```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn import preprocessing
import plotly.express as px
from sklearn.preprocessing import StandardScaler
%matplotlib inline
plt.style.use('dark_background')
```

```
data = pd.read_csv('/content/drive/MyDrive/Dataset/cancer.csv')
data.drop(['id', 'diagnosis', 'Unnamed: 32'], axis=1, inplace=True) # Removing t
```

```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data) # Scaling the dataset to reduce dataset
# statistics of scaled data
pd.DataFrame(data_scaled).describe()
```

	0	1	2	3	
count	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.69000
mean	-1.373633e-16	6.868164e-17	-1.248757e-16	-2.185325e-16	-8.36667
std	1.000880e+00	1.000880e+00	1.000880e+00	1.000880e+00	1.00088
min	-2.029648e+00	-2.229249e+00	-1.984504e+00	-1.454443e+00	-3.112085
25%	-6.893853e-01	-7.259631e-01	-6.919555e-01	-6.671955e-01	-7.10962
50%	-2.150816e-01	-1.046362e-01	-2.359800e-01	-2.951869e-01	-3.48910
75%	4.693926e-01	5.841756e-01	4.996769e-01	3.635073e-01	6.36199
max	3.971288e+00	4.651889e+00	3.976130e+00	5.250529e+00	4.770911

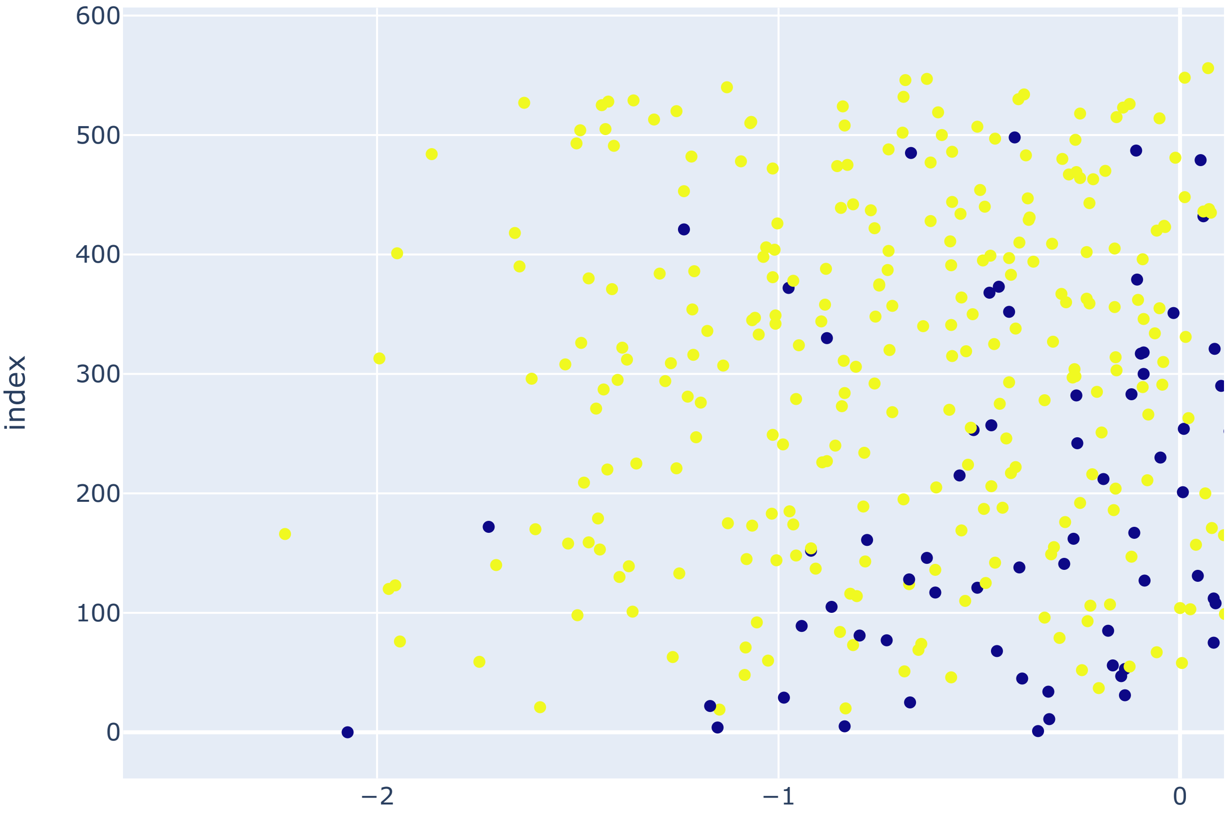
8 rows × 30 columns

```
class KMeansClustering:
    def __init__(self, X, num_clusters):
        self.K = num_clusters # cluster number
        self.max_iterations = 100 # max iteration. don't want to run inf time
        self.num_examples, self.num_features = X.shape # num of examples, num of
        self.plot_figure = True # plot figure

    # randomly initialize centroids
```

✓ 0s completed at 11:18 AM ● ✕

```
        centroids = np.zeros((self.K, self.num_features)) # row , column full wi
        for k in range(self.K): # iterations of
            centroid = X[np.random.choice(range(self.num_examples))] # random ce
            centroids[k] = centroid
        return centroids # return random centroids

    # create cluster Function
    def create_cluster(self, X, centroids):
        clusters = [[] for _ in range(self.K)]
        for point_idx, point in enumerate(X):
            closest_centroid = np.argmin(
                np.sqrt(np.sum((point-centroids)**2, axis=1))
            ) # closest centroid using euler distance equation(calculate distanc
            clusters[closest_centroid].append(point_idx)
        return clusters

    # new centroids
    def calculate_new_centroids(self, cluster, X):
        centroids = np.zeros((self.K, self.num_features)) # row , column full wi
        for idx, cluster in enumerate(cluster):
            new_centroid = np.mean(X[cluster], axis=0) # find the value for new
            centroids[idx] = new_centroid
        return centroids

    # prediction
    def predict_cluster(self, clusters, X):
        y_pred = np.zeros(self.num_examples) # row1 fillup with zero
        for cluster_idx, cluster in enumerate(clusters):
            for sample_idx in cluster:
                y_pred[sample_idx] = cluster_idx
        return y_pred

    # plotinng scatter plot
    def plot_fig(self, X, y):
        fig = px.scatter(X[:, 0], X[:, 1], color=y)
        fig.show() # visualize


    # fit data
    def fit(self, X):
        centroids = self.initialize_random_centroids(X) # initialize random cent
        for _ in range(self.max_iterations):
            clusters = self.create_cluster(X, centroids) # create cluster
            previous_centroids = centroids
            centroids = self.calculate_new_centroids(clusters, X) # calculate ne
            diff = centroids - previous_centroids # calculate difference
            if not diff.any():
                break
        y_pred = self.predict_cluster(clusters, X) # predict function
        if self.plot_figure: # if true
            self.plot_fig(X, y_pred) # plot function
        return y_pred
```

```
if __name__ == "__main__":
   np.random.seed(10)
   num_clusters = 2 # num of cluster
   X = np.array(data_scaled.astype(float))
  #  print(data.shape)
   Kmeans = KMeansClustering(X, num_clusters)
   y_pred = Kmeans.fit(X)
  #  print(y_pred)
```



```
print(y_pred) # Printing the predictions of cluster points
```

```
    [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 1. 0. 0.
     0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0.
     1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1.
     0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 1. 1. 1. 0. 0.
     1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1.
     1. 0. 0. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1.
     1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 0.
     0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
     1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 0. 0.
     1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1. 0. 0. 1. 0.
     1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1.
     0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 1.
     1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
```

```
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 0.
 1. 0. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 0.
 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 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. 0. 0. 0. 0. 0. 0. 1.]
```

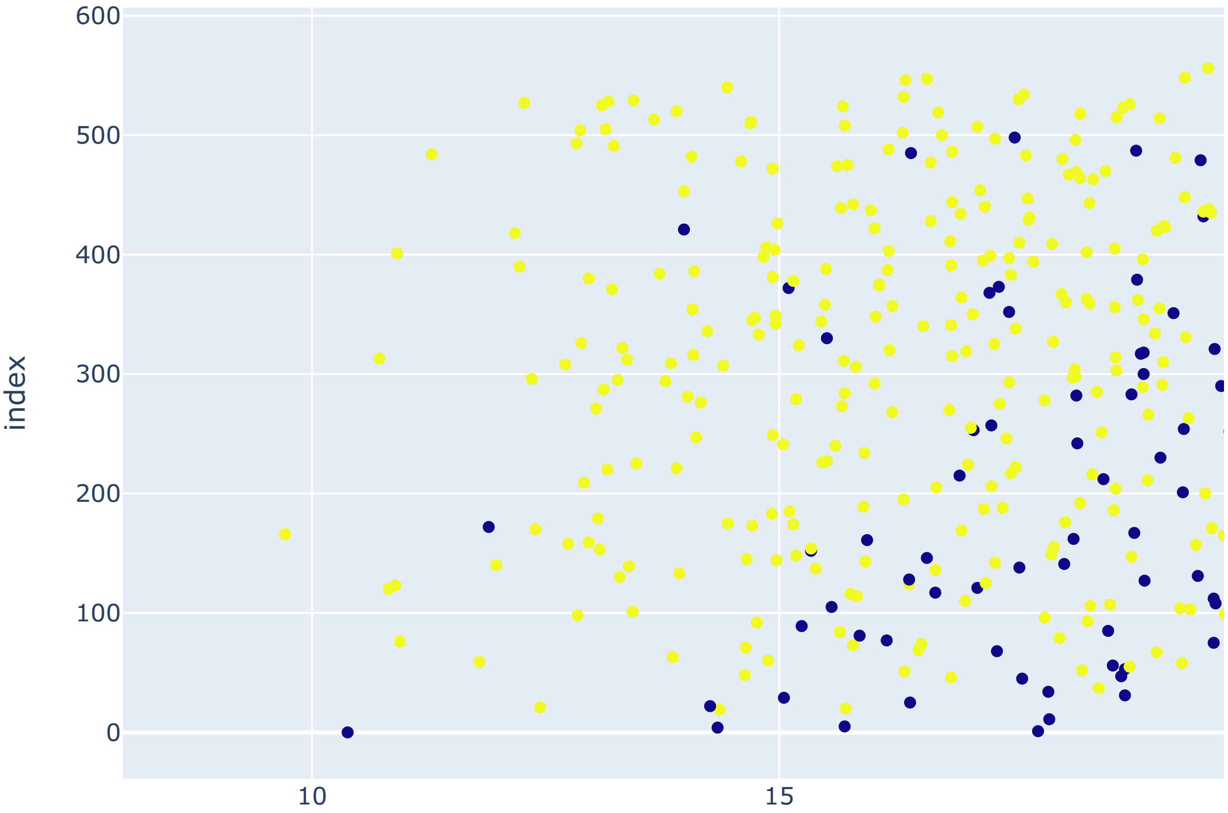
```
frame = pd.DataFrame(data)
frame['cluster'] = y_pred
frame['cluster'].value_counts() # Printing the cluster points in each cluster.
```

```
1.0    381
0.0    188
Name: cluster, dtype: int64
```

```
dataset = data.copy()
dataset['cluster'] = y_pred
```

```
fig = px.scatter_3d(dataset, x="radius_mean", y="texture_mean", z="perimeter_mea
fig.show() # ploting 3D scatter plot for better visualizing of cluster points.
```

```
fig = px.scatter(dataset['radius_mean'], dataset['texture_mean'], color=dataset[
fig.show() # Ploting scatter plot using radius_mean and texture_mean feature vec
```


index