Author ~ Saurabh Kumar

Lets First UnderStand Clustering

What is Clustering?

Defination:

- **Clustering is Unsupervised Machine Learning Techinique (Unsupervised ~ Unlabled data) where we are trying grouping of data points.These groups are called Cluster.Data points that are in the same group should have similar properties and/or features, while data points in different groups should have highly dissimilar properties and/or features.**
- **The aim is to segregate groups with similar traits and assign them into clusters.**

Simple Use Case:

> Suppose, you are the head of a rental store and wish to understand preferences of your costumers to scale up your business. Is it possible for you to look at details of each costumer and devise a unique business strategy for each one of them? Definitely not. But, what you can do is to cluster all of your costumers into say 10 groups based on their purchasing habits and use a separate strategy for costumers in each of these 10 groups. And this is what we call clustering.

Why we use Clustering? or How its helps in Data Science???

- Clustering is used for things like **feature engineering** or **pattern discovery**.
- We can use clustering analysis to gain some valuable insights from our data by seeing what groups the data points falls.
- Statistical Analysis of Data.

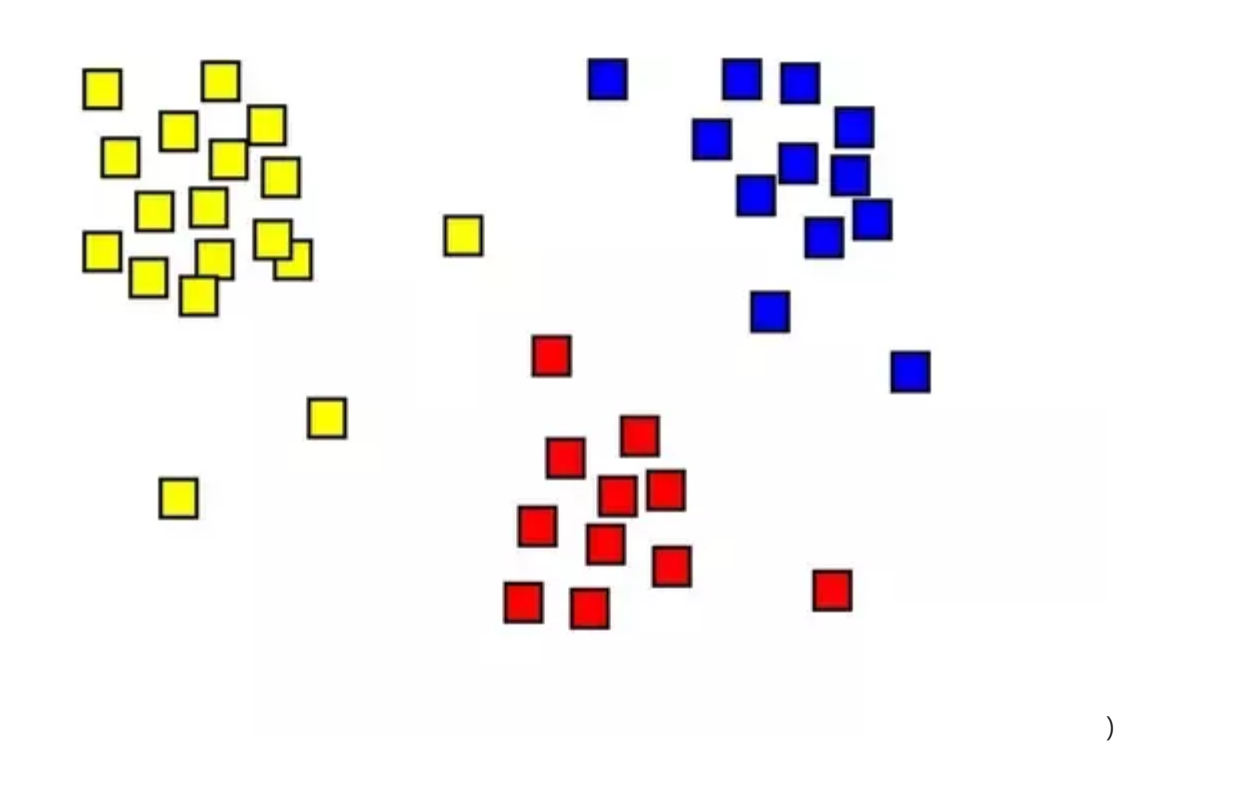
Real-time Use-cases:

- One of the most importat application is related to image processing. detecting distinct kinds of pattern in image data. This can be very effective in biology research, distinguishing objects and identifying patterns.
- The personal data combined with shopping, location, interest, actions and a infinite number of indicators, can be analysed with this methodology, providing very important information and trends. Examples of this are the market research, marketing strategies, web analytics, and a lot of others.
- Other types of applications based on clustering algorithms are climatology, robotics, recommender systems, mathematical and statistical analysis, providing a broad spectrum of utilization.

Possible Applications

- Marketing: Finding groups of customers with similar behavior given a large database of customer data containing their properties and past buying records;
- Biology: Classification of plants and animals given their features;
- Libraries: Book ordering;
- Insurance: Identifying groups of motor insurance policy holders with a high average claim cost; identifying frauds;
- City-planning: Identifying groups of houses according to their house type, value and geographical location;
- Earthquake studies: Clustering observed earthquake epicenters to identify dangerous zones;
- WWW document classification: Clustering weblog data to discover groups of similar access patterns.

Clustering Visualization



)

Types of Clustering

Based on the area of overlap

1. **Hard Clustering**

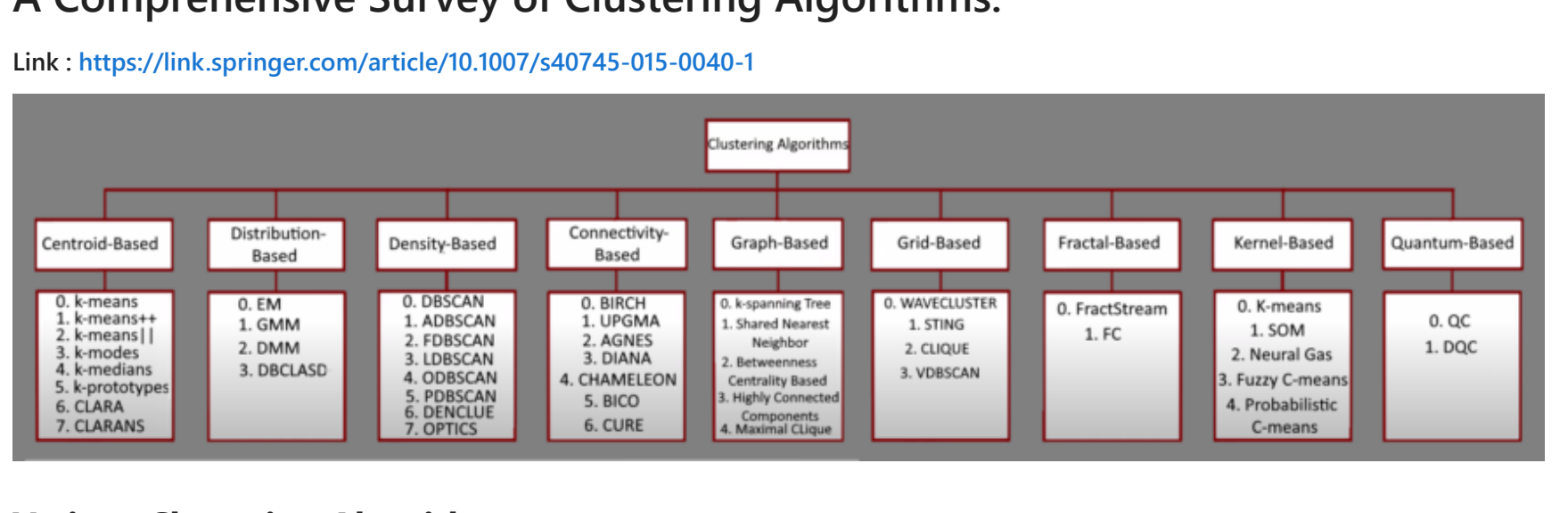
- Clusters don't overlap:k-means, k-means++.Each data point either belongs to a cluster completely or not.

1. **Soft Clustering**

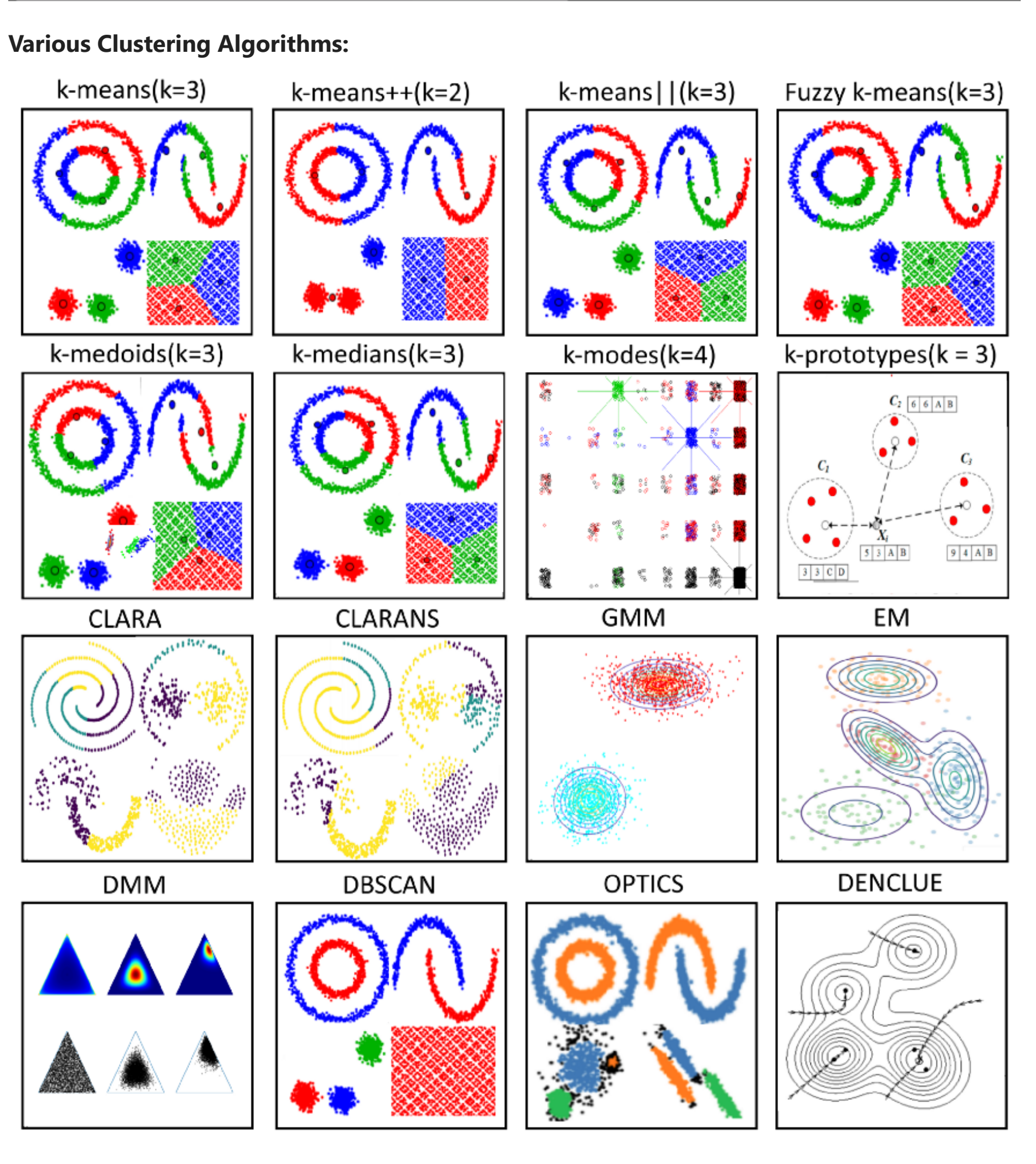
- Clusters overlap:Fuzzy c-means, EM. A data object can exist in more than one cluster with a certain probability or degree of membership. or
- Instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned.

A Comprehensive Survey of Clustering Algorithms.

Link : https://link.springer.com/article/10.1007/s40745-015-0040-1



Various Clustering Algorithms:



Lets Discuss difrrerent type of method used for clustering

Clustering Algorithm Based on Distribution

The basic idea is that the data, generated from the same distribution, belongs to the same cluster if there exists several distributions in the original data. The typical algorithms are DBCLASD and GMM. The core idea of DBCLASD, a dynamic incremental algorithm, is that if the distance between a cluster and its nearest data point satisfies the distribution of expected distance which is generated from the existing data points of that cluster, the nearest data point should belong to this cluster. The core idea of GMM is that GMM consists of several Gaussian distributions from which the original data is generated and the data, obeying the same independent Gaussian distribution, is considered to belong to the same cluster.

Analysis

Algorithms	**DBCLASD**	**GMM**
Complexity	O(n*log(n))	O(n^2*kt)
Comparison	Middle	High

Advantages:

More realistic to give the probability of belonging, relatively high scalability by changing the distribution, number of clusters and so on, and supported by the well developed statistical science;

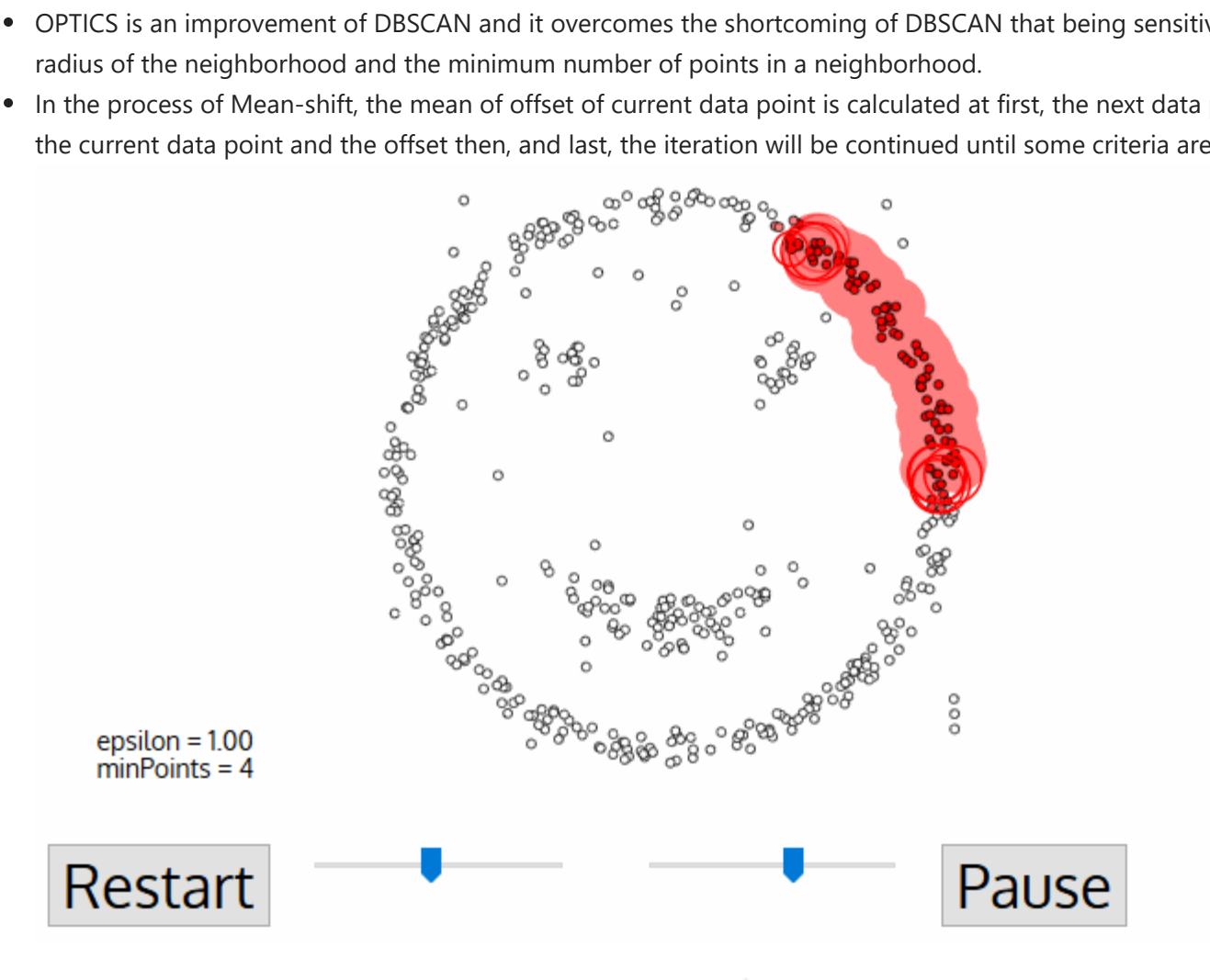
Disadvantages:

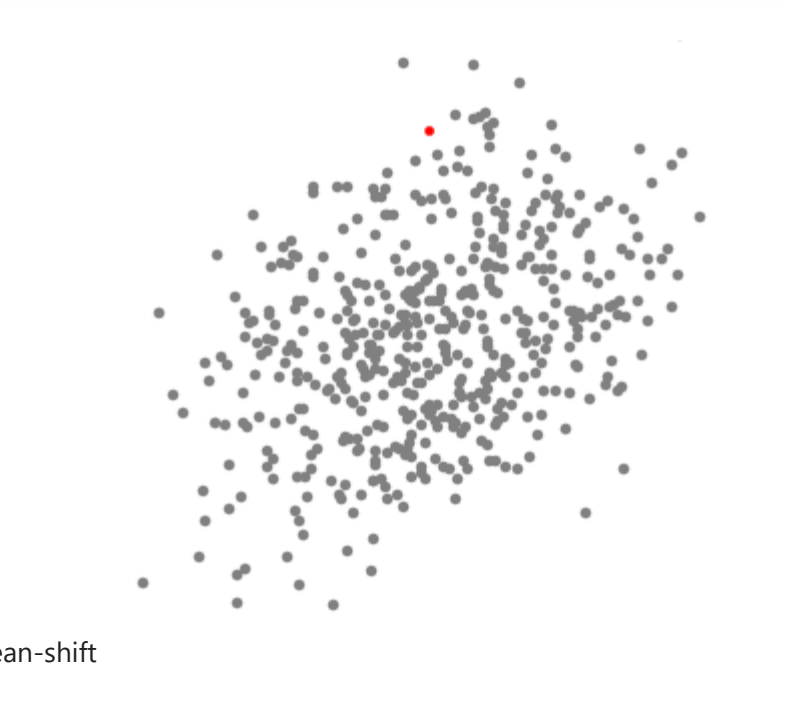
The premise not completely correct, involved in many parameters which have a strong influence on the clustering result and relatively high time complexity.

Clustering Based on Density:

The basic idea of this kind of clustering algorithms is that the data which is in the region with high density of the data space is considered to belong to the same cluster. The typical ones include DBSCAN, OPTICS and Mean-shift.

- DBSCAN is the most well known density-based clustering algorithm, which is generated from the basic idea of this kind of clustering algorithms directly.
- OPTICS is an improvement of DBSCAN and it overcomes the shortcoming of DBSCAN that being sensitive to two parameters, the radius of the neighborhood and the minimum number of points in a neighborhood.
- In the process of Mean-shift, the mean of offset of current data point is calculated at first, the next data point is figured out based on the current data point and the offset then, and last, the iteration will be continued until some criteria are met. #### Analysis :





- ### Mean-shift

Advantages:

Clustering in high efficiency and suitable for data with arbitrary shape.

Disadvantages:

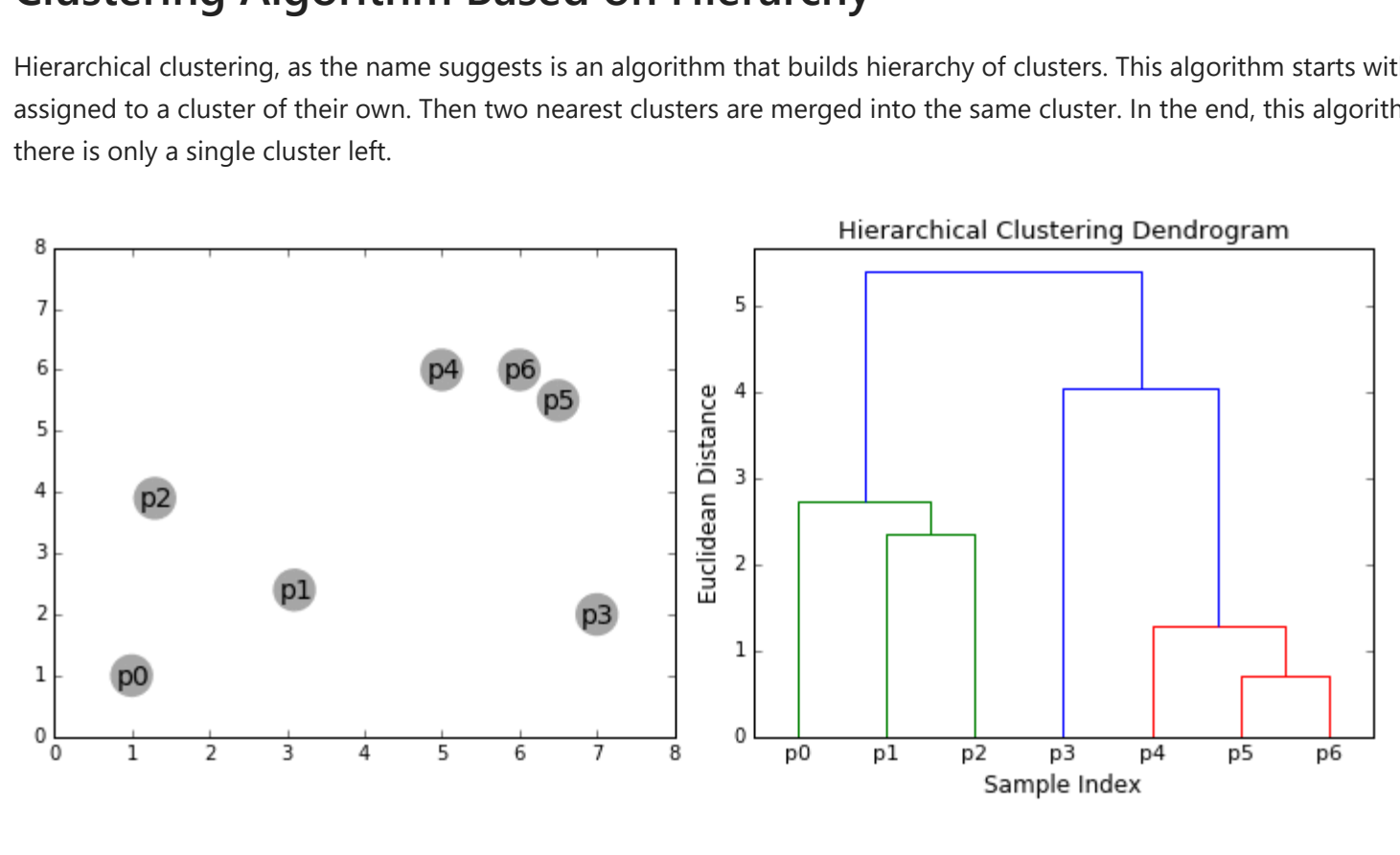
Resulting in a clustering result with low quality when the density of data space isn't even, a memory with big size needed when the data volume is big, and the clustering result highly sensitive to the parameters.

Analysis

Algorithms	DBCLASD	GMM
Complexity	O(n*log(n))	O(n^2*kt)
Comparison	Middle	High

Clustering Algorithm Based on Hierarchy

Hierarchical clustering, as the name suggests is an algorithm that builds hierarchy of clusters. This algorithm starts with all the data points assigned to a cluster of their own. Then two nearest clusters are merged into the same cluster. In the end, this algorithm terminates when there is only a single cluster left.



Advantages:

Suitable for the data set with arbitrary shape and attribute of arbitrary type, the hierarchical relationship among clusters easily detected, and relatively high scalability in general;

Disadvantages:

Relatively high in time complexity in general, the number of clusters needed to be preset.

Analysis

Algorithms	BIRCH	CURE	ROCK	Chameleon
Complexity	O(n)	O(s^2*s)	O(n^2*log(n))	O(n^2)
Comparison	Low	Low	High	High

Clustering Algorithm Based on Fuzzy Theory

The basic idea of this kind of clustering algorithms is that the discrete value of belonging label, {0, 1}, is changed into the continuous interval [0,1], in order to describe the belonging relationship among objects more reasonably. Typical algorithms of this kind of clustering include FCM, FCS and MM. The core idea of FCM is to get membership of each data point to every cluster by optimizing the object function. FCS, different from the traditional fuzzy clustering algorithms, takes the multidimensional hypersphere as the prototype of each cluster, so as to cluster with the distance function based on the hypersphere. MM, based on the Mountain Function, is used to find the center of cluster

Advantages:

More realistic to give the probability of belonging, relatively high accuracy of clustering;

Disadvantages:

Relatively low scalability in general, easily drawn into local optimal, the clustering result sensitive to the initial parameter values, and the number of clusters needed to be preset.

Algorithms	FCM	FCS	MM
Complexity	O(n)	Kernel	O(v^^2*n)
Comparison	Low	High	Middle

Now Lets Jumps to Implementation Section

Famaous Clustering Algorithms:

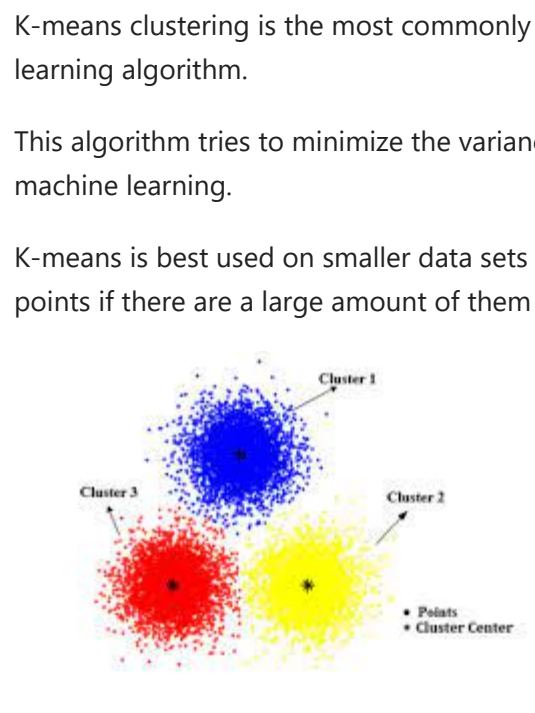
1. K-means clustering algorithm
2. DBSCAN clustering algorithm
3. Gaussian Mixture Model algorithm
4. BIRCH algorithm
5. Affinity Propagation clustering algorithm
6. Mean-Shift clustering algorithm
7. OPTICS algorithm
8. Agglomerative Hierarchy clustering algorithm

K-means clustering algorithm

K-means clustering is the most commonly used clustering algorithm. It's a centroid-based algorithm and the simplest unsupervised learning algorithm.

This algorithm tries to minimize the variance of data points within a cluster. It's also how most people are introduced to unsupervised machine learning.

K-means is best used on smaller data sets because it iterates over all of the data points. That means it'll take more time to classify data points if there are a large amount of them in the data set.



General Implementation

In [ ]:

```python
from numpy import unique
from numpy import where
from matplotlib import pyplot
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans

# initialize the data set we'll work with
training_data, _ = make_classification(
    n_samples=1000,
    n_features=2,
    n_informative=2,
    n_redundant=0,
    n_clusters_per_class=1,
    random_state=4
)

# define the model
kmeans_model = KMeans(n_clusters=2)

# assign each data point to a cluster
kmeans_result = kmeans_model.fit_predict(training_data)

# get all of the unique clusters
kmeans_clusters = unique(kmeans_result)

# plot the kmeans clusters
for kmeans_cluster in kmeans_clusters:
    # get data points that fall in this cluster
    index = where(kmeans_result == kmeans_cluster)
    # make the plot
    pyplot.scatter(training_data[index, 0], training_data[index, 1])

# show the kmeans plot
pyplot.show()
```

Lets Implement K-means Clustering as per our dataset:

In [1]:

```python
#importing Library
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import os
from sklearn import cluster
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import  LabelEncoder, RobustScaler , MinMaxScaler ,StandardScaler


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation ,Dropout
from tensorflow.keras import layers

import plotly.express as px
from scipy import signal
import scipy
#to supress warning
import warnings
warnings.filterwarnings('ignore')


#to make shell more intractive
from IPython.display import display

# setting up the chart size and background
plt.rcParams['figure.figsize'] = (16, 8)
plt.style.use('fivethirtyeight')
sns.set_style("darkgrid")
```

In [2]:

```python
os.listdir()
```

Out[2]:

```
['.ipynb_checkpoints',
 'Algorithms.docx',
 'all-Clustering-algo.png',
 'clustering-algorithms.ipynb',
 'data',
 'variousclusterimg.png']
```

In [3]:

```python
pwd
```

Out[3]:

```
'E:\\DataScience\\Clustering-Algorithms'
```

In [4]:

```python
path='E:\\DataScience\\Clustering-Algorithms\\data'
```

In [5]:

```python
os.listdir(path)
```

Out[5]:

```
['data.csv', 'sample_submission.csv']
```

In [6]:

```python
#import dataset
df =pd.read_csv(path+"\\data.csv")
sample =pd.read_csv(path+"\\sample_submission.csv")
```

In [7]:

```python
df.head(10).style.background_gradient(cmap = 'rocket_r')
```

Out[7]:

	id	f_00	f_01	f_02	f_03	f_04	f_05	f_06	f_07	f_08	f_09	f_10	f_11	f_12	f_13	f_14	f_15
0	0	-0.389420	-0.912791	0.648951	0.589045	-0.830817	0.733624	2.258560	2	13	14	5	13	6	6	-0.469819	0.358126
1	1	-0.689249	-0.453954	0.654175	0.995248	-1.653020	0.863810	-0.090651	2	3	6	4	6	16	9	0.591035	-0.396915
2	2	0.809079	0.324568	-1.170602	-0.624491	0.105448	0.783948	1.988301	5	11	5	8	9	3	11	-0.679875	0.469326
3	3	-0.500923	0.229049	0.264109	0.231520	0.415012	-1.221269	0.138850	6	2	13	8	9	6	4	-0.389456	0.626762
4	4	-0.671268	-1.039533	-0.270155	-1.830264	-0.290108	-1.852809	0.781898	8	7	5	3	1	13	11	-0.120743	-0.615578
5	5	-0.888945	-1.121026	0.988608	0.224778	1.655074	-1.274271	0.371443	4	1	1	3	2	12	10	0.944266	-1.270700
6	6	-0.096137	-0.247528	-0.642176	-0.534849	1.051045	0.257412	-2.129200	2	2	17	13	5	3	0	0.400967	0.164763
7	7	-0.908615	0.065319	-0.427002	-0.392070	1.135532	0.797638	-0.510480	2	8	5	6	10	3	8	1.927999	1.536572
8	8	0.997730	-1.153765	-0.908964	0.984962	1.176252	-0.061570	-1.015247	1	4	4	1	10	8	15	0.375556	0.155453
9	9	-0.641308	0.376707	-0.427771	0.802239	-1.049559	-0.215596	-1.277036	1	15	1	1	5	6	8	-0.577919	0.076958

In [8]:

```python
df.shape
```

Out[8]:

```
(98000, 30)
```

In [9]:

```python
df.drop("id", axis=1, inplace=True)
```

In [10]:

```python
df.describe().T.style.background_gradient(cmap = 'RdYlGn')
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
f_00	98000.000000	0.001220	1.002801	-4.732235	-0.675226	0.002022	0.677271	4.490521
f_01	98000.000000	0.005580	1.000742	-4.202795	-0.670985	0.006650	0.677746	4.324974
f_02	98000.000000	-0.001042	1.001373	-4.377021	-0.672779	-0.000324	0.677086	4.560247
f_03	98000.000000	-0.000700	1.000422	-4.010826	-0.672540	-0.003185	0.672097	4.399373
f_04	98000.000000	-0.003522	1.003061	-4.535903	-0.682510	-0.003307	0.677589	4.050549
f_05	98000.000000	-0.001612	1.000532	-4.300767	-0.675066	0.001024	0.673344	4.710316
f_06	98000.000000	-0.003042	0.997434	-4.894525	-0.680421	-0.002053	0.668112	3.998595
f_07	98000.000000	5.545918	3.691840	0.000000	3.000000	5.000000	8.000000	32.000000
f_08	98000.000000	6.763061	4.152348	0.000000	4.000000	6.000000	9.000000	30.000000
f_09	98000.000000	8.193163	5.904919	0.000000	4.000000	7.000000	11.000000	44.000000
f_10	98000.000000	8.057878	4.700226	0.000000	5.000000	7.000000	11.000000	36.000000
f_11	98000.000000	8.084990	4.197106	0.000000	5.000000	8.000000	11.000000	28.000000
f_12	98000.000000	7.076388	4.433308	0.000000	4.000000	6.000000	10.000000	38.000000
f_13	98000.000000	6.286102	4.120077	0.000000	3.000000	6.000000	9.000000	30.000000
f_14	98000.000000	0.003413	0.998940	-4.377021	-0.671810	0.003536	0.678679	4.490521
f_15	98000.000000	-0.002639	1.001096	-4.535903	-0.677757	-0.004299	0.673859	4.270708
f_16	98000.000000	0.005404	1.003277	-4.182233	-0.671055	0.006535	0.680903	4.157538
f_17	98000.000000	0.001656	1.001258	-4.468930	-0.672635	0.001663	0.673174	4.230849
f_18	98000.000000	-0.005399	0.995965	-4.149688	-0.680243	-0.001574	0.666996	4.710316
f_19	98000.000000	-0.004513	1.004372	-4.894525	-0.678773	-0.000587	0.672149	4.560247
f_20	98000.000000	-0.000515	1.002962	-4.732235	-0.679777	-0.000806	0.675437	4.399373
f_21	98000.000000	-0.001670	0.999703	-4.438130	-0.675147	0.000819	0.676881	4.135419
f_22	98000.000000	-0.038752	1.477858	-6.873999	-1.022964	-0.056687	0.930158	6.517721
f_23	98000.000000	-0.220002	1.494836	-8.234305	-1.203204	-0.219046	0.764690	6.054831
f_24	98000.000000	0.166434	1.543014	-7.792363	-0.903385	0.167074	1.217432	7.527271
f_25	98000.000000	-0.064309	1.576086	-6.593842	-1.128966	-0.099221	0.987684	7.544731
f_26	98000.000000	-0.062540	1.428055	-7.375719	-0.975680	-0.070852	0.843212	7.005608
f_27	98000.000000	0.098472	1.305407	-7.335556	-0.746489	0.082230	0.925306	7.205971
f_28	98000.000000	-0.230910	1.528476	-6.954151	-1.262606	-0.271319	0.770516	6.977150

In [11]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98000 entries, 0 to 97999
Data columns (total 29 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   f_00    98000 non-null  float64
 1   f_01    98000 non-null  float64
 2   f_02    98000 non-null  float64
 3   f_03    98000 non-null  float64
 4   f_04    98000 non-null  float64
 5   f_05    98000 non-null  float64
 6   f_06    98000 non-null  float64
 7   f_07    98000 non-null  int64  
 8   f_08    98000 non-null  int64  
 9   f_09    98000 non-null  int64  
 10  f_10    98000 non-null  int64  
 11  f_11    98000 non-null  int64  
 12  f_12    98000 non-null  int64  
 13  f_13    98000 non-null  int64  
 14  f_14    98000 non-null  float64
 15  f_15    98000 non-null  float64
 16  f_16    98000 non-null  float64
 17  f_17    98000 non-null  float64
 18  f_18    98000 non-null  float64
 19  f_19    98000 non-null  float64
 20  f_20    98000 non-null  float64
 21  f_21    98000 non-null  float64
 22  f_22    98000 non-null  float64
 23  f_23    98000 non-null  float64
 24  f_24    98000 non-null  float64
 25  f_25    98000 non-null  float64
 26  f_26    98000 non-null  float64
 27  f_27    98000 non-null  float64
 28  f_28    98000 non-null  float64
dtypes: float64(22), int64(7)
memory usage: 21.7 MB
```

In [12]:

```python
# Correlation matrix
df.corr().style.background_gradient(cmap = 'rocket_r')
```

Out[12]:

	f_00	f_01	f_02	f_03	f_04	f_05	f_06	f_07	f_08	f_09	f_10	f_11	f_12
f_00	1.000000	-0.001955	0.002660	0.000105	0.008797	0.003817	0.002903	-0.003847	-0.000866	-0.000976	0.002510	-0.000938	0.004578
f_01	-0.001955	1.000000	0.001402	-0.001698	-0.001727	-0.001444	0.001826	0.006539	0.001023	0.003428	0.002071	-0.002024	-0.007009
f_02	0.002660	0.001402	1.000000	-0.003271	-0.000732	-0.003743	0.003730	0.002603	0.004888	-0.004690	0.004708	0.001624	0.003369
f_03	0.000105	-0.001698	-0.003271	1.000000	0.000238	0.009835	0.002570	-0.000562	0.000095	0.001626	-0.001752	0.003533	-0.002081
f_04	0.008797	-0.001727	-0.000732	0.000238	1.000000	0.004550	0.004143	0.007164	-0.002044	0.002025	-0.003960	0.000218	0.001141
f_05	0.003817	-0.001444	-0.003743	0.009835	0.004550	1.000000	0.004825	0.003289	-0.002826	-0.000695	0.002807	-0.001368	-0.000530
f_06	0.002903	0.001826	0.003730	0.002570	0.004143	0.004825	1.000000	-0.000300	-0.005988	0.009750	-0.004827	-0.004692	-0.001239
f_07	-0.003847	0.006539	0.002603	-0.000562	0.007164	0.003289	-0.000300	1.000000	-0.131742	0.036707	-0.106755	-0.012389	-0.196534
f_08	-0.000866	0.001023	0.004888	0.000095	-0.002044	-0.002826	-0.005988	-0.131742	1.000000	-0.360734	-0.051459	0.179527	-0.076524
f_09	-0.000976	0.003428	-0.004690	0.001626	0.002025	-0.000695	0.009750	0.036707	-0.360734	1.000000	0.035432	-0.202523	-0.295563
f_10	0.002510	0.002071	0.004708	-0.001752	-0.003960	0.002807	-0.004827	-0.106755	-0.051459	0.035432	1.000000	-0.267171	-0.070415
f_11	-0.000938	-0.002024	0.001624	0.003533	0.000218	-0.001368	-0.004692	-0.012389	0.179527	-0.202523	-0.267171	1.000000	-0.126511
f_12	0.004578	-0.007009	0.003369	-0.002081	0.001141	-0.000530	-0.001239	-0.196534	-0.076524	-0.295563	-0.070415	-0.126511	1.000000
f_13	-0.001075	-0.006544	-0.000620	-0.004417	-0.000351	-0.001335	0.000689	-0.117606	-0.014362	-0.281970	-0.281057	0.017010	0.195191
f_14	-0.003255	-0.003968	0.002112	0.004566	0.001196	0.003204	-0.000707	0.001908	0.000242	0.006359	-0.004625	-0.001010	-0.000842
f_15	0.005267	-0.000157	0.005397	0.001454	-0.002549	-0.001235	0.004625	0.000890	0.000011	0.000875	0.000805	-0.000184	0.001116
f_16	-0.001998	0.000941	-0.003674	-0.000372	-0.003036	0.003359	0.003590	0.004975	-0.006046	-0.000845	-0.008010	0.003038	0.003925
f_17	0.005100	-0.000327	-0.002383	0.003240	-0.002535	-0.002734	0.004411	-0.001435	-0.000272	-0.001134	-0.004454	0.003952	-0.003726
f_18	-0.000561	0.003146	0.006187	0.002039	0.000748	0.002783	-0.006175	-0.004443	-0.000551	0.000692	0.000219	0.000722	-0.000228
f_19	0.000437	-0.001402	-0.002743	0.000540	0.000731	0.007995	-0.005252	0.003504	0.002888	0.000224	-0.000655	0.005591	-0.004642
f_20	0.002687	-0.001852	0.002398	-0.005607	0.000549	0.000832	0.000395	-0.002860	-0.001366	0.003695	0.001802	0.001004	0.000585
f_21	-0.000447	0.001099	-0.000301	0.006154	-0.000956	-0.002269	-0.002691	-0.000085	-0.000753	0.000747	0.007463	-0.002893	-0.002922
f_22	0.004087	-0.002965	0.004971	0.001807	0.003801	0.002085	0.000597	0.003461	0.072612	-0.123708	-0.022650	0.034807	0.053503
f_23	-0.003756	0.001760	0.002971	0.001241	-0.002564	-0.003774	-0.002754	-0.091825	0.028442	-0.017430	0.061926	0.009575	-0.007275
f_24	0.001088	-0.005635	-0.002328	-0.002235	-0.000324	0.002300	-0.002924	0.030238	0.011038	-0.014524	-0.016214	-0.006282	0.015209
f_25	0.006216	0.000033	0.001797	0.000362	-0.000765	0.002087	-0.006603	0.093444	0.072565	-0.081116	-0.063222	0.028018	-0.011212
f_26	0.000615	-0.001936	-0.001460	0.002074	0.002906	-0.001417	-0.000150	-0.017622	0.085690	-0.085901	0.034416	0.002184	0.001042
f_27	0.003888	0.000229	0.002210	0.000427	0.001421	-0.003124	-0.005421	-0.024546	-0.015953	-0.064825	0.040545	-0.044690	0.122341
f_28	0.001103	-0.005528	0.000544	0.001842	-0.003389	0.002519	-0.000110	-0.022357	-0.029955	0.018955	0.015919	-0.011339	0.030514

In [13]:

```python
# Correlation matrix
corrMatrix =df.corr(method='pearson', min_periods=1)
corrMatrix.style.background_gradient(cmap = 'rocket_r')
```

Out[13]:

	f_00	f_01	f_02	f_03	f_04	f_05	f_06	f_07	f_08	f_09	f_10	f_11	f_12
f_00	1.000000	-0.001955	0.002660	0.000105	0.008797	0.003817	0.002903	-0.003847	-0.000866	-0.000976	0.002510	-0.000938	0.004578
f_01	-0.001955	1.000000	0.001402	-0.001698	-0.001727	-0.001444	0.001826	0.006539	0.001023	0.003428	0.002071	-0.002024	-0.007009
f_02	0.002660	0.001402	1.000000	-0.003271	-0.000732	-0.003743	0.003730	0.002603	0.004888	-0.004690	0.004708	0.001624	0.003369
f_03	0.000105	-0.001698	-0.003271	1.000000	0.000238	0.009835	0.002570	-0.000562	0.000095	0.001626	-0.001752	0.003533	-0.002081
f_04	0.008797	-0.001727	-0.000732	0.000238	1.000000	0.004550	0.004143	0.007164	-0.002044	0.002025	-0.003960	0.000218	0.001141
f_05	0.003817	-0.001444	-0.003743	0.009835	0.004550	1.000000	0.004825	0.003289	-0.002826	-0.000695	0.002807	-0.001368	-0.000530
f_06	0.002903	0.001826	0.003730	0.002570	0.004143	0.004825	1.000000	-0.000300	-0.005988	0.009750	-0.004827	-0.004692	-0.001239
f_07	-0.003847	0.006539	0.002603	-0.000562	0.007164	0.003289	-0.000300	1.000000	-0.131742	0.036707	-0.106755	-0.012389	-0.196534
f_08	-0.000866	0.001023	0.004888	0.000095	-0.002044	-0.002826	-0.005988	-0.131742	1.000000	-0.360734	-0.051459	0.179527	-0.076524
f_09	-0.000976	0.003428	-0.004690	0.001626	0.002025	-0.000695	0.009750	0.036707	-0.360734	1.000000	0.035432	-0.202523	-0.295563
f_10	0.002510	0.002071	0.004708	-0.001752	-0.003960	0.002807	-0.004827	-0.106755	-0.051459	0.035432	1.000000	-0.267171	-0.070415
f_11	-0.000938	-0.002024	0.001624	0.003533	0.000218	-0.001368	-0.004692	-0.012389	0.179527	-0.202523	-0.267171	1.000000	-0.126511
f_12	0.004578	-0.007009	0.003369	-0.002081	0.001141	-0.000530	-0.001239	-0.196534	-0.076524	-0.295563	-0.070415	-0.126511	1.000000
f_13	-0.001075	-0.006544	-0.000620	-0.004417	-0.000351	-0.001335	0.000689	-0.117606	-0.014362	-0.281970	-0.281057	0.017010	0.195191
f_14	-0.003255	-0.003968	0.002112	0.004566	0.001196	0.003204	-0.000707	0.001908	0.000242	0.006359	-0.004625	-0.001010	-0.000842
f_15	0.005267	-0.000157	0.005397	0.001454	-0.002549	-0.001235	0.004625	0.000890	0.000011	0.000875	0.000805	-0.000184	0.001116
f_16	-0.001998	0.000941	-0.003674	-0.000372	-0.003036	0.003359	0.003590	0.004975	-0.006046	-0.000845	-0.008010	0.003038	0.003925
f_17	0.005100	-0.000327	-0.002383	0.003240	-0.002535	-0.002734	0.004411	-0.001435	-0.000272	-0.001134	-0.004454	0.003952	-0.003726
f_18	-0.000561	0.003146	0.006187	0.002039	0.000748	0.002783	-0.006175	-0.004443	-0.000551	0.000692	0.000219	0.000722	-0.000228
f_19	0.000437	-0.001402	-0.002743	0.000540	0.000731	0.007995	-0.005252	0.003504	0.002888	0.000224	-0.000655	0.005591	-0.004642
f_20	0.002687	-0.001852	0.002398	-0.005607	0.000549	0.000832	0.000395	-0.002860	-0.001366	0.003695	0.001802	0.001004	0.000585
f_21	-0.000447	0.001099	-0.000301	0.006154	-0.000956	-0.002269	-0.002691	-0.000085	-0.000753	0.000747	0.007463	-0.002893	-0.002922
f_22	0.004087	-0.002965	0.004971	0.001807	0.003801	0.002085	0.000597	0.003461	0.072612	-0.123708	-0.022650	0.034807	0.053503
f_23	-0.003756	0.001760	0.002971	0.001241	-0.002564	-0.003774	-0.002754	-0.091825	0.028442	-0.017430	0.061926	0.009575	-0.007275
f_24	0.001088	-0.005635	-0.002328	-0.002235	-0.000324	0.002300	-0.002924	0.030238	0.011038	-0.014524	-0.016214	-0.006282	0.015209
f_25	0.006216	0.000033	0.001797	0.000362	-0.000765	0.002087	-0.006603	0.093444	0.072565	-0.081116	-0.063222	0.028018	-0.011212
f_26	0.000615	-0.001936	-0.001460	0.002074	0.002906	-0.001417	-0.000150	-0.017622	0.085690	-0.085901	0.034416	0.002184	0.001042
f_27	0.003888	0.000229	0.002210	0.000427	0.001421	-0.003124	-0.005421	-0.024546	-0.015953	-0.064825	0.040545	-0.044690	0.122341
f_28	0.001103	-0.005528	0.000544	0.001842	-0.003389	0.002519	-0.000110	-0.022357	-0.029955	0.018955	0.015919	-0.011339	0.030514

In [14]:

```python
#Classifying Catogorical and Numerical
cat_columns = [x for (x, y) in df.dtypes.items() if y == "int64"]
num_columns = [x for (x, y) in df.dtypes.items() if y == "float64"]

num_data = df[num_columns]
cat_data = df[cat_columns]
print('Categorical columns -> ', cat_columns)
print('\nNumerical columns -> ', num_columns)
```

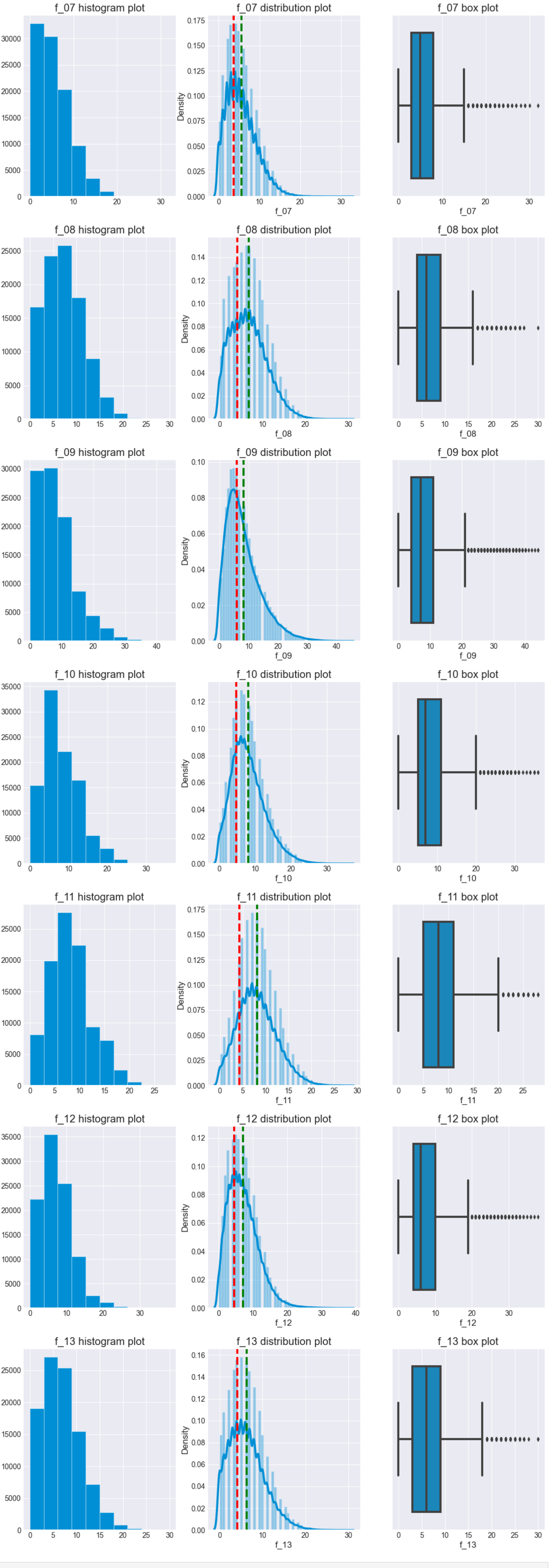
```
Categorical columns ->  ['f_07', 'f_08', 'f_09', 'f_10', 'f_11', 'f_12', 'f_13']

Numerical columns ->  ['f_00', 'f_01', 'f_02', 'f_03', 'f_04', 'f_05', 'f_06', 'f_14', 'f_15', 'f_16', 'f_17', 'f_18', 'f_19', 'f_20', 'f_21', 'f_22', 'f_23', 'f_24', 'f_25', 'f_26', 'f_27', 'f_28']
```

In [15]:

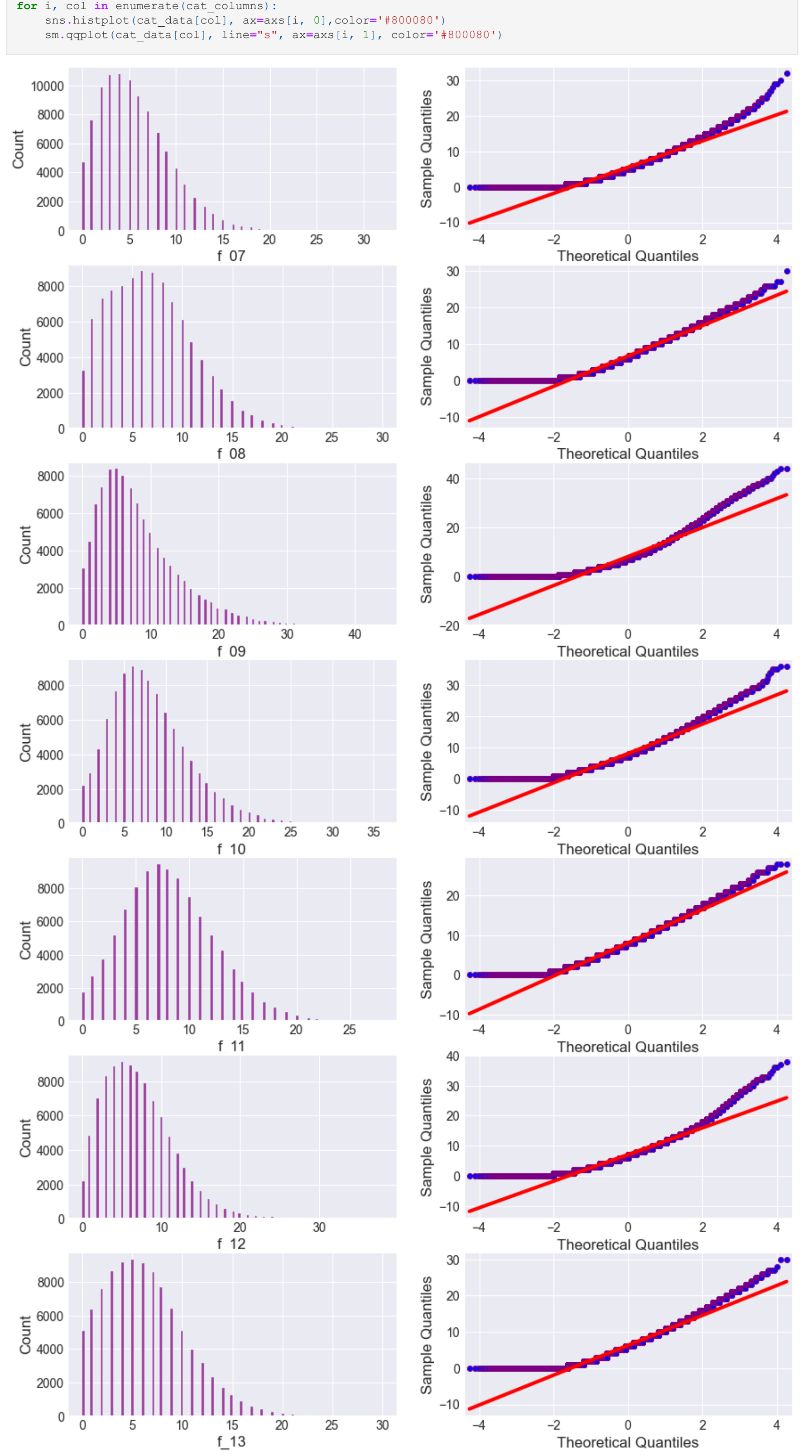
```python
def mix_plot(feature):
    plt.figure(figsize=(16, 6))
    plt.subplot(1, 3, 1)
    feature.plot(kind = 'hist')
    plt.title(f'{feature.name} histogram plot')
    #mean = feature.describe().mean()
    plt.subplot(1, 3, 2)
    mu, sigma = scipy.stats.norm.fit(feature)
    sns.distplot(feature)
    #plt.legend({'--': mu, 'sigma': sigma})
    plt.axvline(mu, linestyle = '--', color = 'green', )
    plt.axvline(sigma, linestyle = '--', color = 'red')
    plt.title(f'{feature.name} distribution plot')
    plt.subplot(1, 3, 3)
    sns.boxplot(feature)
    plt.title(f'{feature.name} box plot')
    plt.show()
```

In [16]:

```python
for i in cat_data.columns:
    mix_plot(cat_data[i])
```



In [17]:
```
import statsmodels.api as sm
fig, axs = plt.subplots(7, 2,
                        figsize=(13, 26))
for i, col in enumerate(cat_columns):
    sns.histplot(cat_data[col], ax=axs[i, 0],color='#800080')
    sm.qqplot(cat_data[col], line="s", ax=axs[i, 1], color='#800080')
```



In [18]:
```
from sklearn.cluster import KMeans
X = df.values
# 20 clusters
n_clusters = 20
# Train K-Means.
kmeans = KMeans(n_clusters = n_clusters, n_init = 20).fit(X)
```

In [19]:
```
kmeans.labels_
```

Out[19]:
```
array([ 8,  7, 18, ...,  2, 13, 10])
```

In [20]:
```
target = kmeans.labels_
```

In [21]:
```
pd.DataFrame(target)[0].value_counts()
```

Out[21]:
```
10    7510
5     6714
1     6234
11    5984
2     5967
8     5927
16    5343
19    5322
15    5189
14    5170
18    5026
0     5011
12    4496
4     4196
9     4183
13    3970
6     3436
3     3256
7     2677
17    2389
Name: 0, dtype: int64
```

In [22]:
```
from sklearn.preprocessing import StandardScaler,RobustScaler
sc = StandardScaler()
rb = RobustScaler()

##Scaling of Data
rb.fit(df)
df = rb.transform(df)
df.shape, target.shape
```

Out[22]:
```
((98000, 29), (98000,))
```

In [23]:
```
#creating_neural_network
model =Sequential()
#1st_hidden_layers
model.add(Dense(units=256, activation='relu', use_bias=True, kernel_initializer='he_uniform',input_shape=(29,)))
#2nd_hidden_layers
model.add(Dense(units=128,activation='relu',kernel_initializer='he_uniform'))
model.add(layers.Dropout(0.25))
#3rd_hidden_layers
model.add(Dense(units=128,activation='relu',kernel_initializer='he_uniform'))
model.add(layers.Dropout(0.25))
#4nd_hidden_layers
model.add(Dense(units=64,activation='relu',kernel_initializer='he_uniform'))
#5th_hidden_layers
model.add(Dense(units=32,activation='relu',kernel_initializer='he_uniform'))
#6th_hidden_layer
model.add(Dense(units=20,activation='softmax',kernel_initializer='glorot_uniform'))
#model compile
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001), loss=tf.keras.losses.SparseCategoricalC
```

In [24]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	7680
dense_1 (Dense)	(None, 128)	32896
dropout (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 128)	16512
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 32)	2080
dense_5 (Dense)	(None, 20)	660

Total params: 68,084
Trainable params: 68,084
Non-trainable params: 0

In [25]:
```
model.layers
```

Out[25]:
```
[<keras.layers.core.dense.Dense at 0x1e63dd74370>,
 <keras.layers.core.dense.Dense at 0x1e63dd74a60>,
 <keras.layers.core.dropout.Dropout at 0x1e63ddb4a60>,
 <keras.layers.core.dense.Dense at 0x1e630fc2700>,
 <keras.layers.core.dropout.Dropout at 0x1e630fc2ee0>,
 <keras.layers.core.dense.Dense at 0x1e630fc27f0>,
 <keras.layers.core.dense.Dense at 0x1e63113ce20>,
 <keras.layers.core.dense.Dense at 0x1e631151f10>]
```

In [26]:
```
save_best = tf.keras.callbacks.ModelCheckpoint("Model.h5", verbose=1, monitor='val_accuracy', save_best_only=Tr
```

In [27]:
```
#model traning
model.fit(df, target, validation_split=0.2, epochs=30, batch_size=256, shuffle=True, callbacks=[save_best])
```

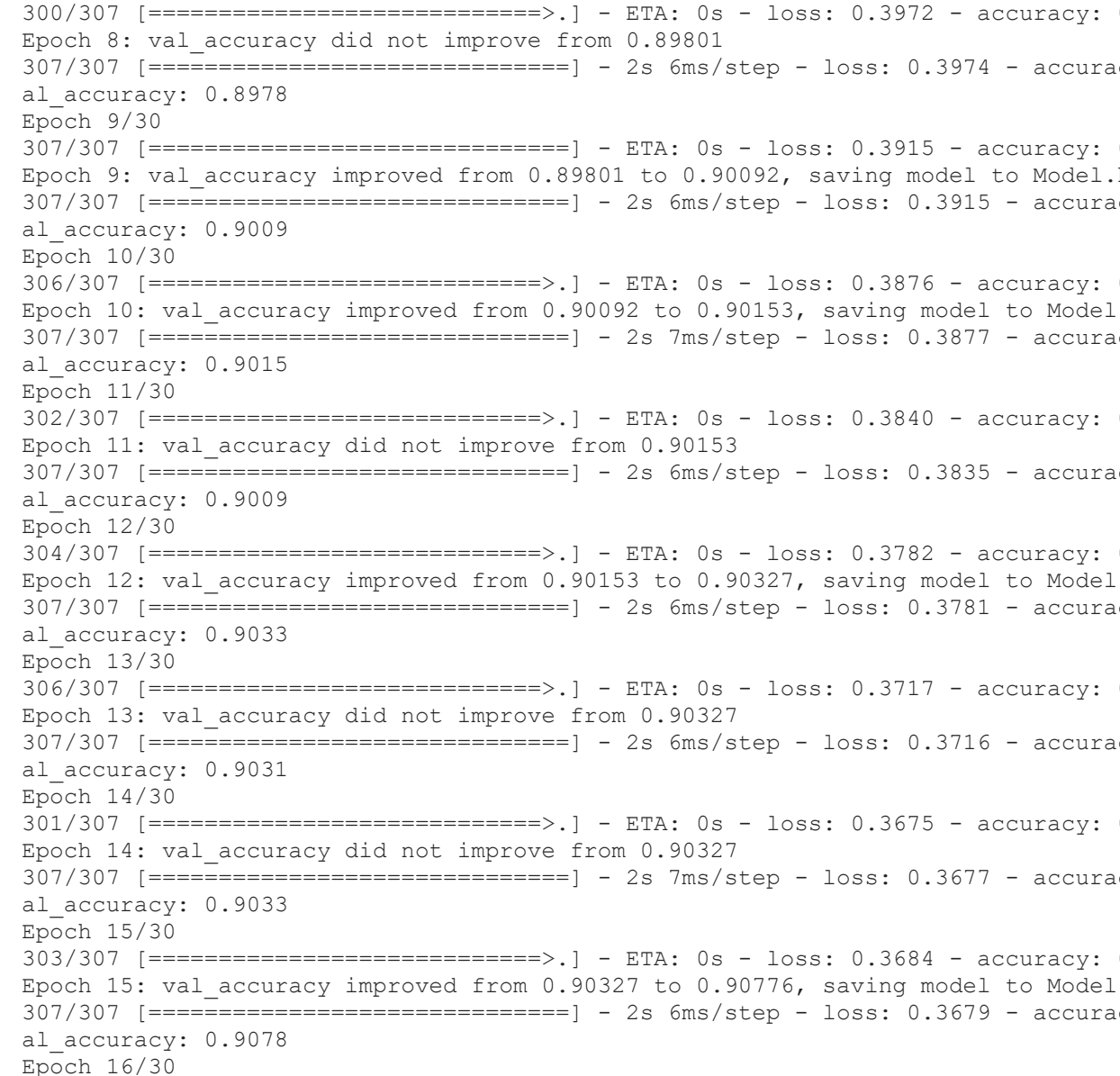
```
Epoch 1/30
296/307 [============================>.] - ETA: 0s - loss: 2.9110 - accuracy: 0.0968
Epoch 1: val_accuracy improved from -inf to 0.23673, saving model to Model.h5
307/307 [==============================] - 6s 7ms/step - loss: 2.9013 - accuracy: 0.0996 - val_loss: 2.4914 - v
al_accuracy: 0.2367
Epoch 2/30
299/307 [============================>.] - ETA: 0s - loss: 2.1158 - accuracy: 0.3024
Epoch 2: val_accuracy improved from 0.23673 to 0.53745, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 2.1071 - accuracy: 0.3046 - val_loss: 1.3987 - v
al_accuracy: 0.5374
Epoch 3/30
305/307 [============================>.] - ETA: 0s - loss: 1.4481 - accuracy: 0.4977
Epoch 3: val_accuracy improved from 0.53745 to 0.67255, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 1.4477 - accuracy: 0.4980 - val_loss: 0.9609 - v
al_accuracy: 0.6726
Epoch 4/30
299/307 [============================>.] - ETA: 0s - loss: 1.1500 - accuracy: 0.5914
Epoch 4: val_accuracy improved from 0.67255 to 0.73628, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 1.1479 - accuracy: 0.5923 - val_loss: 0.7446 - v
al_accuracy: 0.7363
Epoch 5/30
305/307 [============================>.] - ETA: 0s - loss: 0.9854 - accuracy: 0.6421
Epoch 5: val_accuracy improved from 0.73628 to 0.77342, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.9849 - accuracy: 0.6423 - val_loss: 0.6407 - v
al_accuracy: 0.7734
Epoch 6/30
303/307 [============================>.] - ETA: 0s - loss: 0.8920 - accuracy: 0.6727
Epoch 6: val_accuracy improved from 0.77342 to 0.78980, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.8912 - accuracy: 0.6733 - val_loss: 0.5839 - v
al_accuracy: 0.7898
Epoch 7/30
302/307 [============================>.] - ETA: 0s - loss: 0.8240 - accuracy: 0.6961
Epoch 7: val_accuracy improved from 0.78980 to 0.80306, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.8236 - accuracy: 0.6961 - val_loss: 0.5398 - v
al_accuracy: 0.8031
Epoch 8/30
301/307 [============================>.] - ETA: 0s - loss: 0.7770 - accuracy: 0.7107
Epoch 8: val_accuracy improved from 0.80306 to 0.81209, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.7773 - accuracy: 0.7108 - val_loss: 0.5110 - v
al_accuracy: 0.8121
Epoch 9/30
297/307 [============================>.] - ETA: 0s - loss: 0.7395 - accuracy: 0.7221
Epoch 9: val_accuracy improved from 0.81209 to 0.82281, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.7388 - accuracy: 0.7226 - val_loss: 0.4819 - v
al_accuracy: 0.8228
Epoch 10/30
306/307 [============================>.] - ETA: 0s - loss: 0.7003 - accuracy: 0.7372
Epoch 10: val_accuracy improved from 0.82281 to 0.82980, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.7002 - accuracy: 0.7373 - val_loss: 0.4613 - v
al_accuracy: 0.8298
Epoch 11/30
307/307 [==============================] - ETA: 0s - loss: 0.6740 - accuracy: 0.7470
Epoch 11: val_accuracy improved from 0.82980 to 0.83520, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.6740 - accuracy: 0.7470 - val_loss: 0.4456 - v
al_accuracy: 0.8352
Epoch 12/30
307/307 [==============================] - ETA: 0s - loss: 0.6497 - accuracy: 0.7564
Epoch 12: val_accuracy improved from 0.83520 to 0.84077, saving model to Model.h5
307/307 [==============================] - 2s 7ms/step - loss: 0.6497 - accuracy: 0.7564 - val_loss: 0.4283 - v
al_accuracy: 0.8408
Epoch 13/30
305/307 [============================>.] - ETA: 0s - loss: 0.6258 - accuracy: 0.7644
Epoch 13: val_accuracy improved from 0.84077 to 0.84138, saving model to Model.h5
307/307 [==============================] - 2s 7ms/step - loss: 0.6258 - accuracy: 0.7643 - val_loss: 0.4179 - v
al_accuracy: 0.8414
Epoch 14/30
304/307 [============================>.] - ETA: 0s - loss: 0.6106 - accuracy: 0.7702
Epoch 14: val_accuracy improved from 0.84138 to 0.84949, saving model to Model.h5
307/307 [==============================] - 2s 7ms/step - loss: 0.6105 - accuracy: 0.7703 - val_loss: 0.4056 - v
al_accuracy: 0.8495
Epoch 15/30
298/307 [============================>.] - ETA: 0s - loss: 0.5971 - accuracy: 0.7741
Epoch 15: val_accuracy improved from 0.84949 to 0.85577, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.5968 - accuracy: 0.7743 - val_loss: 0.3921 - v
al_accuracy: 0.8558
Epoch 16/30
296/307 [============================>.] - ETA: 0s - loss: 0.5778 - accuracy: 0.7811
Epoch 16: val_accuracy did not improve from 0.85577
307/307 [==============================] - 2s 5ms/step - loss: 0.5772 - accuracy: 0.7813 - val_loss: 0.3878 - v
al_accuracy: 0.8547
Epoch 17/30
304/307 [============================>.] - ETA: 0s - loss: 0.5594 - accuracy: 0.7885
Epoch 17: val_accuracy improved from 0.85577 to 0.86250, saving model to Model.h5
307/307 [==============================] - 2s 5ms/step - loss: 0.5599 - accuracy: 0.7883 - val_loss: 0.3720 - v
al_accuracy: 0.8625
Epoch 18/30
298/307 [============================>.] - ETA: 0s - loss: 0.5561 - accuracy: 0.7911
Epoch 18: val_accuracy did not improve from 0.86250
307/307 [==============================] - 2s 5ms/step - loss: 0.5547 - accuracy: 0.7918 - val_loss: 0.3713 - v
al_accuracy: 0.8612
Epoch 19/30
306/307 [============================>.] - ETA: 0s - loss: 0.5395 - accuracy: 0.7950
Epoch 19: val_accuracy improved from 0.86250 to 0.86648, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.5395 - accuracy: 0.7950 - val_loss: 0.3600 - v
al_accuracy: 0.8665
Epoch 20/30
304/307 [============================>.] - ETA: 0s - loss: 0.5252 - accuracy: 0.8009
Epoch 20: val_accuracy improved from 0.86648 to 0.86939, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.5252 - accuracy: 0.8010 - val_loss: 0.3504 - v
al_accuracy: 0.8694
Epoch 21/30
296/307 [============================>.] - ETA: 0s - loss: 0.5187 - accuracy: 0.8035
Epoch 21: val_accuracy improved from 0.86939 to 0.87209, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.5184 - accuracy: 0.8038 - val_loss: 0.3450 - v
al_accuracy: 0.8721
Epoch 22/30
307/307 [==============================] - ETA: 0s - loss: 0.5066 - accuracy: 0.8081
Epoch 22: val_accuracy improved from 0.87209 to 0.87480, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.5066 - accuracy: 0.8081 - val_loss: 0.3386 - v
al_accuracy: 0.8748
Epoch 23/30
303/307 [============================>.] - ETA: 0s - loss: 0.4957 - accuracy: 0.8117
Epoch 23: val_accuracy improved from 0.87480 to 0.87806, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.4957 - accuracy: 0.8117 - val_loss: 0.3347 - v
al_accuracy: 0.8781
Epoch 24/30
305/307 [============================>.] - ETA: 0s - loss: 0.4873 - accuracy: 0.8157
Epoch 24: val_accuracy improved from 0.87806 to 0.87867, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.4873 - accuracy: 0.8158 - val_loss: 0.3270 - v
al_accuracy: 0.8787
Epoch 25/30
299/307 [============================>.] - ETA: 0s - loss: 0.4757 - accuracy: 0.8186
Epoch 25: val_accuracy improved from 0.87867 to 0.88158, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.4761 - accuracy: 0.8185 - val_loss: 0.3214 - v
al_accuracy: 0.8816
Epoch 26/30
303/307 [============================>.] - ETA: 0s - loss: 0.4699 - accuracy: 0.8211
Epoch 26: val_accuracy improved from 0.88158 to 0.88383, saving model to Model.h5
307/307 [==============================] - 2s 5ms/step - loss: 0.4697 - accuracy: 0.8211 - val_loss: 0.3157 - v
al_accuracy: 0.8838
Epoch 27/30
307/307 [==============================] - ETA: 0s - loss: 0.4639 - accuracy: 0.8231
Epoch 27: val_accuracy did not improve from 0.88383
307/307 [==============================] - 2s 5ms/step - loss: 0.4639 - accuracy: 0.8231 - val_loss: 0.3152 - v
al_accuracy: 0.8819
Epoch 28/30
298/307 [============================>.] - ETA: 0s - loss: 0.4560 - accuracy: 0.8251
Epoch 28: val_accuracy improved from 0.88383 to 0.88684, saving model to Model.h5
307/307 [==============================] - 2s 5ms/step - loss: 0.4566 - accuracy: 0.8251 - val_loss: 0.3096 - v
al_accuracy: 0.8868
Epoch 29/30
296/307 [============================>.] - ETA: 0s - loss: 0.4481 - accuracy: 0.8295
Epoch 29: val_accuracy improved from 0.88684 to 0.88857, saving model to Model.h5
307/307 [==============================] - 2s 5ms/step - loss: 0.4480 - accuracy: 0.8297 - val_loss: 0.3042 - v
al_accuracy: 0.8886
Epoch 30/30
298/307 [============================>.] - ETA: 0s - loss: 0.4429 - accuracy: 0.8329
Epoch 30: val_accuracy improved from 0.88857 to 0.89107, saving model to Model.h5
307/307 [==============================] - 2s 5ms/step - loss: 0.4429 - accuracy: 0.8332 - val_loss: 0.2990 - v
al_accuracy: 0.8911
```

Out[27]:
```
<keras.callbacks.History at 0x1e6311541c0>
```

In [28]:
```
history =model.fit(df, target, validation_split=0.2, epochs=30, batch_size=256, shuffle=True, callbacks=[save_b
```

```
Epoch 1/30
296/307 [============================>.] - ETA: 0s - loss: 0.4330 - accuracy: 0.8350
Epoch 1: val_accuracy improved from 0.89107 to 0.89122, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.4332 - accuracy: 0.8351 - val_loss: 0.2952 - v
al_accuracy: 0.8912
Epoch 2/30
306/307 [============================>.] - ETA: 0s - loss: 0.4254 - accuracy: 0.8375
Epoch 2: val_accuracy improved from 0.89122 to 0.89235, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.4255 - accuracy: 0.8376 - val_loss: 0.2906 - v
al_accuracy: 0.8923
Epoch 3/30
307/307 [==============================] - ETA: 0s - loss: 0.4215 - accuracy: 0.8380
Epoch 3: val_accuracy did not improve from 0.89235
307/307 [==============================] - 2s 6ms/step - loss: 0.4215 - accuracy: 0.8380 - val_loss: 0.2885 - v
al_accuracy: 0.8923
Epoch 4/30
303/307 [============================>.] - ETA: 0s - loss: 0.4138 - accuracy: 0.8416
Epoch 4: val_accuracy improved from 0.89235 to 0.89495, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.4138 - accuracy: 0.8416 - val_loss: 0.2821 - v
al_accuracy: 0.8949
Epoch 5/30
299/307 [============================>.] - ETA: 0s - loss: 0.4117 - accuracy: 0.8417
Epoch 5: val_accuracy improved from 0.89495 to 0.89730, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.4115 - accuracy: 0.8418 - val_loss: 0.2784 - v
al_accuracy: 0.8973
Epoch 6/30
306/307 [============================>.] - ETA: 0s - loss: 0.4085 - accuracy: 0.8447
Epoch 6: val_accuracy improved from 0.89730 to 0.89750, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.4083 - accuracy: 0.8448 - val_loss: 0.2768 - v
al_accuracy: 0.8975
Epoch 7/30
301/307 [============================>.] - ETA: 0s - loss: 0.4015 - accuracy: 0.8463
Epoch 7: val_accuracy improved from 0.89750 to 0.89801, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.4015 - accuracy: 0.8463 - val_loss: 0.2735 - v
al_accuracy: 0.8980
Epoch 8/30
300/307 [============================>.] - ETA: 0s - loss: 0.3972 - accuracy: 0.8486
Epoch 8: val_accuracy did not improve from 0.89801
307/307 [==============================] - 2s 6ms/step - loss: 0.3974 - accuracy: 0.8484 - val_loss: 0.2728 - v
al_accuracy: 0.8978
Epoch 9/30
307/307 [==============================] - ETA: 0s - loss: 0.3915 - accuracy: 0.8506
Epoch 9: val_accuracy improved from 0.89801 to 0.90092, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.3915 - accuracy: 0.8506 - val_loss: 0.2680 - v
al_accuracy: 0.9009
Epoch 10/30
306/307 [============================>.] - ETA: 0s - loss: 0.3876 - accuracy: 0.8512
Epoch 10: val_accuracy improved from 0.90092 to 0.90153, saving model to Model.h5
307/307 [==============================] - 2s 7ms/step - loss: 0.3877 - accuracy: 0.8511 - val_loss: 0.2649 - v
al_accuracy: 0.9015
Epoch 11/30
302/307 [============================>.] - ETA: 0s - loss: 0.3840 - accuracy: 0.8526
Epoch 11: val_accuracy did not improve from 0.90153
307/307 [==============================] - 2s 6ms/step - loss: 0.3835 - accuracy: 0.8529 - val_loss: 0.2644 - v
al_accuracy: 0.9009
Epoch 12/30
304/307 [============================>.] - ETA: 0s - loss: 0.3782 - accuracy: 0.8550
Epoch 12: val_accuracy improved from 0.90153 to 0.90327, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.3781 - accuracy: 0.8550 - val_loss: 0.2597 - v
al_accuracy: 0.9033
Epoch 13/30
306/307 [============================>.] - ETA: 0s - loss: 0.3717 - accuracy: 0.8578
Epoch 13: val_accuracy did not improve from 0.90327
307/307 [==============================] - 2s 6ms/step - loss: 0.3716 - accuracy: 0.8579 - val_loss: 0.2587 - v
al_accuracy: 0.9031
Epoch 14/30
301/307 [============================>.] - ETA: 0s - loss: 0.3675 - accuracy: 0.8599
Epoch 14: val_accuracy did not improve from 0.90327
307/307 [==============================] - 2s 7ms/step - loss: 0.3677 - accuracy: 0.8599 - val_loss: 0.2569 - v
al_accuracy: 0.9033
Epoch 15/30
303/307 [============================>.] - ETA: 0s - loss: 0.3684 - accuracy: 0.8599
Epoch 15: val_accuracy improved from 0.90327 to 0.90776, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.3679 - accuracy: 0.8600 - val_loss: 0.2501 - v
al_accuracy: 0.9078
Epoch 16/30
302/307 [============================>.] - ETA: 0s - loss: 0.3643 - accuracy: 0.8618
Epoch 16: val_accuracy did not improve from 0.90776
307/307 [==============================] - 2s 6ms/step - loss: 0.3641 - accuracy: 0.8619 - val_loss: 0.2508 - v
al_accuracy: 0.9047
Epoch 17/30
298/307 [============================>.] - ETA: 0s - loss: 0.3607 - accuracy: 0.8616
Epoch 17: val_accuracy did not improve from 0.90776
307/307 [==============================] - 2s 6ms/step - loss: 0.3606 - accuracy: 0.8615 - val_loss: 0.2496 - v
al_accuracy: 0.9052
Epoch 18/30
301/307 [============================>.] - ETA: 0s - loss: 0.3577 - accuracy: 0.8624
Epoch 18: val_accuracy did not improve from 0.90776
307/307 [==============================] - 2s 6ms/step - loss: 0.3573 - accuracy: 0.8626 - val_loss: 0.2469 - v
al_accuracy: 0.9071
Epoch 19/30
298/307 [============================>.] - ETA: 0s - loss: 0.3518 - accuracy: 0.8655
Epoch 19: val_accuracy did not improve from 0.90776
307/307 [==============================] - 2s 6ms/step - loss: 0.3519 - accuracy: 0.8656 - val_loss: 0.2444 - v
al_accuracy: 0.9076
Epoch 20/30
306/307 [============================>.] - ETA: 0s - loss: 0.3454 - accuracy: 0.8670
Epoch 20: val_accuracy did not improve from 0.90776
307/307 [==============================] - 2s 6ms/step - loss: 0.3456 - accuracy: 0.8669 - val_loss: 0.2453 - v
al_accuracy: 0.9076
Epoch 21/30
298/307 [============================>.] - ETA: 0s - loss: 0.3392 - accuracy: 0.8702
Epoch 21: val_accuracy improved from 0.90776 to 0.90929, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.3392 - accuracy: 0.8702 - val_loss: 0.2395 - v
al_accuracy: 0.9093
Epoch 22/30
305/307 [============================>.] - ETA: 0s - loss: 0.3394 - accuracy: 0.8685
Epoch 22: val_accuracy improved from 0.90929 to 0.90985, saving model to Model.h5
307/307 [==============================] - 2s 7ms/step - loss: 0.3394 - accuracy: 0.8686 - val_loss: 0.2389 - v
al_accuracy: 0.9098
Epoch 23/30
306/307 [============================>.] - ETA: 0s - loss: 0.3368 - accuracy: 0.8714
Epoch 23: val_accuracy improved from 0.90985 to 0.91082, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.3368 - accuracy: 0.8713 - val_loss: 0.2375 - v
al_accuracy: 0.9108
Epoch 24/30
307/307 [==============================] - ETA: 0s - loss: 0.3367 - accuracy: 0.8709
Epoch 24: val_accuracy did not improve from 0.91082
307/307 [==============================] - 2s 7ms/step - loss: 0.3367 - accuracy: 0.8709 - val_loss: 0.2354 - v
al_accuracy: 0.9104
Epoch 25/30
303/307 [============================>.] - ETA: 0s - loss: 0.3333 - accuracy: 0.8706
Epoch 25: val_accuracy did not improve from 0.91082
307/307 [==============================] - 2s 6ms/step - loss: 0.3333 - accuracy: 0.8704 - val_loss: 0.2351 - v
al_accuracy: 0.9106
Epoch 26/30
306/307 [============================>.] - ETA: 0s - loss: 0.3309 - accuracy: 0.8732
Epoch 26: val_accuracy improved from 0.91082 to 0.91403, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.3309 - accuracy: 0.8732 - val_loss: 0.2312 - v
al_accuracy: 0.9140
Epoch 27/30
303/307 [============================>.] - ETA: 0s - loss: 0.3284 - accuracy: 0.8744
Epoch 27: val_accuracy did not improve from 0.91403
307/307 [==============================] - 2s 7ms/step - loss: 0.3278 - accuracy: 0.8747 - val_loss: 0.2309 - v
al_accuracy: 0.9133
Epoch 28/30
299/307 [============================>.] - ETA: 0s - loss: 0.3242 - accuracy: 0.8743
Epoch 28: val_accuracy did not improve from 0.91403
307/307 [==============================] - 2s 6ms/step - loss: 0.3245 - accuracy: 0.8742 - val_loss: 0.2288 - v
al_accuracy: 0.9131
Epoch 29/30
297/307 [============================>.] - ETA: 0s - loss: 0.3239 - accuracy: 0.8753
Epoch 29: val_accuracy improved from 0.91403 to 0.91474, saving model to Model.h5
307/307 [==============================] - 2s 6ms/step - loss: 0.3244 - accuracy: 0.8751 - val_loss: 0.2286 - v
al_accuracy: 0.9147
Epoch 30/30
296/307 [============================>.] - ETA: 0s - loss: 0.3206 - accuracy: 0.8761
Epoch 30: val_accuracy did not improve from 0.91474
307/307 [==============================] - 2s 6ms/step - loss: 0.3200 - accuracy: 0.8762 - val_loss: 0.2261 - v
al_accuracy: 0.9147
```

In [29]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(30)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



In [30]:
```
#Model Prediction
pred = model.predict(df, verbose=1)
pred = np.argmax(pred, axis=1)
```

```
3063/3063 [==============================] - 6s 2ms/step
```

In [31]:
```
#Predicting of Clusters
pd.DataFrame(pred)[0].value_counts()
```

Out[31]:
```
10    7294
5     6518
1     6258
2     5965
8     5884
11    5884
16    5529
19    5404
15    5364
14    5206
0     5133
18    4806
12    4617
9     4293
4     4107
13    3831
6     3410
3     3212
7     2824
17    2461
Name: 0, dtype: int64
```

In [32]:
```
sample['Predicted'] = kmeans.labels_
sample.to_csv('submission.csv', index=False)
```

In [33]:
```
sample
```

Out[33]:

	Id	Predicted
0	0	8
1	1	7
2	2	18
3	3	11
4	4	12
...	...	...
97995	97995	5
97996	97996	12
97997	97997	2
97998	97998	13
97999	97999	10

98000 rows × 2 columns

Working in other clustering algorithms....

In [ ]: