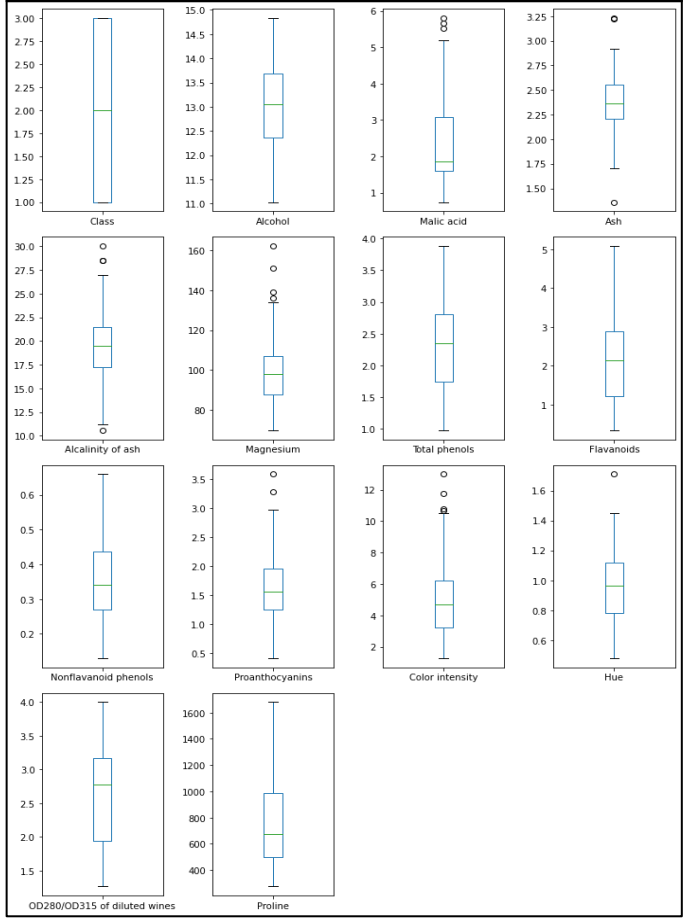
```
df.describe()
```

Python

	Class	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449	2.611685	746.893258
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572	0.709990	314.907474
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000	1.270000	278.000000
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500	1.937500	500.500000
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000	2.780000	673.500000
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000	3.170000	985.000000
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000

IMPORT DATA | EDA | PCA | K-MEANS | GMM | HIERACHICAL CLUSTERING | EVALUATION

```
plt.figure(figsize=(10,15))
for x,y in enumerate(list(df.columns.values)):
  plt.subplot(4,4,x+1)
  df.boxplot(y)
  plt.grid()
  plt.tight_layout()
```



DATA PREPROCESSING

DROPPING CLASS

```
# Dropping Class column from data
X = pd.DataFrame(df.drop(['Class'], axis = 1))
ylabel = df['Class']
```

SCALING DATA

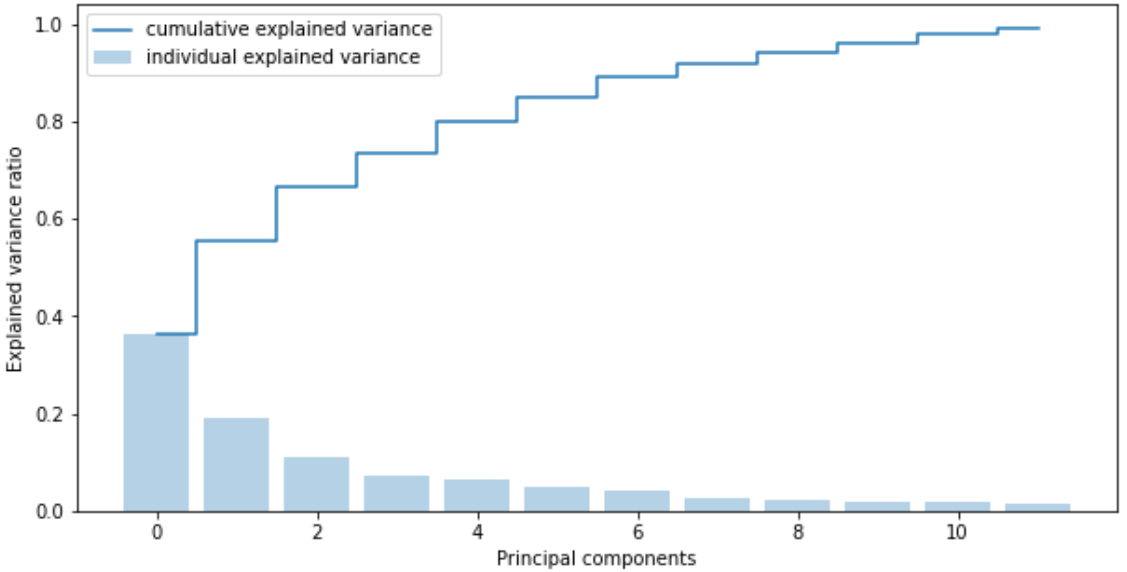
```
# Scaling the data
std_scale = preprocessing.StandardScaler().fit(X)
df_std = std_scale.transform(X)
data = pd.DataFrame(df_std, columns = X.columns)
data.head()
```

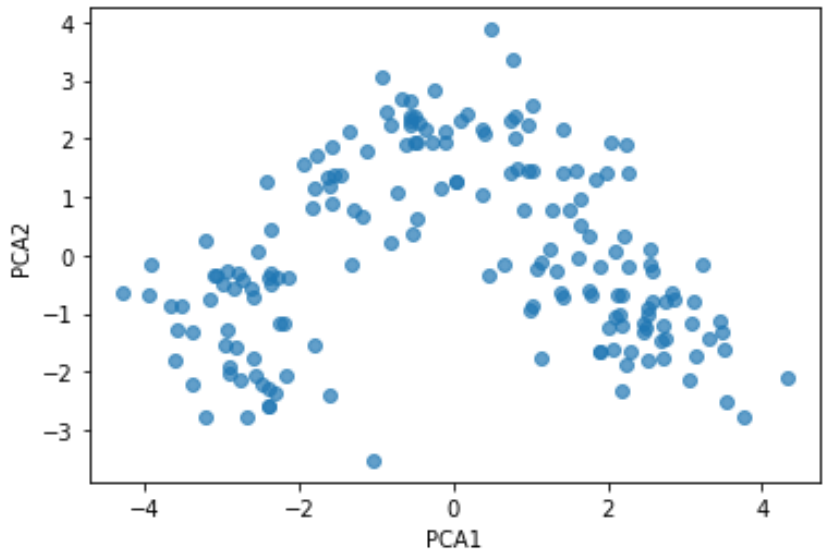
	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols
0	1.518613	-0.562250	0.232053	-1.169593	1.913905	0.808997	1.034819	-0.659563
1	0.246290	-0.499413	-0.827996	-2.490847	0.018145	0.568648	0.733629	-0.820719
2	0.196879	0.021231	1.109334	-0.268738	0.088358	0.808997	1.215533	-0.498407
3	1.691550	-0.346811	0.487926	-0.809251	0.930918	2.491446	1.466525	-0.981875
4	0.295700	0.227694	1.840403	0.451946	1.281985	0.808997	0.663351	0.226796

Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
1.224884	0.251717	0.362177	1.847920	1.013009
-0.544721	-0.293321	0.406051	1.113449	0.965242
2.135968	0.269020	0.318304	0.788587	1.395148
1.032155	1.186068	-0.427544	1.184071	2.334574
0.401404	-0.319276	0.362177	0.449601	-0.037874

```
    pca = PCA(n_components=12, random_state=42)
    pca_result = pca.fit_transform(data)
    print('Explained variation per component: {}\n'.format(pca.explained_variance_ratio_))

    print('Cumulative explained variance of the components: {}\n'.format(pca.explained_variance_ratio_.cumsum()))

    print("Shape of pca_result:")
    print(pca_result.shape)
```

```
Explained variation per component: [0.36198848 0.1920749  0.11123631 0.0706903  0.06563294 0.04935823
 0.04238679 0.02680749 0.02222153 0.01930019 0.01736836 0.01298233]

Cumulative explained variance of the components: [0.36198848 0.55406338 0.66529969 0.73598999 0.80162293 0.85098116
 0.89336795 0.92017544 0.94239698 0.96169717 0.97906553 0.99204785]

Shape of pca_result:
(178, 12)
```

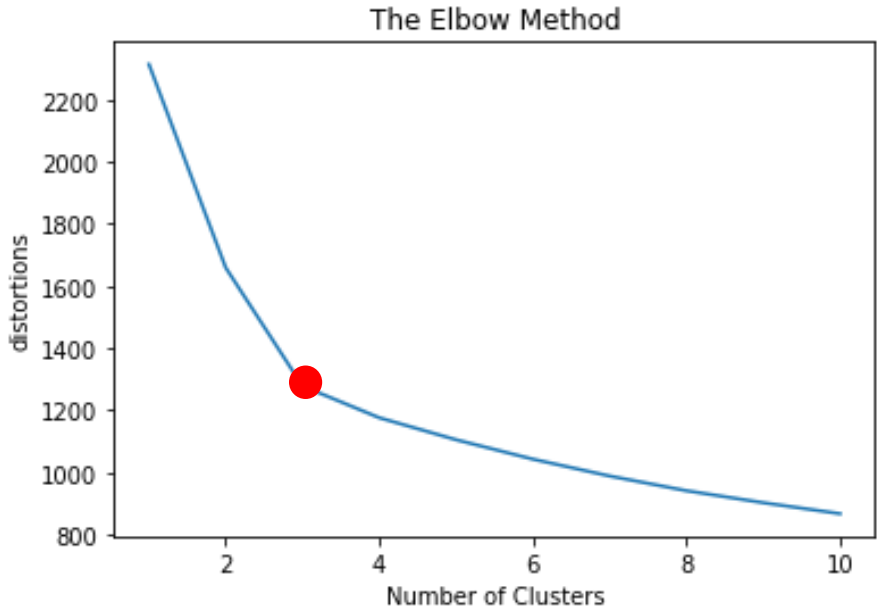





```
        Alcohol  Malic acid       Ash  Alcalinity of ash  Magnesium  \
PC_1   0.144329    0.245188  0.002051           0.239320   0.141992
PC_2   0.483652    0.224931  0.316069           0.010591   0.299634

      Total phenols  Flavanoids  Nonflavanoid phenols  Proanthocyanins  \
PC_1       0.394661    0.422934              0.298533         0.313429
PC_2       0.065040    0.003360              0.028779         0.039302

      Color intensity       Hue  OD280/OD315 of diluted wines   Proline
PC_1         0.088617  0.296715                      0.376167  0.286752
PC_2         0.529996  0.279235                      0.164496  0.364903

*************** Most important features *************************
As per PC 1:
 Total phenols                 0.394661
Flavanoids                    0.422934
Proanthocyanins               0.313429
OD280/OD315 of diluted wines  0.376167
Name: PC_1, dtype: float64


As per PC 2:
 Alcohol             0.483652
Ash                 0.316069
Color intensity     0.529996
Proline             0.364903
Name: PC_2, dtype: float64

*****************************************************************
```

RESULTS FROM PCA

SILHOUETTE SCORE




```
For n_clusters=2, The Silhouette Coefficient is 0.26831340971052126
For n_clusters=3, The Silhouette Coefficient is 0.2848589191898987
For n_clusters=4, The Silhouette Coefficient is 0.25173343011696475
For n_clusters=5, The Silhouette Coefficient is 0.2271732547624458
For n_clusters=6, The Silhouette Coefficient is 0.19582485390848947
For n_clusters=7, The Silhouette Coefficient is 0.20913005310687274
For n_clusters=8, The Silhouette Coefficient is 0.13581656516941268
For n_clusters=9, The Silhouette Coefficient is 0.14576057110571292
For n_clusters=10, The Silhouette Coefficient is 0.13394527355239233
For n_clusters=11, The Silhouette Coefficient is 0.1415050320596543
```

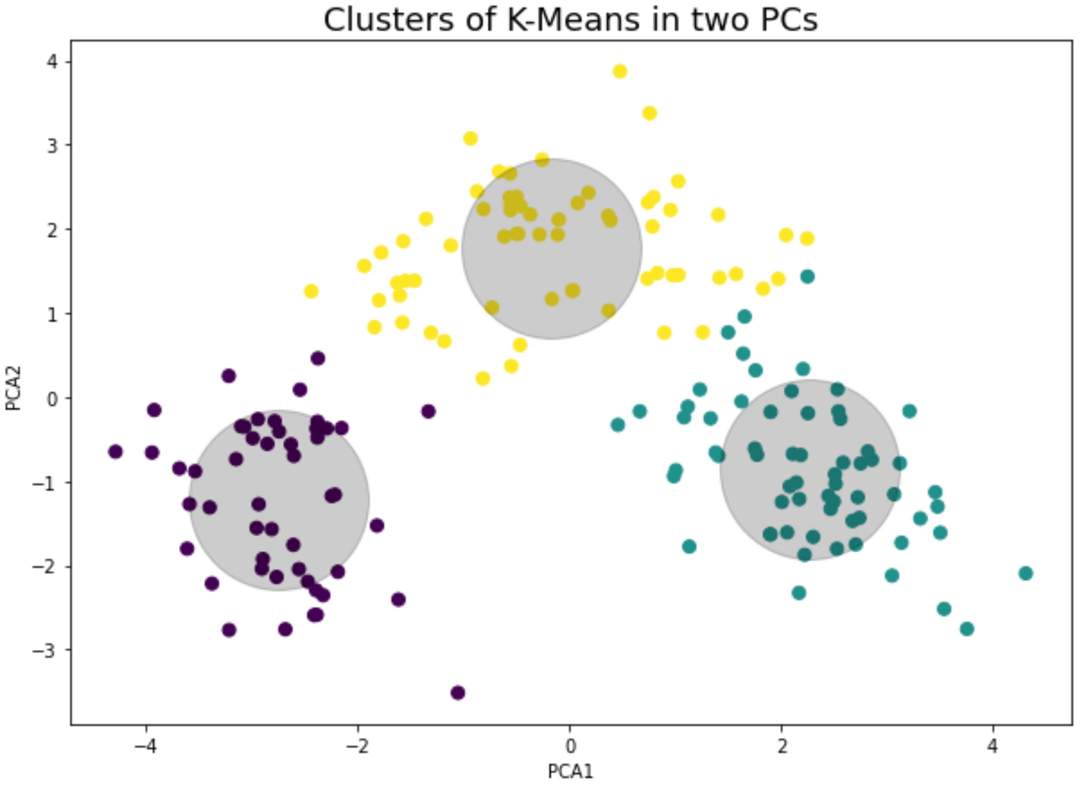
MOST OPTIMAL SCORE IS 3

KMEANS

```
Labels predicted by K-Means: [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 1 0 0 0 0 0 0 0 0 0 0 2
 0 0 0 0 0 0 0 0 0 1 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
 0 0 0 0 0 0 0 1 0 0 2 0 0 0 0 0 0 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Shape of centroids_KM (3, 13)
Centroids: [[-0.92607185 -0.39404154 -0.49451676  0.17060184 -0.49171185 -0.07598265
   0.02081257 -0.03353357  0.0582655  -0.90191402  0.46180361  0.27076419
  -0.75384618]
 [ 0.16490746  0.87154706  0.18689833  0.52436746 -0.07547277 -0.97933029
  -1.21524764  0.72606354 -0.77970639  0.94153874 -1.16478865 -1.29241163
  -0.40708796]
 [ 0.83523208 -0.30380968  0.36470604 -0.61019129  0.5775868   0.88523736
   0.97781956 -0.56208965  0.58028658  0.17106348  0.47398365  0.77924711
   1.12518529]]
Inertia: 1277.9284888446423
```

KMEANS + PCA

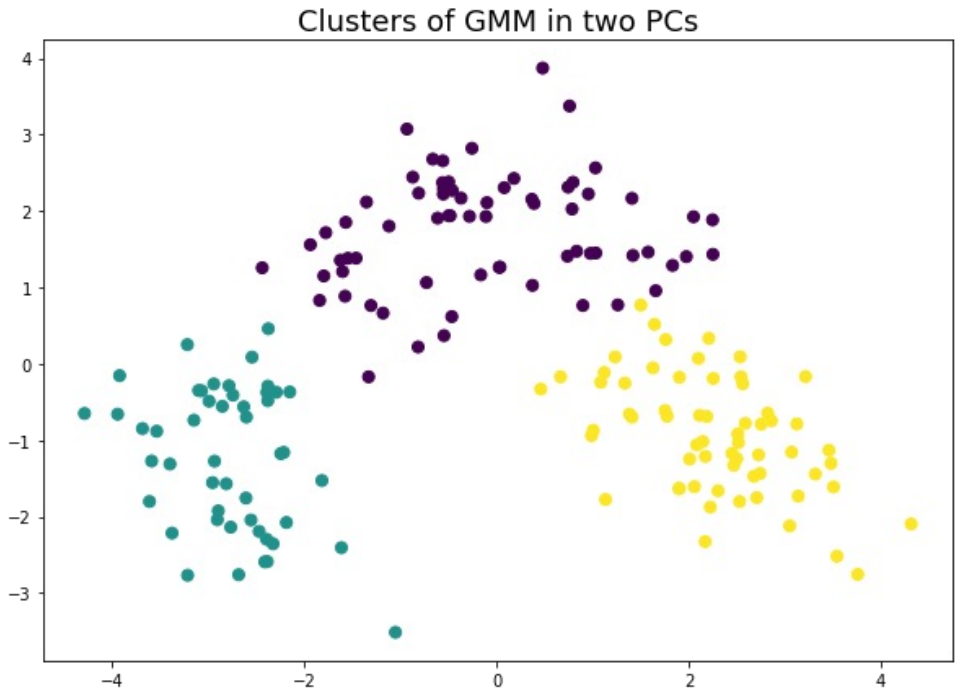
```
Labels predicted by KM+PCA: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1
 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Shape of centroids_KM (3, 2)
Centroids: [[-2.74392982 -1.2141906 ]
 [ 2.26614991 -0.86559213]
 [-0.16278513  1.76758824]]
Inertia: 259.509380751193
```



IMPORT DATA | EDA | PCA | K-MEANS | GMM | HIERACHICAL CLUSTERING | EVALUATION

GAUSSIAN MIXTURE MODEL

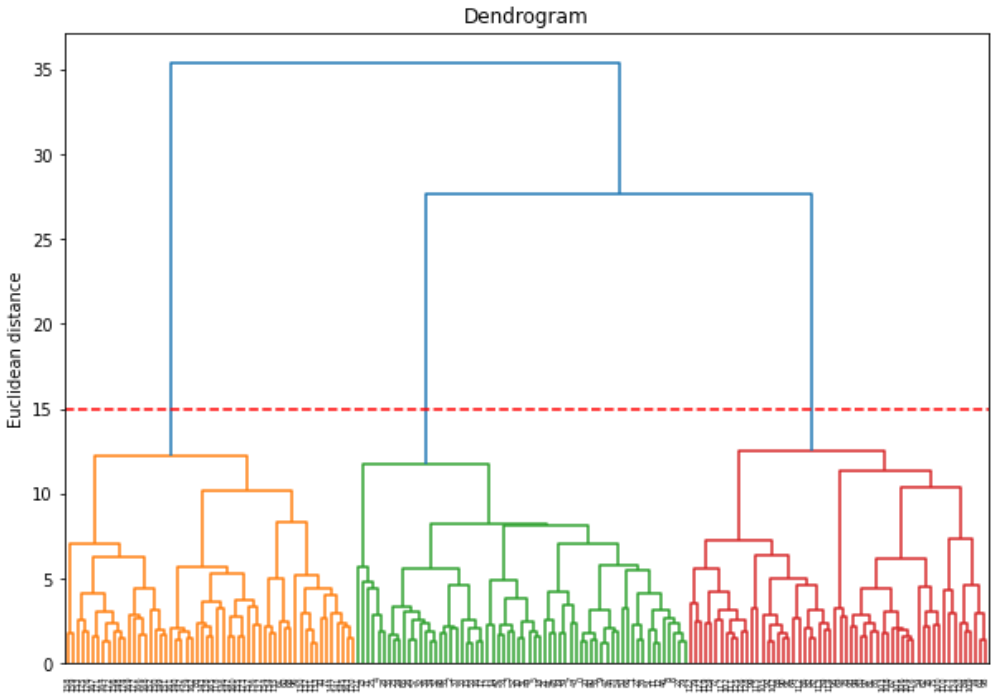
```
Weights: [0.3580442  0.35563555 0.28632025]
Means: [[-0.94194964 -0.36665974 -0.39337021  0.25121283 -0.57289459 -0.03349368
   0.08136144  0.01323491  0.00779434 -0.88476972  0.43434413  0.29281555
  -0.78131234]
 [ 0.81535193 -0.33244352  0.2448001  -0.67576135  0.63685313  0.82213961
   0.89675273 -0.59782121  0.61971204  0.13188764  0.50056412  0.74602428
   1.11395195]
 [ 0.16516987  0.87143373  0.18784666  0.52521419 -0.07462282 -0.97928753
  -1.21559036  0.72599751 -0.77948503  0.94259045 -1.16489416 -1.29279595
  -0.4065956 ]]
Type of covariance_GMM: <class 'numpy.ndarray'>
Labels predicted by GMM: [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 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 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 1 0 0 0 1
 0 0 0 0 1 0 0 0 0 2 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 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
Length of labels is same as data entry (178,)
```



GAUSSIAN MIXTURE MODEL + PCA

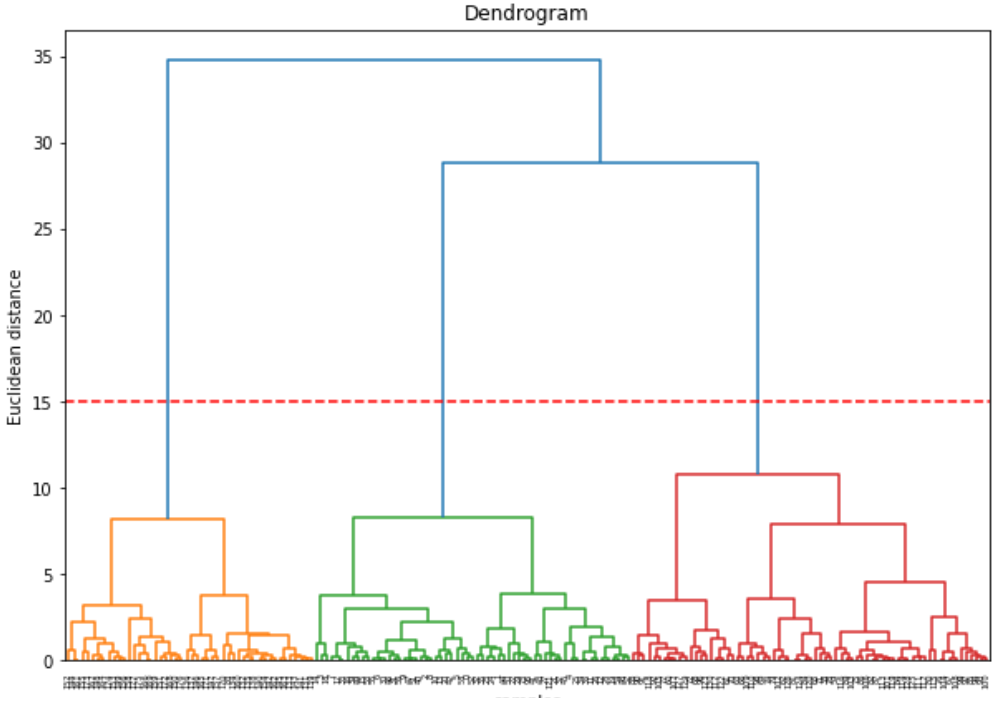
```
Weights: [0.38068048 0.26913256 0.35018695]
Means: [[-0.13128107  1.71090924]
 [-2.76625789 -1.23789567]
 [ 2.26869167 -0.90851963]]
Type of covariance_GMM: <class 'numpy.ndarray'>
Labels predicted by GMM + PCA: [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2
 0 0 0 0 0 0 0 0 0 1 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 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Length of labels is same as data entry (178,)
```

HIERACHICAL CLUSTERING




```
Labels predicted by Clustering model:
 [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 1 1 0 0 0 2 2 0 1 0 1 2 0 2
 0 0 0 1 0 0 0 0 0 1 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
 0 0 0 0 0 0 0 1 0 0 2 0 0 0 0 0 0 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Length of labels is same as data entry (178,)
```

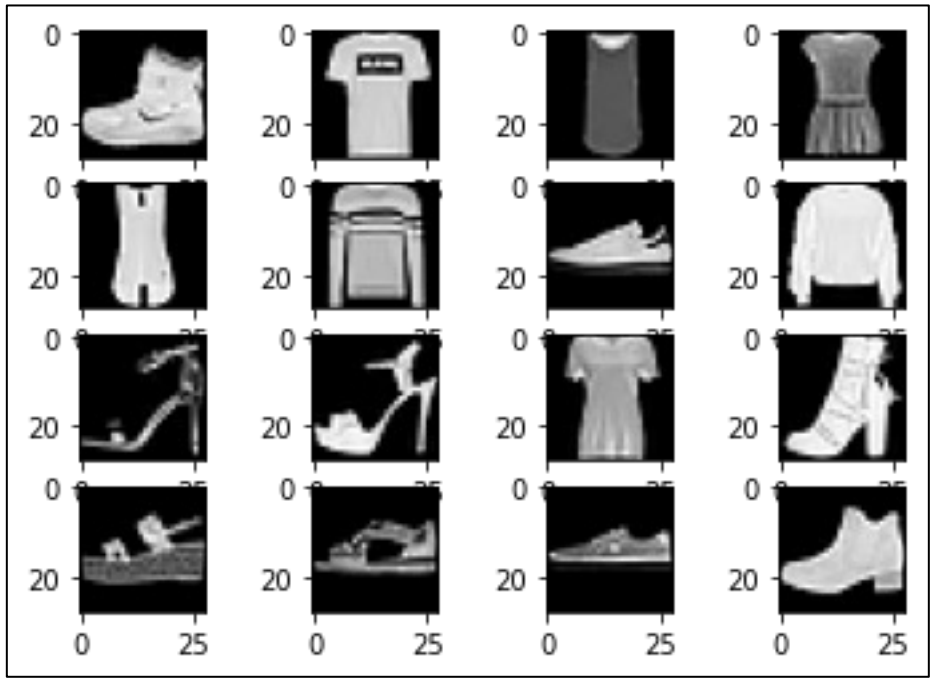
HIERACHICAL CLUSTERING + PCA




```
Labels predicted by Clustering + PCA model:
 [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2
 0 0 0 0 0 0 0 0 0 1 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
 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Length of labels is same as data entry (178,)
```

MODELS	SILHOUETTE SCORES	ADJUSTED RAND INDEX
KMEANS	0.2849	0.8975
KMEANS + PCA	0.2831	0.8951
GMM	0.2844	0.8804
GMM + PCA	0.2829	0.9135
HC	0.2744	0.7899
HC + PCA	0.2829	0.8960

```
# Plot mnist instances
from tensorflow.keras.datasets import fashion_mnist
import matplotlib.pyplot as plt
# load (downloaded if needed) the MNIST dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

for i in range(16):
    plt.subplot(4,4,i+1)
    plt.imshow(X_train[i], cmap=plt.get_cmap('gray'))
plt.show()

# Create a dictionary for each type of label
labels = {0 : "T-shirt/top", 1: "Trouser", 2: "Pullover", 3: "Dress", 4: "Coat",
          5: "Sandal", 6: "Shirt", 7: "Sneaker", 8: "Bag", 9: "Ankle Boot"}
```



```
# Identifying the split ratio
print('X_train: ' + str(X_train.shape))
print('Y_train: ' + str(y_train.shape))
print('X_test:  '  + str(X_test.shape))
print('Y_test:  '  + str(y_test.shape))
```

```
X_train: (60000, 28, 28)
Y_train: (60000,)
X_test:  (10000, 28, 28)
Y_test:  (10000,)
```

IMPORT DATA | FEATURES ENGINEERING | SIMPLE MODEL | CNN MODEL | HYPERPARAMETER TUNING | EVALUATION | MAKING PREDICTIONS

1. FLATTEN IMAGES ->

```
# flatten 28*28 images to a 784 vector for each image
X_train = X_train.reshape((X_train.shape[0], 28*28)).astype('float32')
X_test = X_test.reshape((X_test.shape[0], 28*28)).astype('float32')
```

2. NORMALIZE INPUTS ->

```
# normalize inputs from 0-255 to 0-1 (min-max scaling)
X_train = X_train / 255
X_test = X_test / 255
print('X_train: ' + str(X_train.shape))
print('X_test:  '  + str(X_test.shape))
```

3. ENCODE OUTPUTS ->

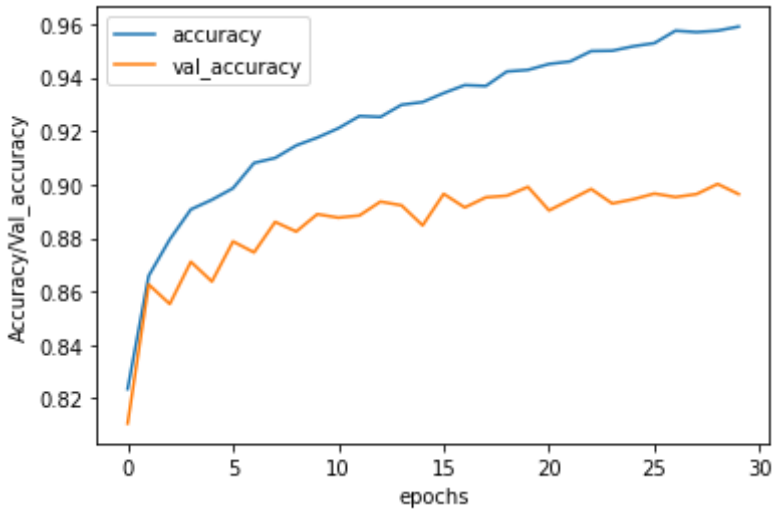
```
# one hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
num_classes = y_test.shape[1]
```

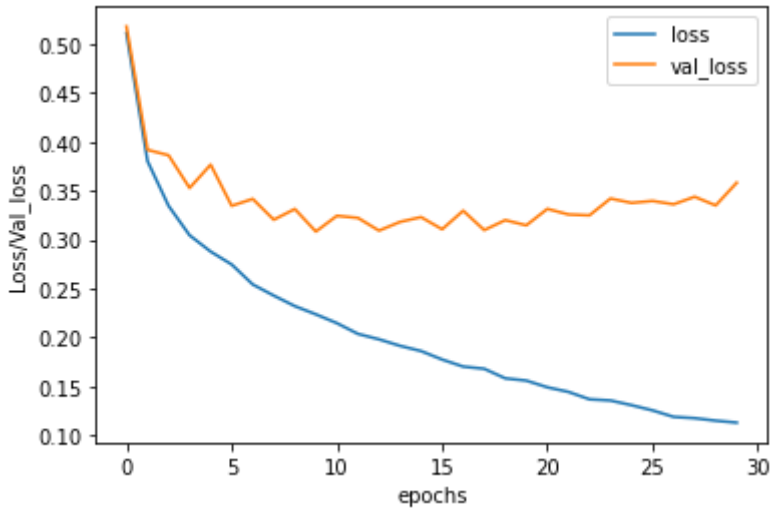
```
# create model
model = Sequential()

# add the first hidden layer
model.add(Dense(784, input_shape=(784,),
                kernel_initializer='normal', activation='relu'))

# add the output layer
model.add(Dense(num_classes,
                kernel_initializer='normal', activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
```

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_8 (Dense)              (None, 784)               615440

 dense_9 (Dense)              (None, 10)                7850

=================================================================
Total params: 623,290
Trainable params: 623,290
Non-trainable params: 0
_________________________________________________________________
```

FIT THE MODEL

```
# Fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
          epochs=30, batch_size=200, verbose=1)
```





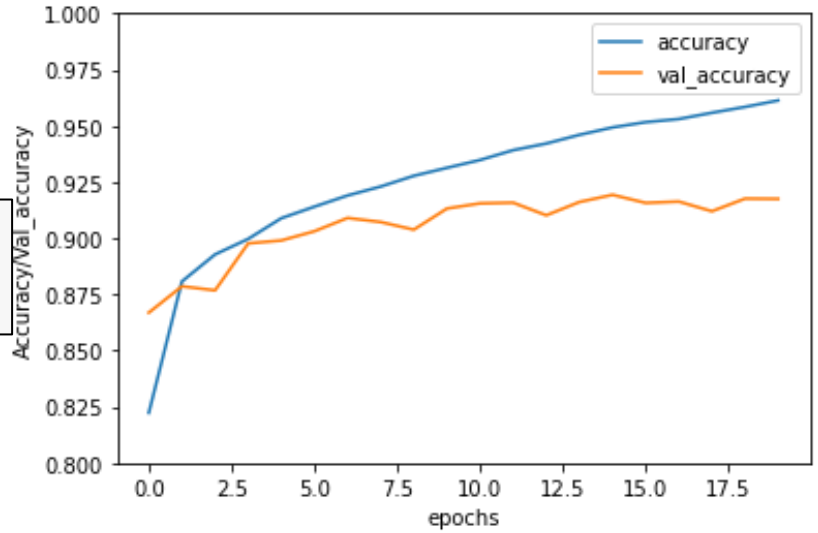
ACCURACY: 89.64% | BASELINE ERROR: 10.36%

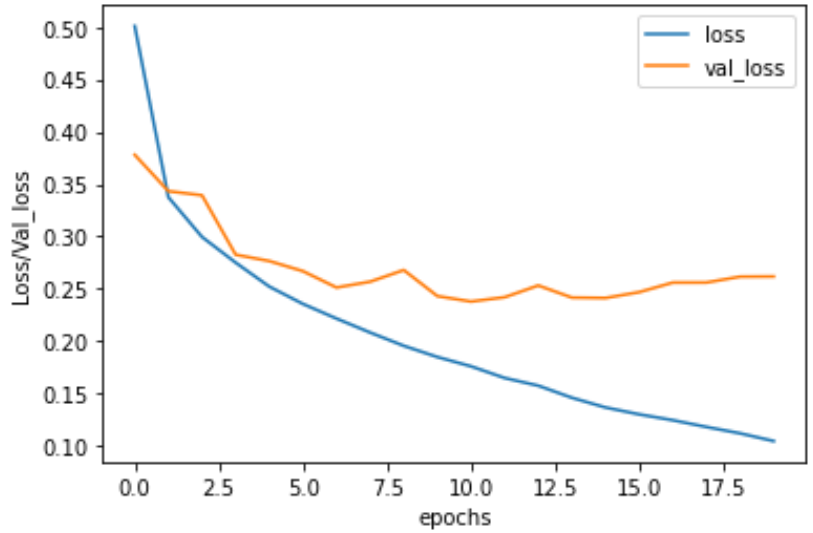
```python
# Create model
model = Sequential()
model.add(Conv2D(32, (5, 5), input_shape=(28, 28, 1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy',
    optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 24, 24, 32)        832

 max_pooling2d_5 (MaxPooling  (None, 12, 12, 32)       0
 2D)

 dropout_9 (Dropout)         (None, 12, 12, 32)        0

 flatten_3 (Flatten)         (None, 4608)              0

 dense_10 (Dense)            (None, 128)               589952

 dense_11 (Dense)            (None, 10)                1290

=================================================================
Total params: 592,074
Trainable params: 592,074
Non-trainable params: 0
_________________________________________________________________
None
```

FIT THE MODEL

```
# Fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
          epochs=20, batch_size=200, verbose=1)
```





ACCURACY: 91.75% | BASELINE ERROR: 8.25%

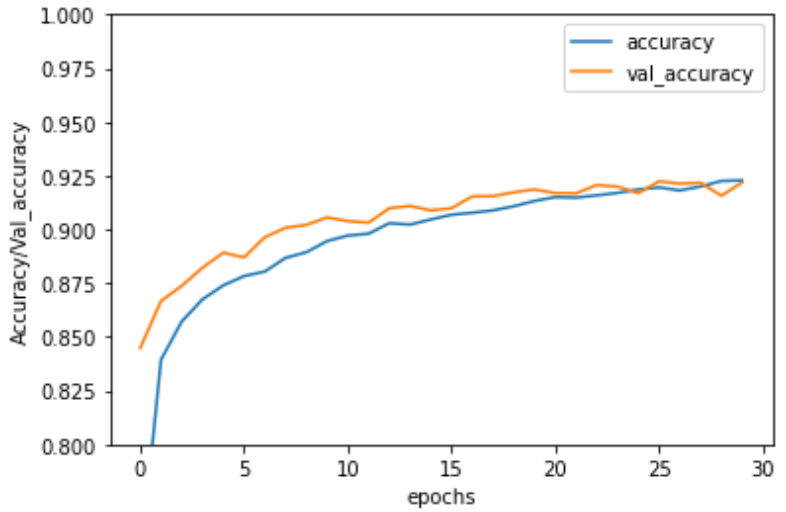
NEW CNN MODEL (AFTER ADDING NEW LAYERS)

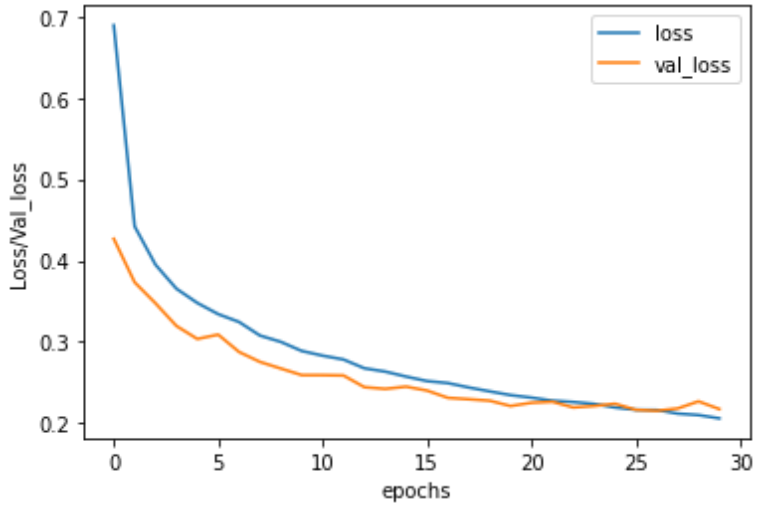
```
# Adding more layers to the CNN model
model = Sequential()
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=2, padding='same', activation='relu', input_shape=(28,28,1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

```
# Compile model
model.compile(loss='categorical_crossentropy',
    optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 28, 28, 64)        320

 max_pooling2d_16 (MaxPoolin  (None, 14, 14, 64)       0
 g2D)

 dropout_27 (Dropout)        (None, 14, 14, 64)        0

 conv2d_21 (Conv2D)          (None, 14, 14, 32)        8224

 max_pooling2d_17 (MaxPoolin  (None, 7, 7, 32)         0
 g2D)

 dropout_28 (Dropout)        (None, 7, 7, 32)          0

 flatten_9 (Flatten)         (None, 1568)              0

 dense_22 (Dense)            (None, 256)               401664

 dropout_29 (Dropout)        (None, 256)               0

 dense_23 (Dense)            (None, 10)                2570

=================================================================
Total params: 412,778
Trainable params: 412,778
Non-trainable params: 0
```

```
# Fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
          epochs=30, batch_size=200, verbose=1)
```





ACCURACY: 92.19% | BASELINE ERROR: 7.81%

IMPORT DATA | FEATURES ENGINEERING | SIMPLE MODEL | **CNN MODEL** | HYPERPARAMETER TUNING | EVALUATION | MAKING PREDICTIONS

```
# Defining hyperparameter values
params={'batch_size':[100, 200],
        'epochs':[10, 20, 30]
        } ##

new_model = KerasClassifier(build_fn=model)

models = GridSearchCV(estimator = new_model, param_grid=params, cv=3)
```

```
best_model = models.fit(X_train, y_train)
print('Best model :')
print(best_model.best_params_)
```

```
Epoch 16/20
300/300 [==============================] - 3s 10ms/step - loss: 0.1795 - accuracy: 0.9311
Epoch 17/20
300/300 [==============================] - 3s 9ms/step - loss: 0.1768 - accuracy: 0.9329
Epoch 18/20
300/300 [==============================] - 3s 9ms/step - loss: 0.1795 - accuracy: 0.9325
Epoch 19/20
300/300 [==============================] - 3s 9ms/step - loss: 0.1774 - accuracy: 0.9331
Epoch 20/20
300/300 [==============================] - 3s 10ms/step - loss: 0.1752 - accuracy: 0.9341
Best model :
{'batch_size': 200, 'epochs': 20}
```

MOST OPTIMAL:

- BATCH SIZE: 200
- EPOCHS: 20

BUILDING A MODEL BASED ON HYPERPARAMETER RESULTS

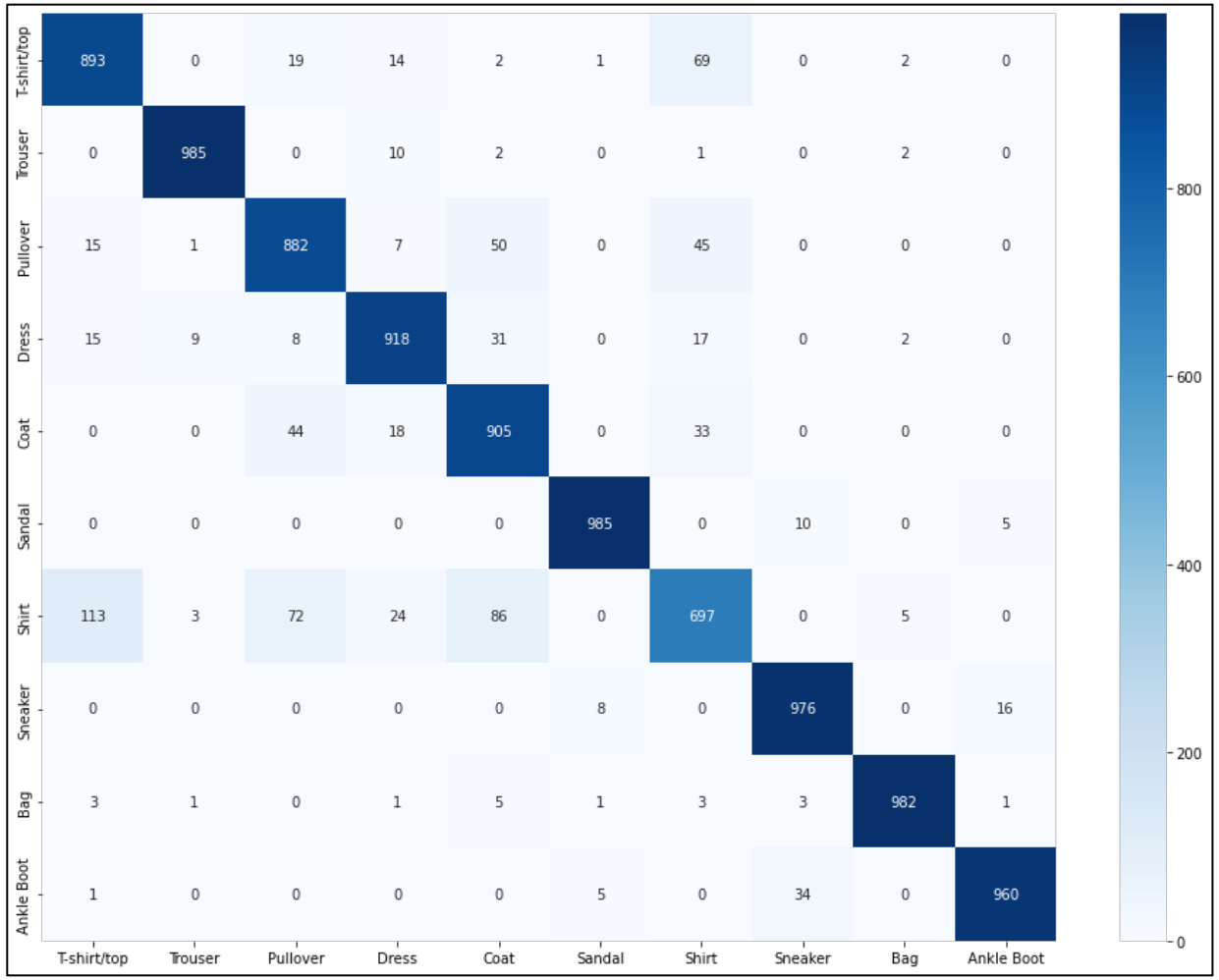
```
model = Sequential()
# Add convolution 2D
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=2, padding='same', activation='relu', input_shape=(28,28,1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy',
    optimizer='adam', metrics=['accuracy'])

history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
          epochs=20, batch_size=200, verbose=1)
```

ACCURACY: 91.83% | **BASELINE ERROR: 8.17%**

CONFUSION MATRIX

```
from sklearn.metrics import confusion_matrix
categories = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle Boot"]

y_pred=model.predict(X_test)
y_pred1=np.argmax(y_pred, axis=1)
y_test1=np.argmax(y_test, axis=1)
cm = confusion_matrix(y_test1, y_pred1)
```

```
cf_matrix = pd.DataFrame(cm, index= ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle Boot"],
                         columns=["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle Boot"])

f, ax = plt.subplots(figsize=(16, 12))
sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='g')
```



IMPORT DATA | FEATURES ENGINEERING | SIMPLE MODEL | CNN MODEL | HYPERPARAMETER TUNING | EVALUATION | MAKING PREDICTIONS

PREDICTION RESULTS ON TEST DATASET

```
p = y_pred1
y = y_test1
correct = np.nonzero(p==y)[0]
incorrect = np.nonzero(p!=y)[0]
```

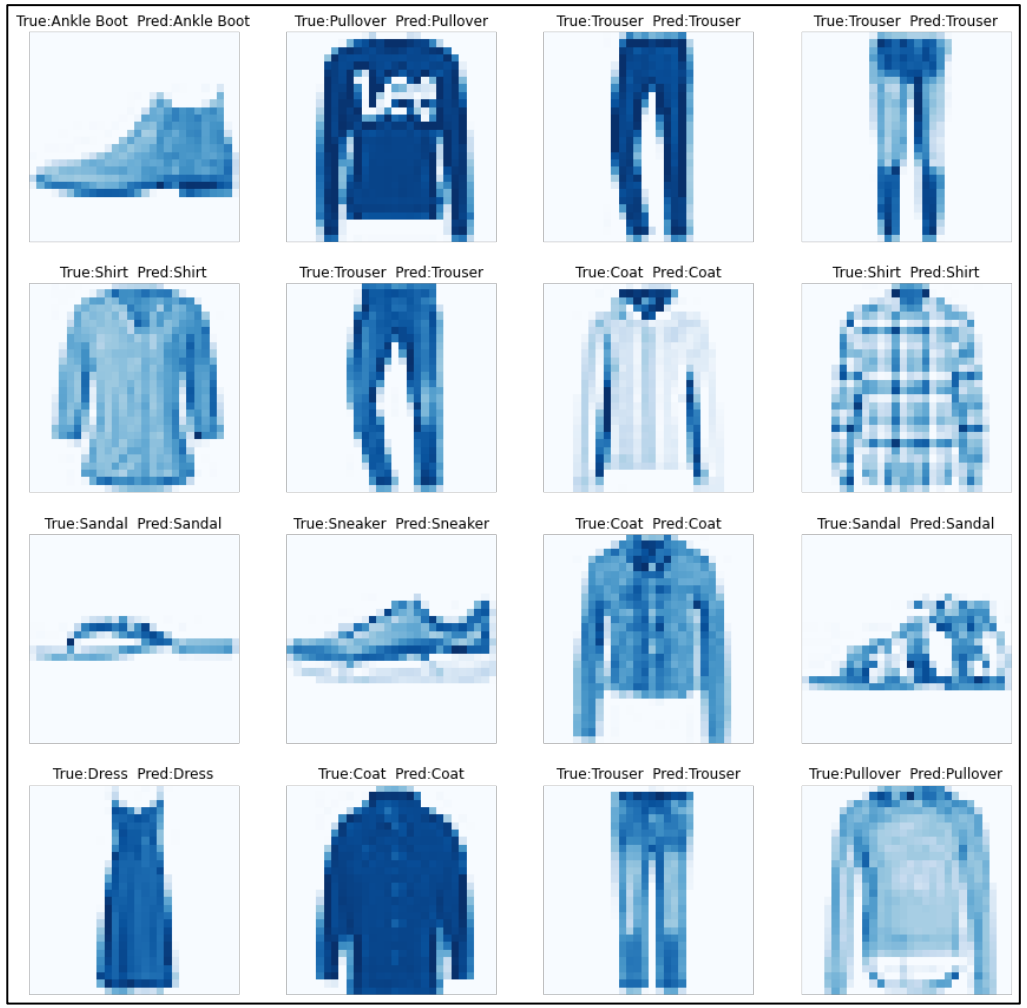
```
print("Correctly predicted classes:",correct.shape[0])
print("Incorrectly predicted classes:",incorrect.shape[0])
```

```
Correctly predicted classes: 9183
Incorrectly predicted classes: 817
```

```
target_names = ["Class {} ({}) :".format(i,labels[i]) for i in range(num_classes)]
print(classification_report(y_test1, y_pred1, target_names=target_names))
```

```
                      precision    recall  f1-score   support

Class 0 (T-shirt/top) :       0.86      0.89      0.88      1000
    Class 1 (Trouser) :       0.99      0.98      0.99      1000
   Class 2 (Pullover) :       0.86      0.88      0.87      1000
      Class 3 (Dress) :       0.93      0.92      0.92      1000
       Class 4 (Coat) :       0.84      0.91      0.87      1000
     Class 5 (Sandal) :       0.98      0.98      0.98      1000
      Class 6 (Shirt) :       0.81      0.70      0.75      1000
    Class 7 (Sneaker) :       0.95      0.98      0.96      1000
        Class 8 (Bag) :       0.99      0.98      0.99      1000
 Class 9 (Ankle Boot) :       0.98      0.96      0.97      1000

               accuracy                           0.92     10000
              macro avg       0.92      0.92      0.92     10000
           weighted avg       0.92      0.92      0.92     10000
```

PART B: DEEP LEARNING



CORRECTLY CLASSIFIED IMAGES

INCORRECTLY CLASSIFIED IMAGES

IMPORT DATA | FEATURES ENGINEERING | SIMPLE MODEL | CNN MODEL | HYPERPARAMETER TUNING | EVALUATION | MAKING PREDICTIONS