Report

(a) Frame the problem and look at the big picture:

Customer Personality Analysis is analysing the company's customer.

- It helps the company to better understand their customer and modify the products according to the needs of the customers.
- It helps the company to target particular segment of customers.
- It helps the company to increase their revenues by identifying customer behaviour.

Input Attributes

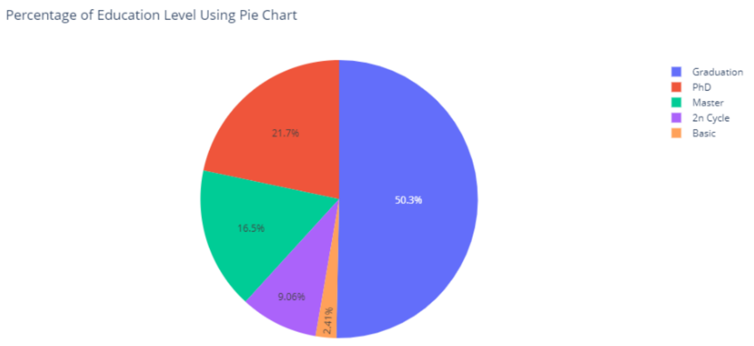
- ID: Customer's unique identifier
- Year_Birth: Customer's birth year
- Education: Customer's education level
- Marital_Status: Customer's marital status
- Income: Customer's yearly household income
- Kidhome: Number of children in customer's household
- Teenhome: Number of teenagers in customer's household
- Dt_Customer: Date of customer's enrollment with the company
- Recency: Number of days since customer's last purchase
- Complain: 1 if customer complained in the last 2 years, 0 otherwise

Other attributes are for Products, Promotion and Place.

Target - Perform Clustering to create customer segments.

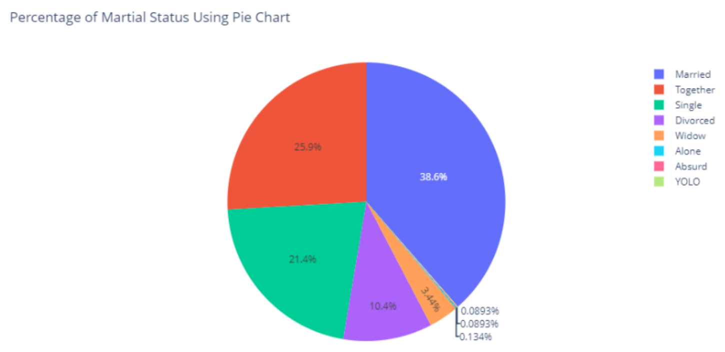
(b) Show 4+ graphs to explore the data and gain insights:

1. Percentage of Education Level using Pie Chart



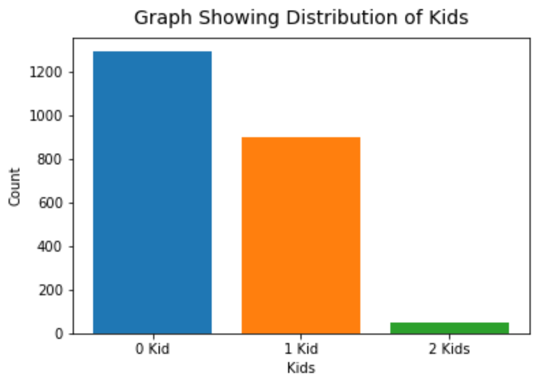
It can be inferred that more than 50 % of customers are graduated.

2. Percentage of Martial Status using Pie Chart



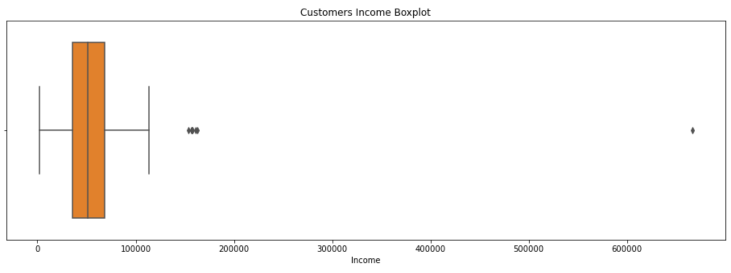
It can be inferred that around 38 % customers are married and 21.4 % of customers are single.

3. Graph Showing Distribution of kids

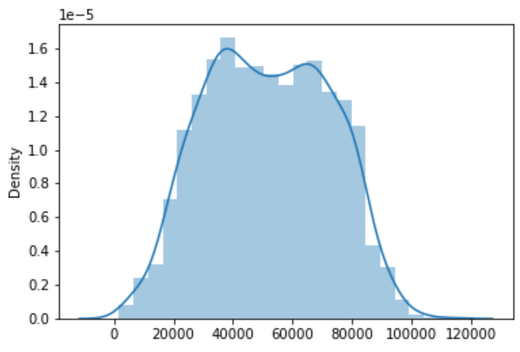


It shows the number of children in customer house. It can be inferred that most of the customers have either 0 or 1 kid and very few customers have 2 kids.

4. Box Plot to find Outliers in Customer's Yearly Income



5. Customer's Yearly Income Distribution



It can be inferred that maximum customer's income lies between 25000 to 65000.

(c) Prepare the data to better expose the underlying data patterns to machine learning algorithms:

Data Cleaning

1. Removed ID column.
2. Removed outliers in income, i.e., income greater than 150000.
3. Found 24 null values in income column. I used median imputation.
4. Removed 182 duplicated rows.

Data Preparation

- Created Age column from birth year
- Created CustomerFor Column from Date of customer's enrollment with the company
- Removed columns: Year_Birth & Dt_Customer
- Merged Education Types as Bachelor, Master, PhD and Basic
- Merged Martial Status (YOLO, Absurd & Alone to Single)
- Created HasPartner Column
- Combined Children and teens to create NumChildren
- Created HasChildren, MntTotal, NumTotalPurchases column
- Deleted rows where total amount is non-zero and purchase is zero.
- Created average check, AcceptedTotal column
- Removed Outliers in Age and AvgCheck
- Used One hot encoding on Education and Martial_Status
- Dropped Non-Useful features and Applied Standardization.

(d) Use three machine learning algorithms (one of which must be a deep neural network), discuss their performance, and show a comparative performance table:

Model 1: K-means Clustering

The K-Means algorithm is a simple algorithm capable of clustering this kind of dataset very quickly and efficiently, often in just a few iterations.

Algorithm –

1. Select the number K to decide the number of clusters.
2. Select random K points or centroids. (It can be other from the input dataset).
3. Assign each data point to their closest centroid, which will form the predefined K clusters.
4. Calculate the variance and place a new centroid of each cluster.
5. Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
6. If any reassignment occurs, then go to step-4 else go to FINISH.

It gives 3 clusters with_Silhouette score as 0.485.

Model 2: Hierarchical or Agglomerative Clustering

Agglomerative Clustering is a type of hierarchical clustering algorithm. It is an unsupervised machine learning technique that divides the population into several clusters such that data points in the same cluster are more similar and data points in different clusters are dissimilar.

Agglomerative — Bottom-up approach. Start with many small clusters and merge them together to create bigger clusters.

Divisive — Top down approach. Start with a single cluster than break it up into smaller clusters.

It gives 3 clusters with Silhouette score as 0.424.

Model 3: Deep Neural Network

As this is an unsupervised clustering task, so a direct deep neural network is hard to construct. So I have converted this to a classification task with 3 classes, where classes can be interpreted as number of clusters and for the labels I have used the clusters predicted by k-means algorithm.

Loss Function – Sparse categorical Crossentropy
Number of Epochs – 20
Number of neurons in output layer – 3
Optimizer – Adam
Number of hidden layers - 2

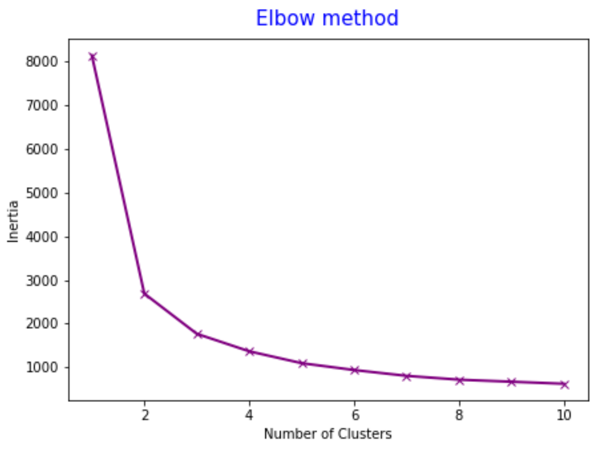
Performance –

The K means performed better as compared to Hierarchical or Agglomerative Clustering. But if we see in general 0.48 seems low scores for clustering task.

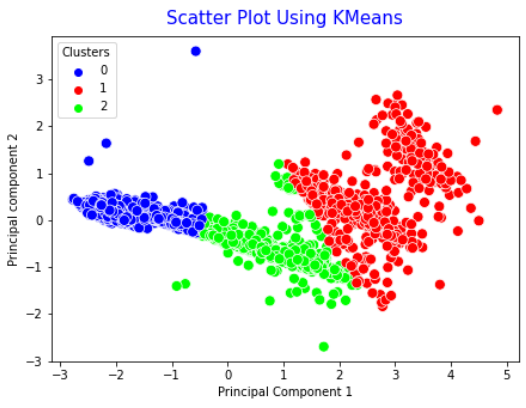
Table 1: Performance Comparison of Models

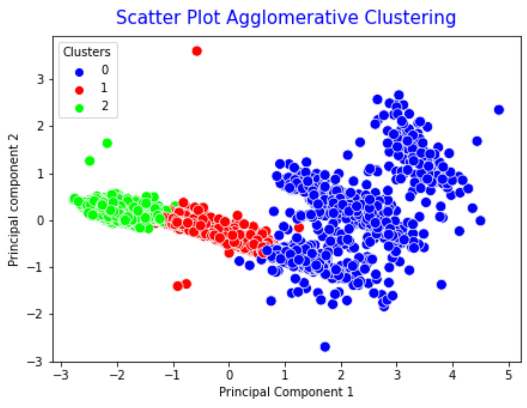
Model	Optimal Number of Cluster	Silhouette Score
K-means Clustering	3	0.48558
Hierarchical or Agglomerative Clustering	3	0.42462
Deep Neural Network	-	0.0317 (Loss)

(e) For the best performing algorithm, show 4+ graphs:

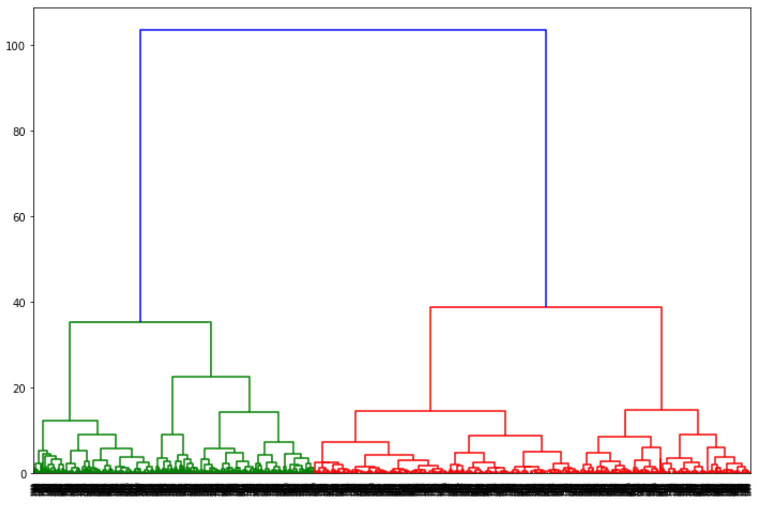


It is used to find optimal number of clusters. Here, from the bend we can see optimal clusters are 3.





Dendrogram using Hierarchical or Agglomerative Clustering



Appendix

```
#!/usr/bin/env python
# coding: utf-8

# ### Importing libraries

# In[1]:



import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from sklearn.cluster import AgglomerativeClustering
import warnings
warnings.filterwarnings('ignore')



# ### Loading The Dataset

# In[2]:
```

```
datasetPath = "C:\\Users\\Desktop\\"

imagePath = "C:\\Users\\Desktop\\Graphs\\"

dataFrame = pd.read_csv(datasetPath + 'marketing_campaign (1).csv', sep = '\t', parse_dates
= ['Dt_Customer'])


print("Shape of Dataset: ", dataFrame.shape)

dataFrame.head(5)



# ### B. Data Exploration & Gaining Insights


# #### B.1 Percentage of Education Level Using Pie Chart


# In[3]:



#Listing all values from education column

educationNamesList = ['2n Cycle', 'Basic', 'Graduation', 'Master', 'PhD']

px.pie(values = dataFrame.groupby('Education').size(), names = educationNamesList, title =
'Percentage of Education Level Using Pie Chart')



# #### B.2 Percentage of Marital Status Using Pie Chart


# In[4]:



maritalStatusNamesList = ['Absurd', 'Alone', 'Divorced', 'Married', 'Single', 'Together',
'Widow', 'YOLO']

px.pie(values = dataFrame.groupby('Marital_Status').size(), names = maritalStatusNamesList,
title = 'Percentage of Martial Status Using Pie Chart')
```

```
# #### B.3 Visualization number of children in customer's household

# In[5]:



count = dataFrame.Kidhome.value_counts()
#print(count) #0 - 1293, 1 - 899, 2 - 48
zeroKidCount = count[0]
oneKidCount = count[1]
twoKidCount = count[2]

#Graph Showing Distribution of Kids
plt.figure(figsize = (6,4))
plt.title("Graph Showing Distribution of Kids", size = 14, color = 'black', pad = 10)
plt.bar('0 Kid', zeroKidCount)
plt.bar('1 Kid', oneKidCount)
plt.bar('2 Kids', twoKidCount)
plt.xlabel("Kids")
plt.ylabel("Count")
plt.savefig(imagePath + "Graph Showing Distribution of Kids")
plt.show()



# #### B.4 Box plot to find outliers in customer's yearly Income

# In[6]:



plt.figure(figsize=(16,5))
plt.title(f'Customers Income Boxplot')
ax = sns.boxplot(x = dataFrame['Income'], color = 'tab:orange')
plt.show()

#Here, we can see there are outliers



# #### B.5 Customers Yearly Income Distribution

# In[7]:



#Now, we apply the filter to get data below 150000 range & plotted the distribution
sns.distplot(x = dataFrame[dataFrame['Income'] < 150000].Income,color = 'tab:blue')



# ### C. Data Preparation

# #### C.1 Removing ID column

# In[8]:



dataFrame.drop(['ID'], axis = 1, inplace = True)



# #### C.2 Removing outliers in case of Income column

# In[9]:
```

```
dataFrame = dataFrame.drop(dataFrame[dataFrame['Income'] > 150000].index)
dataFrame.reset_index(drop = True, inplace = True)


# #### C.3 Finding & removing null values

# In[10]:


#print(dataFrame.isnull().sum())
#Income has 24 missing values. We will be fill them by using median
dataFrame['Income'].fillna(dataFrame['Income'].median(), inplace = True)


# #### C.4 Dropping Duplicate Values

# In[11]:


dataFrame.drop_duplicates(inplace = True)
print("Shape: ", dataFrame.shape)


# #### C.5 Creating Age column from birth year

# In[12]:


#We don't know when the dataset was created, because the creator didn't mention any year
```

```
#So, we will assume that the dataset was created on the next day of the last customer enrollment + 2 years, because most of the features are aggregated for last 2 years.

print('Last day when a client was enrolled is ', dataFrame.Dt_Customer.dt.date.max())


dataFrame['Age'] = 2016 - dataFrame['Year_Birth']



# #### C.6 Creating CustomerFor Column from Date of customer's enrollment with the company


# In[13]:



dataFrame['CustomerFor'] = (np.datetime64('2016-12-07') - dataFrame.Dt_Customer).dt.days



# #### C.7 Removing columns: Year_Birth & Dt_Customer


# In[14]:



dataFrame.drop(columns=['Dt_Customer', 'Year_Birth'], inplace=True)



# #### C.8 Merging Education types


# In[15]:



dataFrame['Education'].replace(['2n Cycle', 'Graduation'], ['Master', 'Bachelor'], inplace = True)
```

```
# In[16]:


#Listing all values from education column
educationNamesList = ['Bachelor', 'Basic', 'Master', 'PhD']
px.pie(values = dataFrame.groupby('Education').size(), names = educationNamesList, title = 'Percentage of Education Level Using Pie Chart After Merging')


# #### C.9 Merging Marital Status

# In[17]:


#YOLO - You only live once
#Merge YOLO, Absurd & Alone to Single
dataFrame['Marital_Status'].replace(['YOLO', 'Absurd', 'Alone'], 'Single', inplace = True)


# #### C.10 Creating HasPartner Column

# In[18]:


dataFrame['HasPartner'] = dataFrame["Marital_Status"].replace({'Single': 0, 'Widow': 0, 'Divorced': 0, 'Together': 1, 'Married': 1})
#0 for no, 1 for yes
```

```
# #### C.11 Combining Children & Teens to create NumChildren

# In[19]:


dataFrame['NumChildren'] = dataFrame['Kidhome'] + dataFrame['Teenhome']


# #### C.12 Creating HasChildren column

# In[20]:


dataFrame['HasChildren'] = (dataFrame['NumChildren'] >= 1).astype('int64')


# #### C.13 Creating MntTotal column

# In[21]:


#It indicated the total amount spent by customer in last 2 years
dataFrame['MntTotal'] = dataFrame.MntWines + dataFrame.MntFruits +
dataFrame.MntMeatProducts + dataFrame.MntFishProducts +
dataFrame.MntSweetProducts + dataFrame.MntGoldProds


# #### C.14 Creating NumTotalPurchases column

# In[22]:
```

```
#It indicates the total purchases made by the customer

dataFrame['NumTotalPurchases'] = dataFrame.NumWebPurchases +
dataFrame.NumCatalogPurchases + dataFrame.NumStorePurchases



# #### C.15 Finding & deleting rows where total amount is non zero and purchase is zero


# In[23]:



dataFrame.drop(dataFrame.loc[(dataFrame['NumTotalPurchases'] == 0) &
(dataFrame['MntTotal'] != 0)].index, inplace=True)



# #### C.16 Creating average check column


# In[24]:



dataFrame['AvgCheck'] = dataFrame.MntTotal / dataFrame.NumTotalPurchases



# #### C.17 Creating AcceptedTotal column


# In[25]:



dataFrame['AcceptedTotal'] = dataFrame.AcceptedCmp1 + dataFrame.AcceptedCmp2 +
dataFrame.AcceptedCmp3 + dataFrame.AcceptedCmp4 +
dataFrame.AcceptedCmp5 + dataFrame.Response
```

```
# #### C.18 Removing outliers in Age & AvgCheck

# In[26]:


dataFrame = dataFrame.drop(dataFrame[(dataFrame['Age'] > 100) | (dataFrame['AvgCheck'] > 150)].index)
dataFrame.reset_index(drop = True, inplace = True)
print("Shape: ", dataFrame.shape)


# #### C.19 Using one hot encoding on Education & Martial_Status

# In[27]:


columns = ['Education', 'Marital_Status']
dataFrame = pd.get_dummies(dataFrame, columns = columns, drop_first = True)


# #### C.20 Dropping the non-useful features

# In[28]:


dataFrame = dataFrame.drop(['Kidhome', 'Teenhome', 'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',                    'MntGoldProds', 'NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases', 'AcceptedCmp1', 'AcceptedCmp2',                    'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'Response'], axis = 1)
dataFrame = dataFrame.drop(['Marital_Status_Married', 'Marital_Status_Single', 'Marital_Status_Together', 'Marital_Status_Widow' ], axis = 1)

dataFrame = dataFrame.drop(['Education_Basic', 'Education_PhD', 'Education_Master', 'Education_PhD' ], axis = 1)

dataFrame = dataFrame.drop(['Recency', 'NumDealsPurchases', 'NumWebVisitsMonth', 'Complain', 'Z_CostContact', 'Z_Revenue' ], axis = 1)

dataFrame = dataFrame.drop(['Age', 'CustomerFor', 'HasPartner', 'NumChildren', 'AcceptedTotal', 'HasChildren' ], axis = 1)

print("Shape: ", dataFrame.shape)



# #### C.21 Scaling the data


# In[29]:



scaler = StandardScaler()

dataFrame = scaler.fit_transform(dataFrame)



# ### D. ML/DL Algorithms


# #### D.1 K-Means clustering


# In[30]:



### Finding optimal clusters

inertiaList = []

K = range(1,11) #Sum of squared distances of samples to their closest cluster center.
```

```
for k in K:
    model = KMeans(n_clusters = k)
    model.fit(dataFrame)
    inertiaList.append(model.inertia_)


plt.figure(figsize=(7,5))
plt.plot(K, inertiaList, 'bx-', color = 'purple', linewidth=2)
plt.title('Elbow method', size=15, color='blue', pad=10)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.savefig(imagePath +  "Optimal Clusters Elbow Method")
plt.show()



# In[31]:



kmeans = KMeans(n_clusters = 3).fit(dataFrame)
kmeans.predict(dataFrame)
score = silhouette_score(dataFrame, kmeans.labels_, metric = 'euclidean')
print('Silhouette Score using K Means Algorithm: ', score)



# In[32]:



pca = PCA(n_components = 2)
principalComponents = pca.fit_transform(dataFrame)
#Convert To Dataframe
pcaDf = pd.DataFrame(data = principalComponents, columns = ['Principal Component 1',
'Principal component 2'])
```

```
pcaDf['Clusters'] = kmeans.labels_


#Plot

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(7,5))

sns.scatterplot(pcaDf['Principal Component 1'], pcaDf['Principal component 2'], hue =
pcaDf['Clusters'], s=80, legend="full", palette = 'brg')

plt.title("Scatter Plot Using KMeans", size=15, color='blue', pad = 10)

plt.savefig(imagePath + "Scatter Plot Using KMeans")

plt.show()



# #### D.2 Hierarchical or Agglomerative Clustering


# In[33]:



var = linkage(dataFrame, method = 'ward')

plt.figure(figsize = (12,8))

dendrogram(var)

plt.show()



# In[34]:



clustering = AgglomerativeClustering(n_clusters = 3).fit_predict(dataFrame)

c = AgglomerativeClustering(n_clusters = 3).fit(dataFrame)

score = silhouette_score(dataFrame, c.labels_, metric = 'euclidean')

print('Silhouette Score using Agglomerative Clustering: ', score)
```

```
# In[35]:


pca = PCA(n_components = 2)
principalComponents = pca.fit_transform(dataFrame)
#Convert To Dataframe
pcaDf = pd.DataFrame(data = principalComponents, columns = ['Principal Component 1',
'Principal component 2'])
pcaDf['Clusters'] = c.labels_

#Plot
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(7,5))
sns.scatterplot(pcaDf['Principal Component 1'], pcaDf['Principal component 2'], hue =
pcaDf['Clusters'], s=80, legend="full", palette = 'brg')
plt.title("Scatter Plot Agglomerative Clustering", size=15, color='blue', pad = 10)
plt.savefig(imagePath + "Scatter Plot Using Agglomerative Clustering")
plt.show()


# #### D.3 Deep Neural Network


# In[36]:


from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense
```

```
model = Sequential()

#1. 1st Layer - Input Layer
model.add(Input(shape = dataFrame.shape[1:]))

#2. 2nd Layer - Dense Layer with 13 units
model.add(Dense(units = 8, activation = 'relu'))

#3. 3rd Layer - Dense Layer with 6 units
model.add(Dense(units = 16, activation = 'relu'))

#4. 4th Layer - Dense Layer with 3 units for 3 classes
model.add(Dense(units = 3, activation = 'softmax'))

#Compiling the Model
model.compile(optimizer = Adam(0.002), loss = 'sparse_categorical_crossentropy')

model.summary()



# In[37]:



history = model.fit(dataFrame, kmeans.labels_, epochs = 20)



# In[ ]:
```