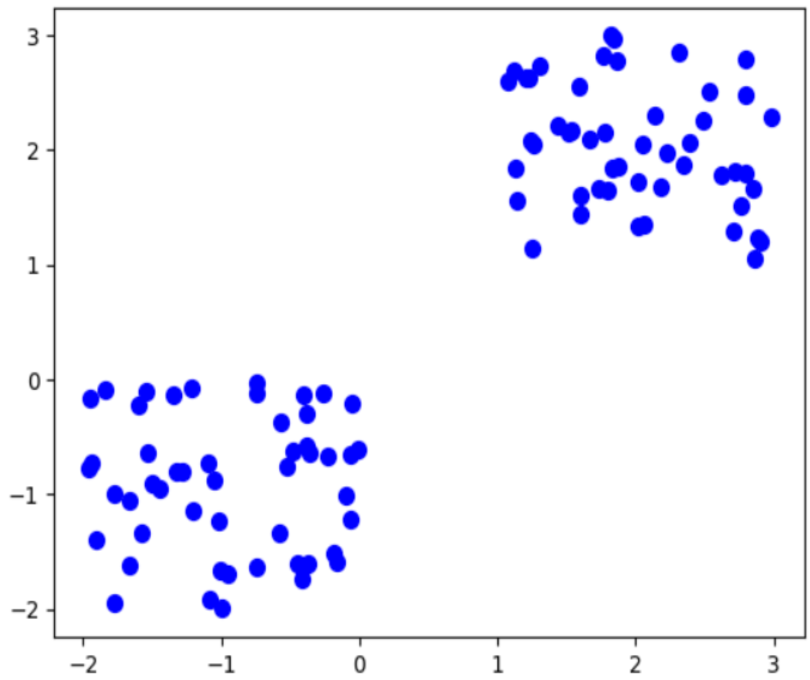
Aim: Write a program to cluster a set of points using K-means. Consider, K=3, clusters. Consider Euclidean distance as the distance measure. Randomly initialize a cluster mean as one of the data points. Iterate for 10 iterations. After iterations are over, print the final cluster means for each of the clusters. Use the ground truth cluster label present in the data set to compute and print the Jacquard distance of the obtained clusters with the ground truth clusters for each of the three clusters.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

[2]:
```
X= -2 * np.random.rand(100,2)
X1 = 1 + 2 * np.random.rand(50,2)
X[50:100, :]= X1
plt.scatter(X[ : , 0], X[ :, 1], s = 50, c = 'b')
plt.show()
```



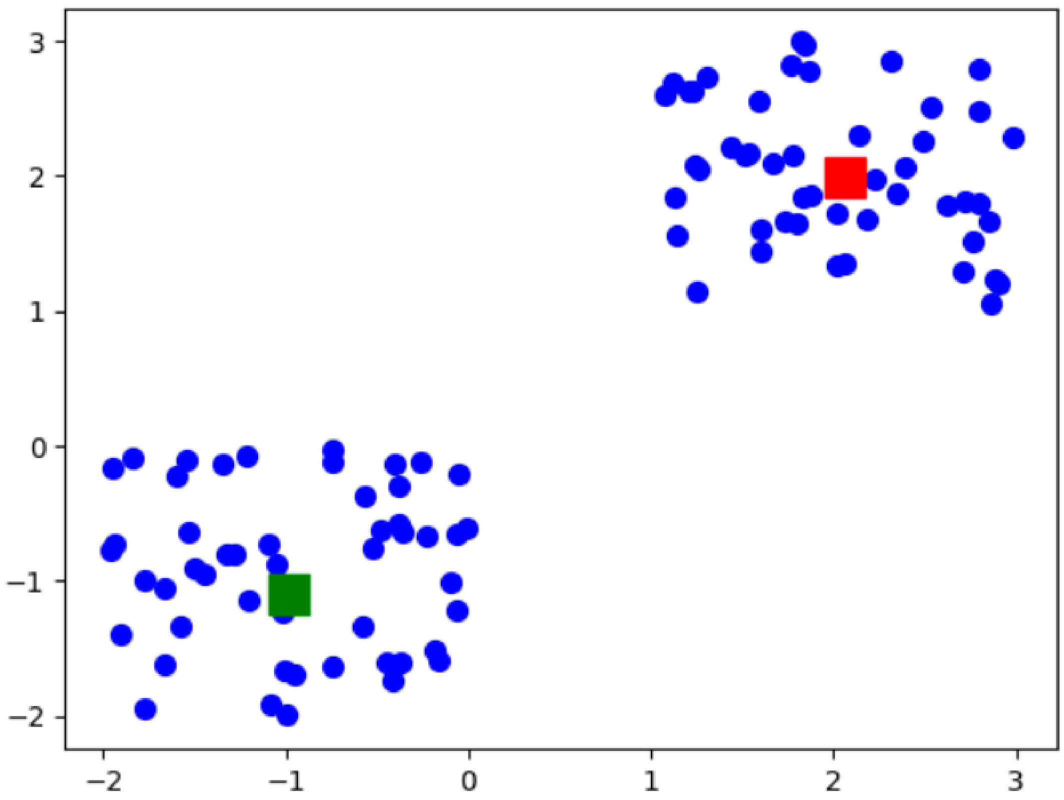
[3]:
```
from sklearn.cluster import KMeans
KMeans =  KMeans(n_clusters=2, n_init=10)
KMeans.fit(X)
```

[3]:
```
▾ KMeans
KMeans(n_clusters=2, n_init=10)
```

[4]:
```
KMeans.cluster_centers_
```

[4]:
```
array([[-0.95128646, -0.89978266],
       [ 1.99434749,  2.03518682]])
```

[5]:
```
plt.scatter(X[ : , 0], X[ : , 1], s = 50, c = 'b')
plt.scatter(-0.98362533, -1.09744804 , s=200, c='g', marker='s')
plt.scatter(2.06708619, 1.99208978, s=200, c='r', marker='s')
plt.show()
```



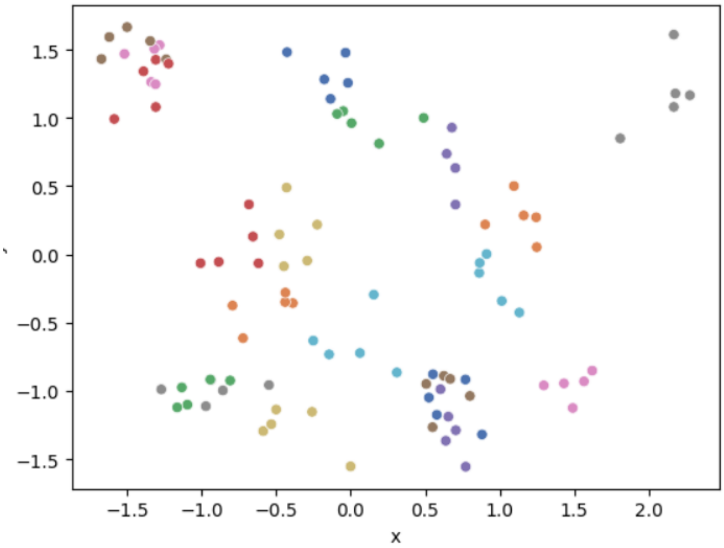
[6]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from numpy.random import uniform
from sklearn.datasets import make_blobs
import seaborn as sns
import random
```

[7]:
```
def euclidean (point, data):
    return np.sqrt(np.sum((point - data)**2, axis=1))
```

```
[8]: class KMeans:
    def __init__(self, n_clusters=8, max_iter=300):
        # Initialization
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X_train):
        self.centroids = [random.choice(X_train)]
        for _ in range(self.n_clusters - 1):
            # Calculate distances from points to the centroids
            dists = np.sum([euclidean(centroid, X_train) for centroid in self.centroids], axis=0)
            # Normalize the distances
            dists /= np.sum(dists)
            # Choose remaining points based on their distances
            new_centroid_idx = np.random.choice(range(len(X_train)), size=1, p=dists)
            self.centroids += [X_train[new_centroid_idx[0]]]

        # Iterative adjustment of centroids until convergence or max iterations
        iteration = 0
        prev_centroids = None
        while np.not_equal(self.centroids, prev_centroids).any() and iteration < self.max_iter:
            # Sort each datapoint, assigning to the nearest centroid
            sorted_points = [[] for _ in range(self.n_clusters)]
            for x in X_train:
                dists = euclidean(x, self.centroids)
                centroid_idx = np.argmin(dists)
                sorted_points[centroid_idx].append(x)

            # Push current centroids to previous, reassign centroids as means
            prev_centroids = self.centroids
            self.centroids = [np.mean(cluster, axis=0) if cluster else self.centroids[i] for i, cluster in enumerate(sorted_points)]

            iteration += 1
```

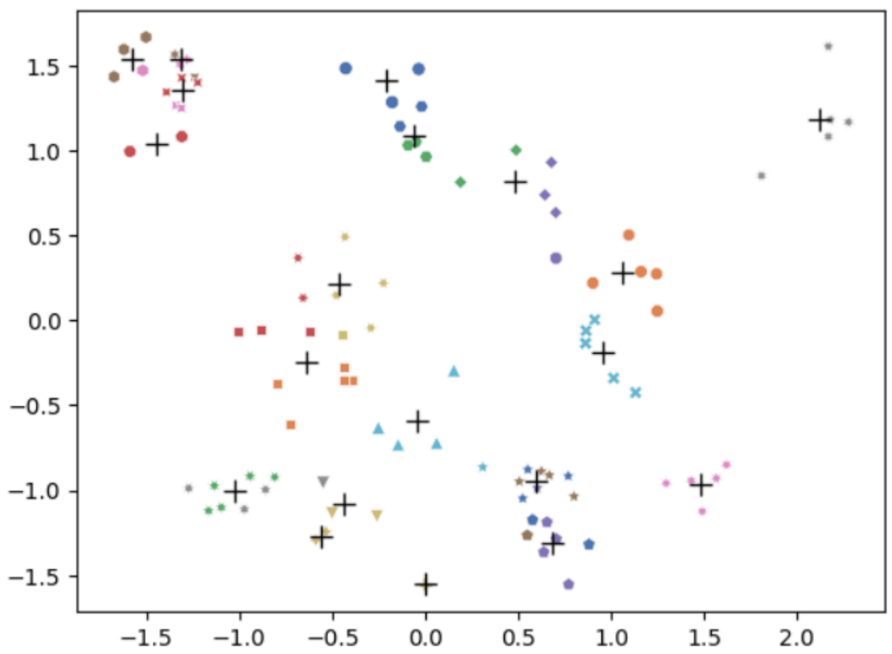
```
    def evaluate(self, X):
        centroids = []
        centroid_idxs = []
        for x in X:
            dists = euclidean(x, self.centroids)
            centroid_idx = np.argmin(dists)
            centroids.append(self.centroids[centroid_idx])
            centroid_idxs.append(centroid_idx)
        return centroids, centroid_idxs
```

[9]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

# Create a dataset with 20 clusters
centers = 20
X_train, true_labels = make_blobs(n_samples=100, centers=centers, random_state=42)

# Standardize the data
X_train = StandardScaler().fit_transform(X_train)

# Scatter plot
sns.scatterplot(x=[X[0] for X in X_train],
                y=[X[1] for X in X_train],
                hue=true_labels,
                palette="deep",
                legend=None
               )

plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



[10]:
```python
#Fit centroids to dataset
kmeans = KMeans(n_clusters=centers)
kmeans.fit(X_train)
```

[11]:
```python
#View results
class_centers, classification = kmeans.evaluate(X_train)
sns.scatterplot(x=[X[0] for X in X_train],
                y=[X[1] for X in X_train],
                hue=true_labels,
                style=classification,
                palette="deep",
                legend=None
               )
plt.plot([x for x, _ in kmeans.centroids],
         [y for _, y in kmeans.centroids],
         'k+',
         markersize=10,
         )
plt.show()
```



[12]:
```
#Import Libraries
import numpy as np # Linear algebra
import pandas as pd # data processing, CSV file 1/0 (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
df = pd.read_csv('breast-cancer.csv')
df.head ()
```

[12]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54

5 rows × 32 columns

[13]:
```
df.describe()
```

[13]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	.
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	.
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	.
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	.
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	.
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	.
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	.
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	.
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	.

8 rows × 31 columns

[14]:
```
df['diagnosis'] = df['diagnosis'].map({'B': 0, 'M': 1})
```

[15]:
```
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']
```

[16]:
```
X.head(5)
```

[16]:

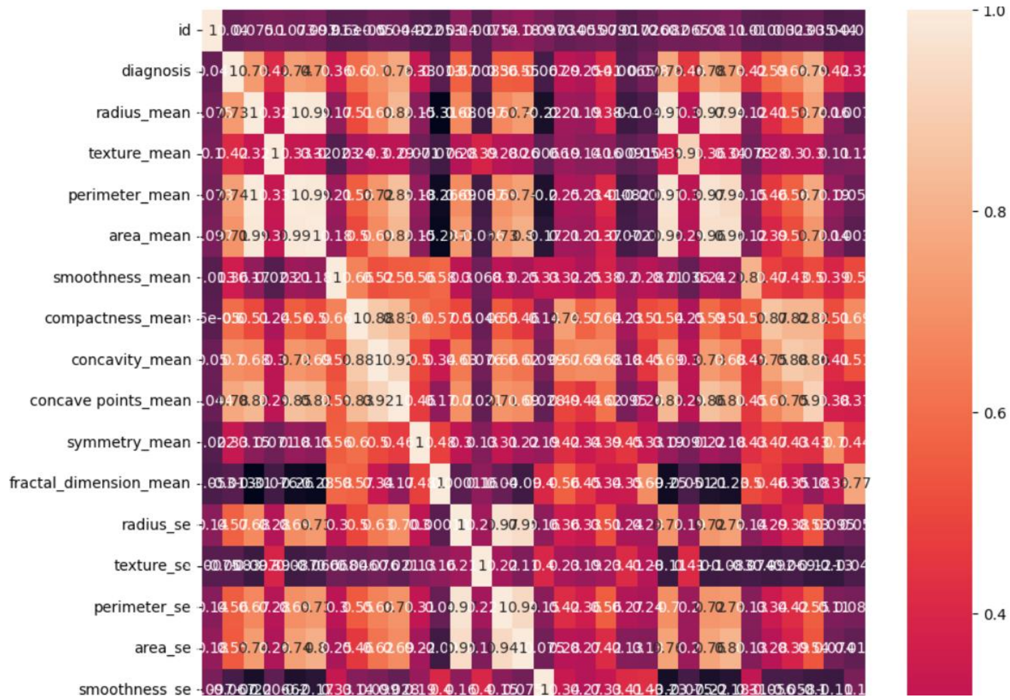
	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_
0	842302	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	
1	842517	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	
2	84300903	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	
3	84348301	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	
4	84358402	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	

5 rows × 31 columns

[17]:
```
y.head(5)
```

[17]:
```
0    1
1    1
2    1
3    1
4    1
Name: diagnosis, dtype: int64
```

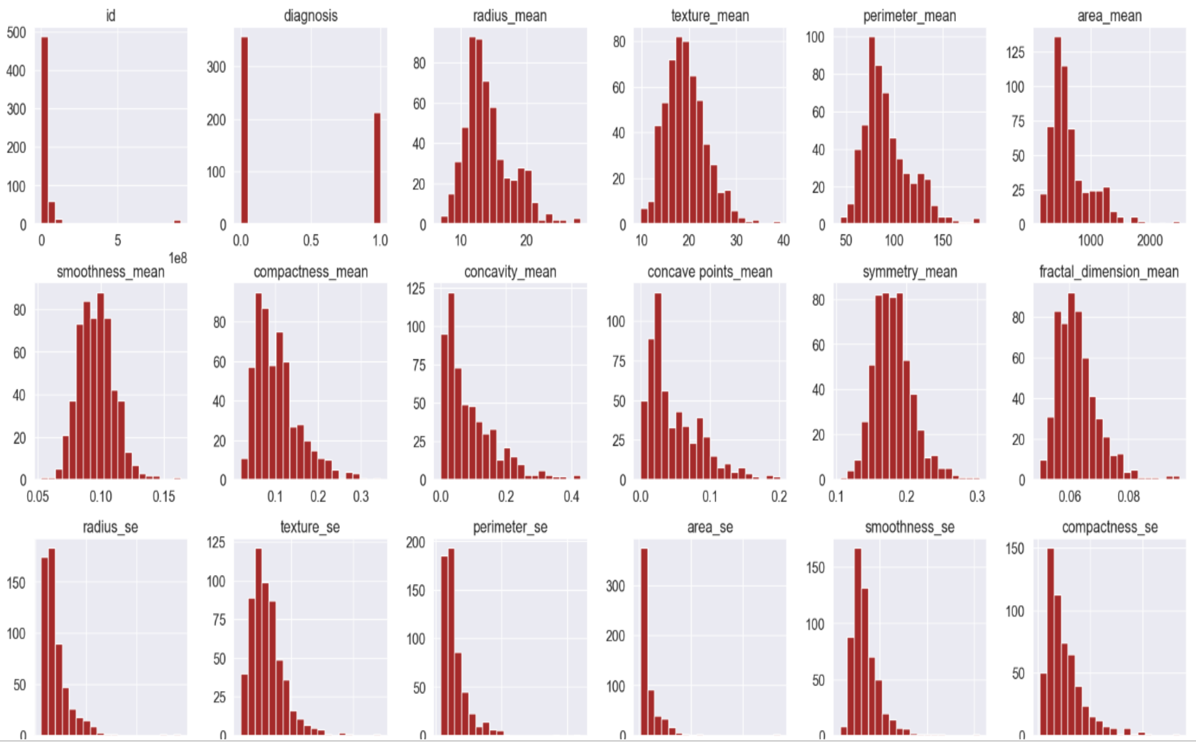
[18]:
```
import seaborn as sns
plt.figure(figsize=(10,16))
dataplot= sns.heatmap(df.corr(), annot=True)
plt.show()
```

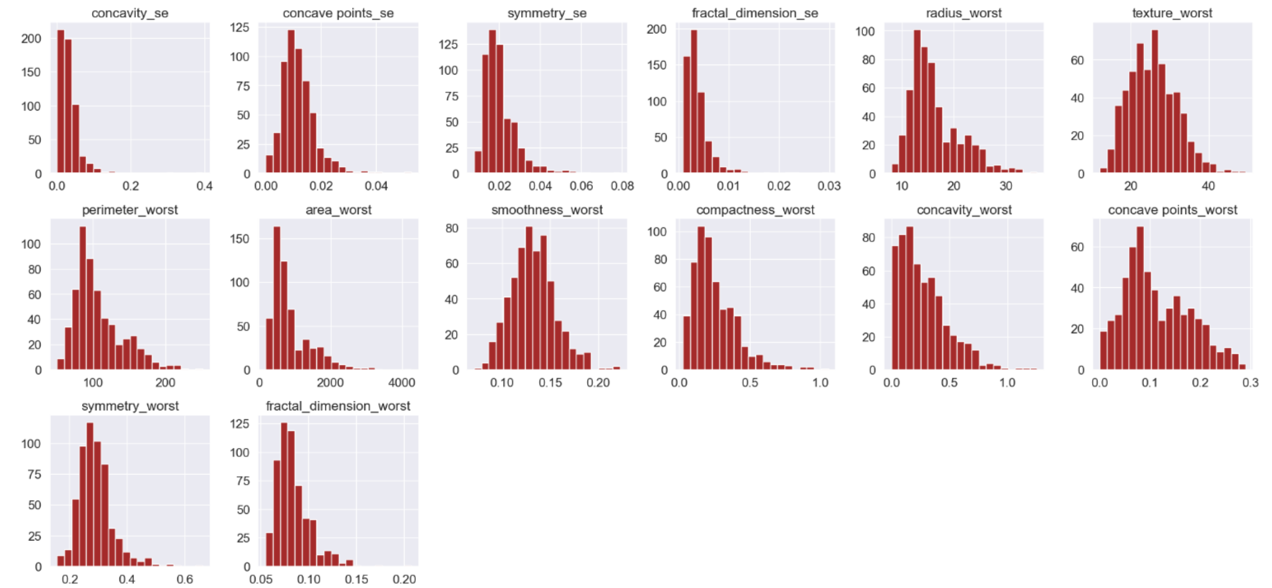


```
[19]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    int64
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
```

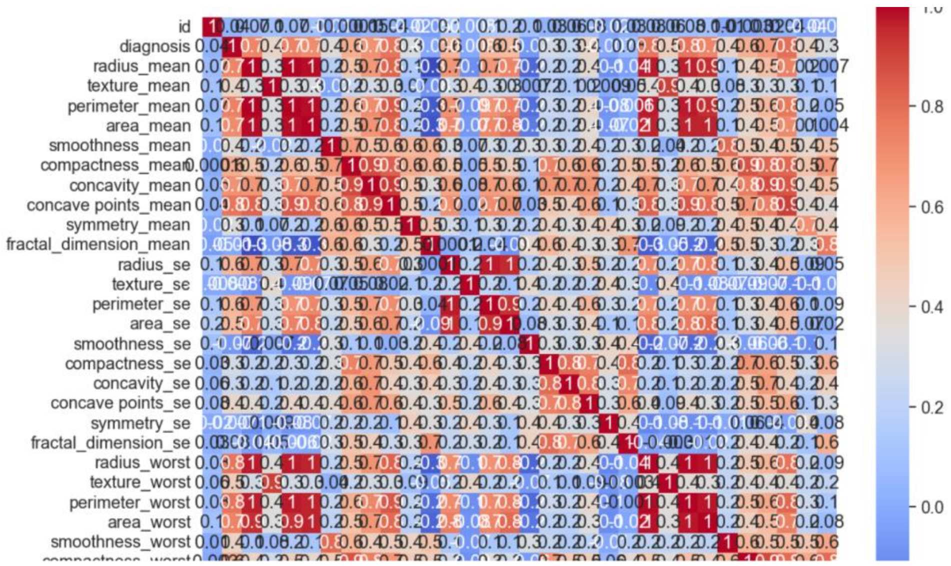
```
[20]: #Draw Hostogram For Each Feature
sns.set(style='darkgrid', font_scale=1.3, rc={'figure.figsize':(25,25)})
ax=df.hist(bins=20,color='brown')
```





[22]:

```
plt.figure(figsize=(12,10))
mask=np.tril(df.corr())
sns.heatmap(df.corr(), cmap="coolwarm", annot=True, fmt='.1g', square=True)
```

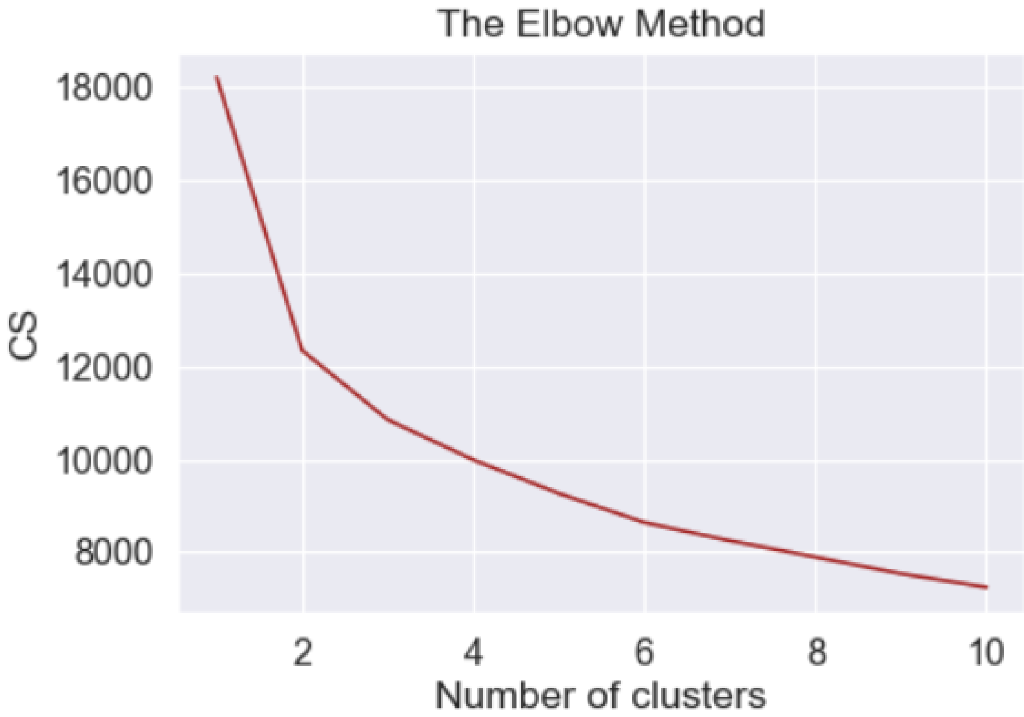


[23]:

```
sc= StandardScaler()
df=sc.fit_transform(df)
```

[24]:

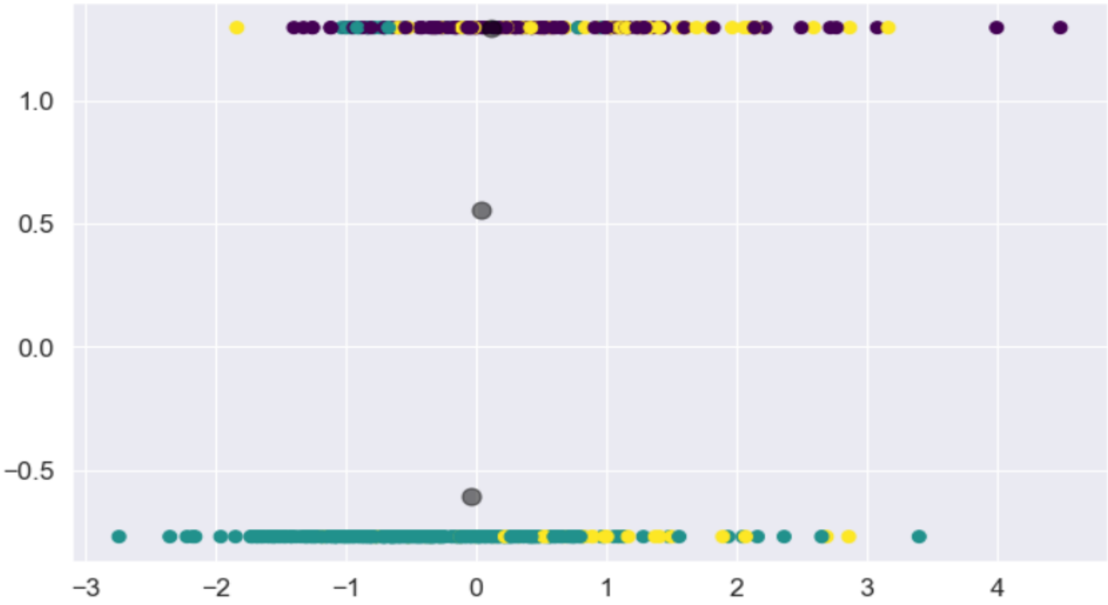
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# Assuming df is your dataset
cs = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=42)
    kmeans.fit(df)
    cs.append(kmeans.inertia_)

# Plotting outside the loop
plt.figure(figsize=(6, 4), dpi=80)
plt.plot(range(1, 11), cs, color='brown')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('CS')
plt.show()
```




```
[25]: kmeans= KMeans(n_clusters=3, init='k-means++', random_state=42, n_init=10)
Y=kmeans.fit_predict(df)
print(Y)
```

```
[0 0 0 2 0 2 0 2 2 2 1 2 0 1 2 2 1 2 0 1 1 1 2 0 0 0 2 0 2 0 0 2 0 0 2 0 2
 1 1 2 1 2 0 2 1 0 1 2 1 1 1 1 1 0 1 1 0 2 1 1 1 1 2 1 2 2 1 1 2 1 0 2 0 1
 1 0 1 0 0 1 1 2 0 0 1 0 1 0 1 2 1 1 1 1 2 0 1 1 1 2 1 1 1 1 1 2 1 1 0 1 1
 2 2 1 1 1 1 2 2 0 1 0 0 1 1 1 1 0 2 0 1 0 0 1 0 1 1 1 0 1 1 0 1 1 1 2 2 1
 1 1 1 2 2 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 1 2 1 1 1 2 2 1 1 0 0 1 1 1
 1 1 1 1 1 2 1 1 2 2 1 2 0 0 2 1 0 0 2 1 1 1 1 2 1 0 1 0 2 2 2 1 1 0 0 1 1
 1 2 1 1 1 1 1 2 0 1 1 0 1 1 0 0 1 0 1 1 2 1 0 1 1 2 1 1 0 1 0 0 0 2 0 2 0
 2 0 1 0 1 0 0 1 1 1 2 1 1 0 1 1 1 1 1 1 1 0 1 0 2 1 1 1 1 2 1 2 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 2 1 1 0 1 0 1 1 1 1 0 2 2 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 2 0 2 1 1 2 1 1 1 1 1 1 1 1 0 0 1 0 0
 2 1 0 0 1 1 2 1 1 2 1 1 1 2 1 1 1 1 2 0 1 1 0 0 1 1 1 1 1 1 2 1 1 1 1 1 1
 1 0 1 1 1 1 1 1 1 1 0 1 1 1 2 1 1 1 1 1 1 1 1 2 1 0 0 1 2 1 1 1 1 2 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 2 1 1 0 2 1 1 1 1 1 1 1 1 1 2 1
 1 1 1 1 2 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 2 1 0 2 2 1 2 1 2 1 1 2 1 1 1 0 0
 1 1 2 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 2 0 0 0 0 0 1]
```

[26]:
```
plt.figure(figsize=(10,6), dpi=80)
plt.scatter(df[:, 10], df[:, 1], c=Y, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:,0], centers[:,1], c='black', s=100, alpha=0.5);
```



Submitted By,

Name:- Shankar Singh Mahanty
Regd. No:- 2101020758
Roll No:- CSE21238
Group:- 3
Sem:- 5th
Branch:- CSE