1.) Imports

```
#Import the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
!pip install kneed
from kneed import KneeLocator
```

```
Collecting kneed
  Downloading kneed-0.8.5-py3-none-any.whl (10 kB)
Requirement already satisfied: numpy>=1.14.2 in
/usr/local/lib/python3.10/dist-packages (from kneed) (1.23.5)
Requirement already satisfied: scipy>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from kneed) (1.11.4)
Installing collected packages: kneed
Successfully installed kneed-0.8.5
```

2.) Reading Data

```
#Read from database CSV file
# Load the CSV file
from google.colab import drive
drive.mount('/content/drive')

df=pd.read_csv('/content/drive/MyDrive/data4.csv')
```

```
Mounted at /content/drive
```

```
df.head()
```

```
         id diagnosis  radius_mean  texture_mean  perimeter_mean
area_mean  \
0    842302         M        17.99         10.38          122.80
1001.0
1    842517         M        20.57         17.77          132.90
1326.0
2  84300903         M        19.69         21.25          130.00
1203.0
3  84348301         M        11.42         20.38           77.58
386.1
4  84358402         M        20.29         14.34          135.10
```

```
1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
2          0.10960           0.15990          0.1974
0.12790
3          0.14250           0.28390          0.2414
0.10520
4          0.10030           0.13280          0.1980
0.10430

   ...  radius_worst  texture_worst  perimeter_worst  area_worst  \
0  ...         25.38          17.33           184.60      2019.0
1  ...         24.99          23.41           158.80      1956.0
2  ...         23.57          25.53           152.50      1709.0
3  ...         14.91          26.50            98.87       567.7
4  ...         22.54          16.67           152.20      1575.0

   smoothness_worst  compactness_worst  concavity_worst  concave
points_worst  \
0            0.1622             0.6656           0.7119
0.2654
1            0.1238             0.1866           0.2416
0.1860
2            0.1444             0.4245           0.4504
0.2430
3            0.2098             0.8663           0.6869
0.2575
4            0.1374             0.2050           0.4000
0.1625

   symmetry_worst  fractal_dimension_worst
0          0.4601                  0.11890
1          0.2750                  0.08902
2          0.3613                  0.08758
3          0.6638                  0.17300
4          0.2364                  0.07678

[5 rows x 32 columns]
```

```
df = df.drop('id', axis=1)
df.head()
```

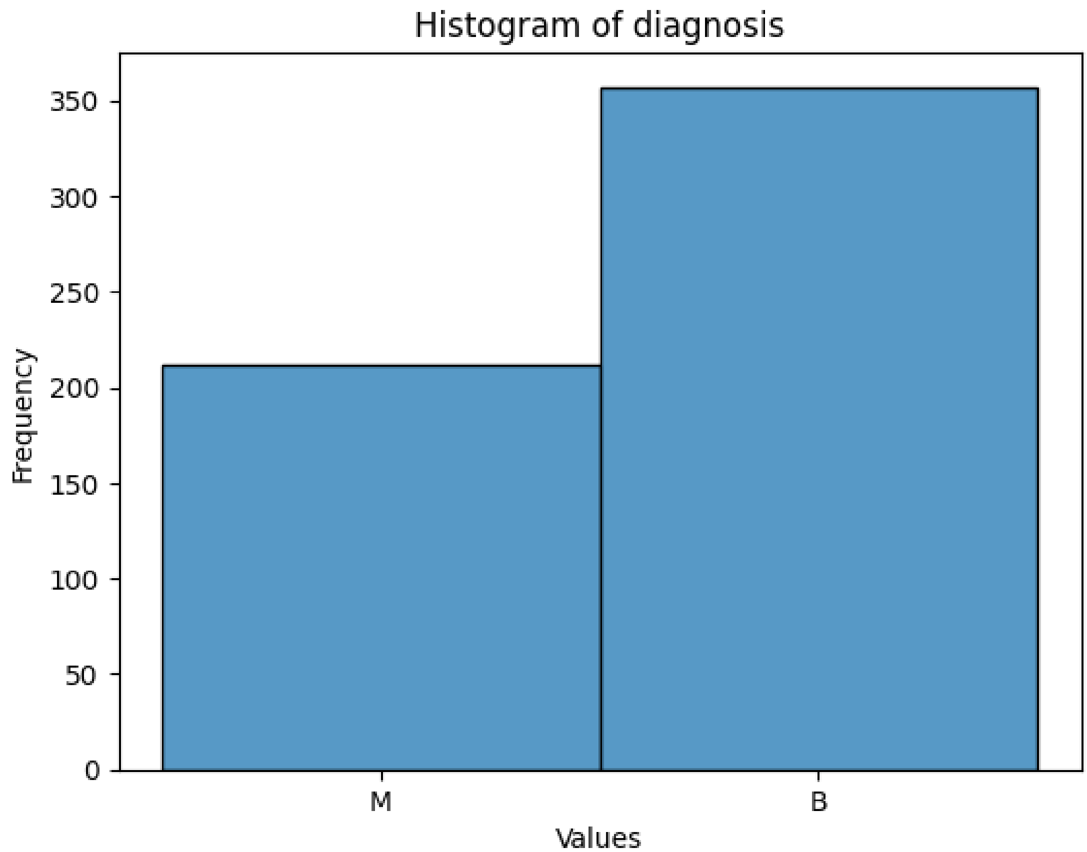
```
  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         M        17.99         10.38          122.80     1001.0
1         M        20.57         17.77          132.90     1326.0
```

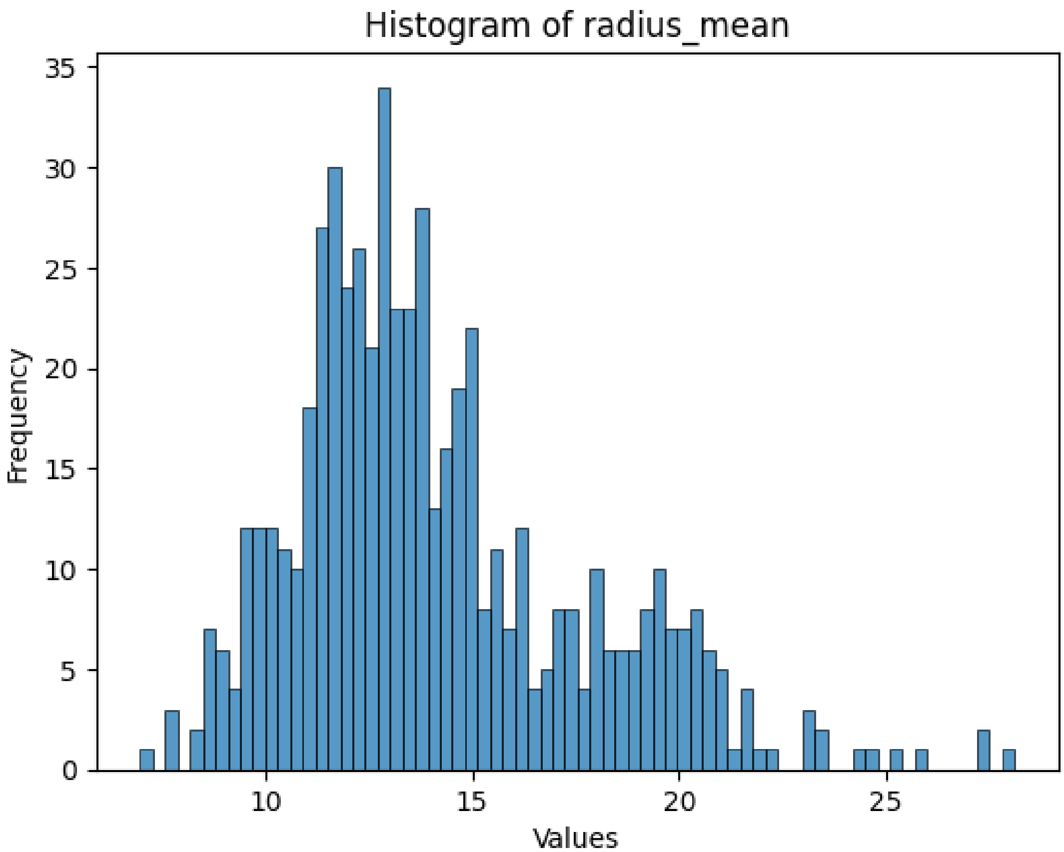
```
2         M        19.69         21.25          130.00     1203.0
3         M        11.42         20.38           77.58      386.1
4         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
2          0.10960           0.15990          0.1974
0.12790
3          0.14250           0.28390          0.2414
0.10520
4          0.10030           0.13280          0.1980
0.10430

   symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0         0.2419  ...         25.38          17.33           184.60
1         0.1812  ...         24.99          23.41           158.80
2         0.2069  ...         23.57          25.53           152.50
3         0.2597  ...         14.91          26.50            98.87
4         0.1809  ...         22.54          16.67           152.20

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   concave points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 31 columns]

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   diagnosis                569 non-null    object
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
```

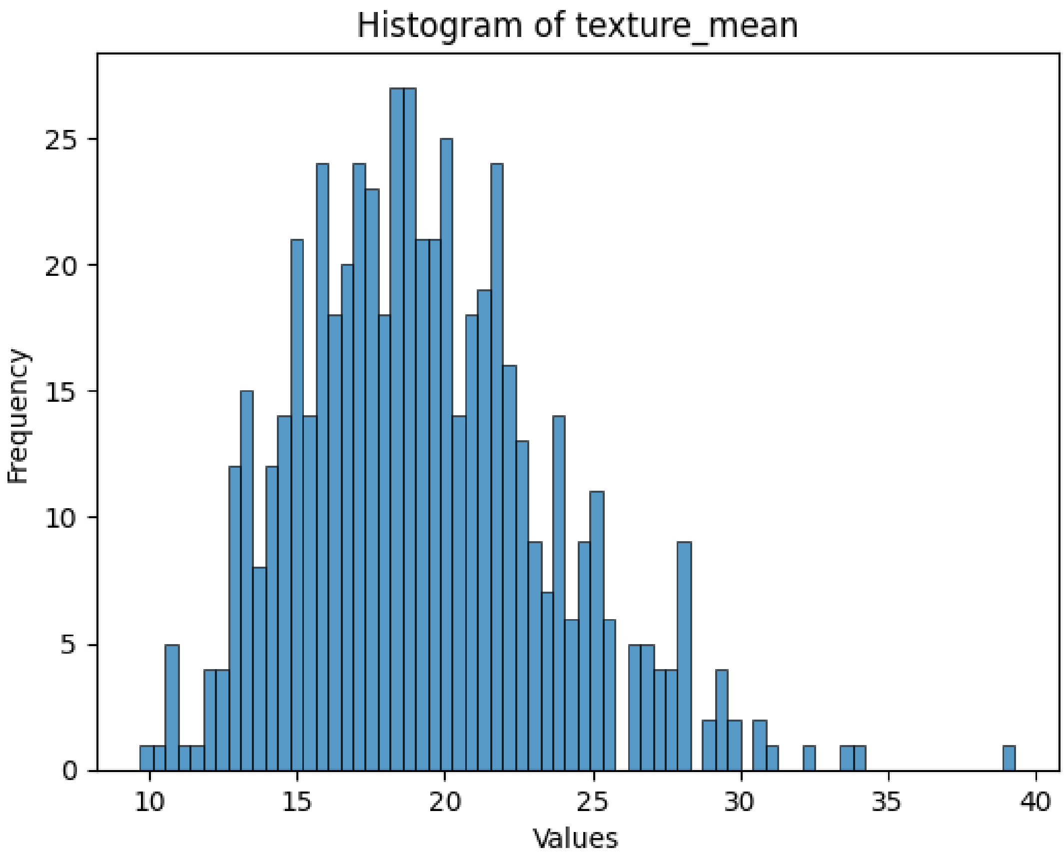
```
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

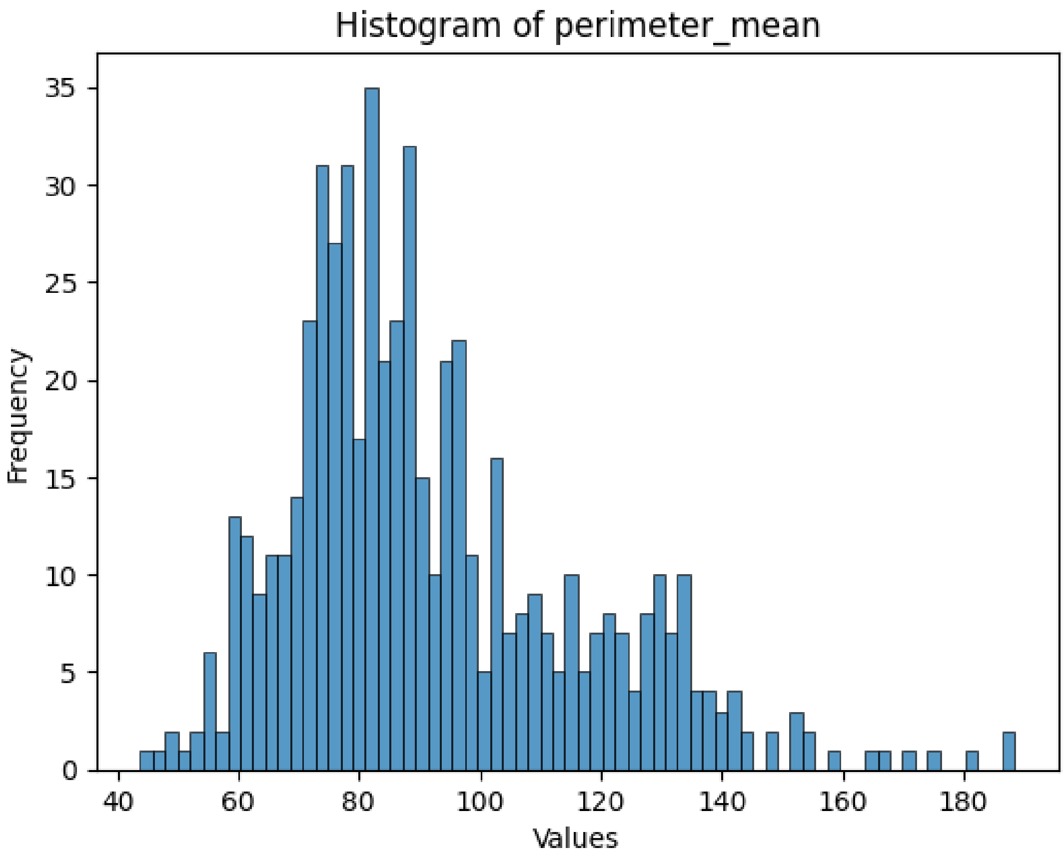
3.) Preprocessing

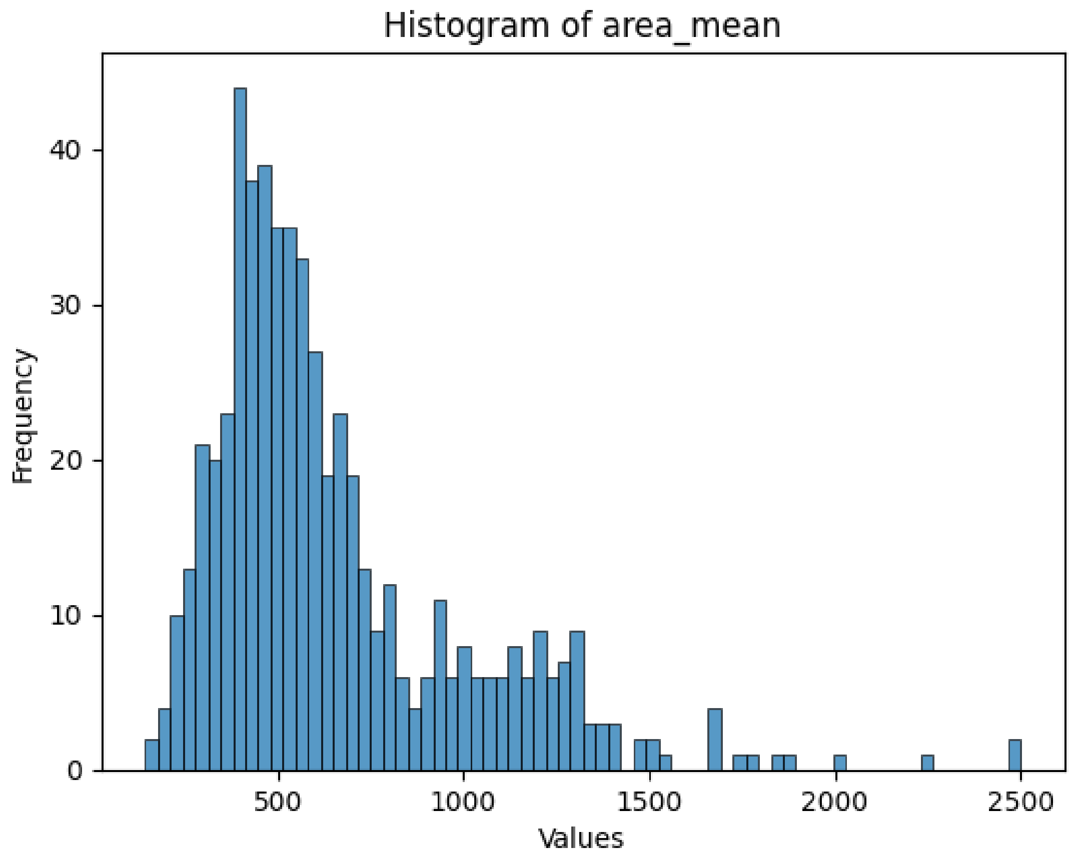
```
for column in df.columns:
    sns.histplot(df[column], bins=70)  # Adjust the number of bins as needed
    plt.title(f'Histogram of {column}')
    plt.xlabel('Values')
    plt.ylabel('Frequency')
    plt.show()
```

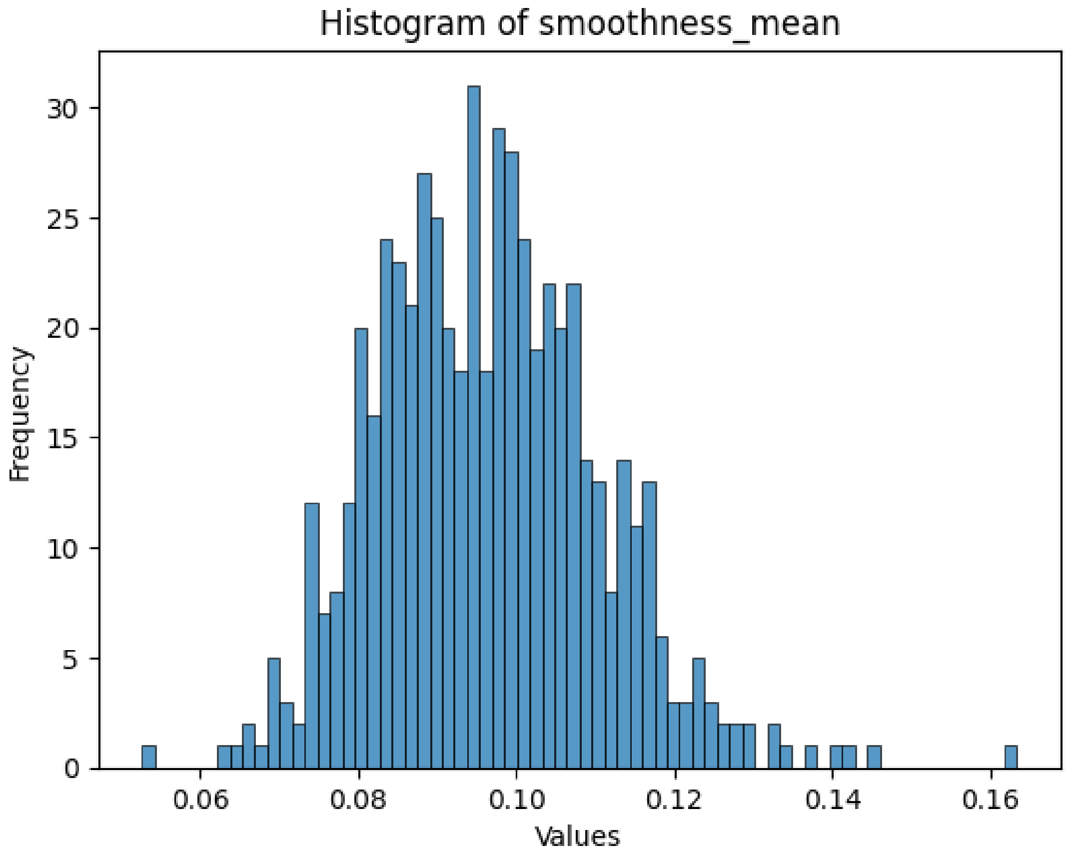

Histogram of diagnosis
Frequency
0
50
100
150
200
250
300
350
M
B
Values

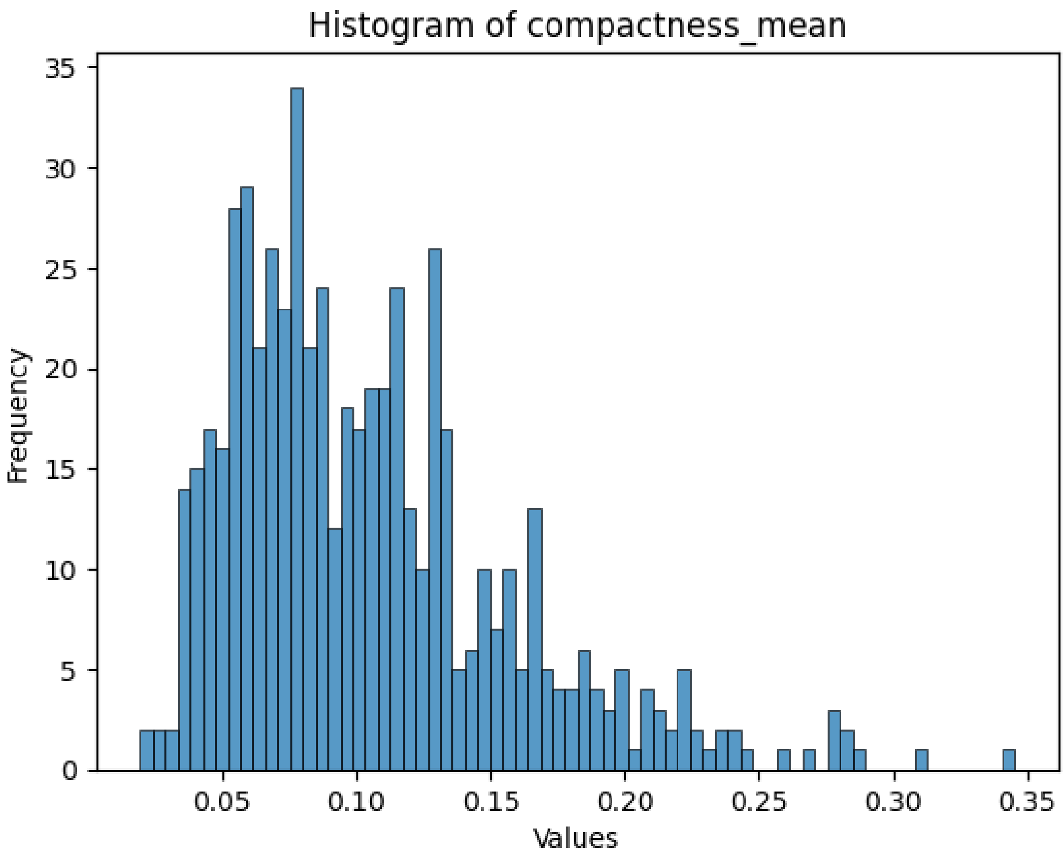

Histogram of radius_mean
Frequency
Values
0
5
10
15
20
25
30
35
10
15
20
25


Histogram of texture_mean
Frequency
Values
0
5
10
15
20
25
10
15
20
25
30
35
40

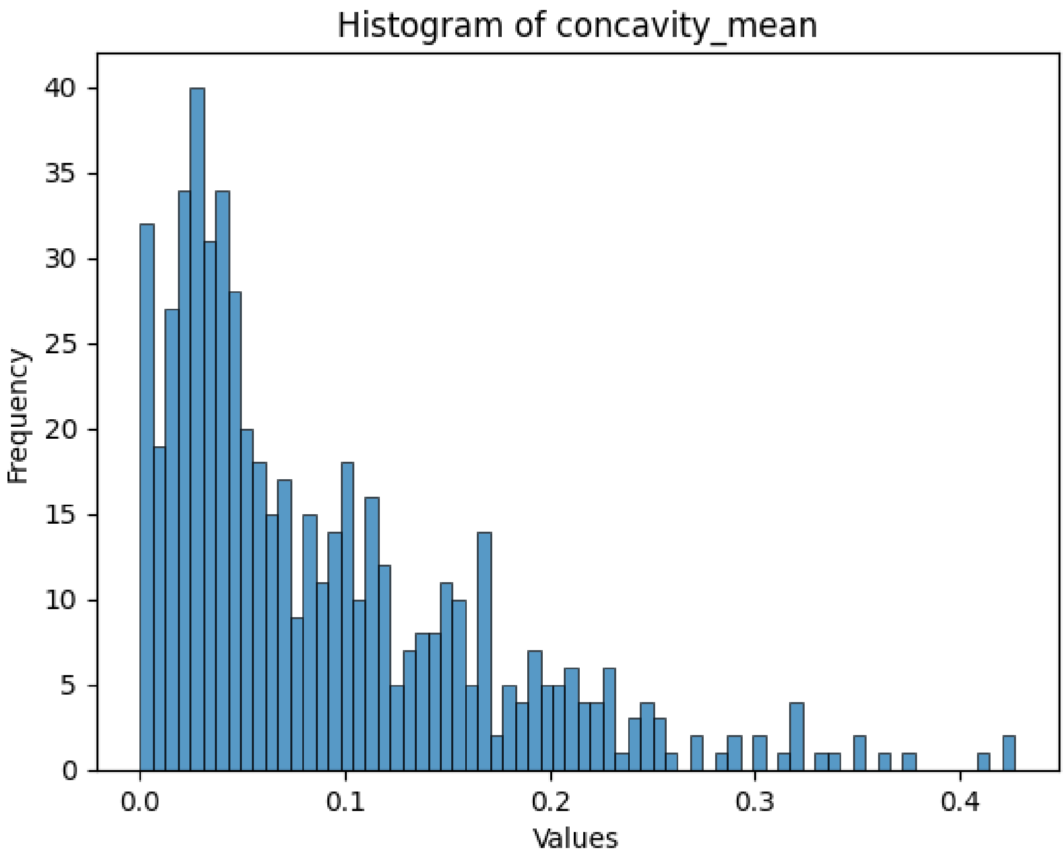

Histogram of perimeter_mean
Frequency
Values
0
5
10
15
20
25
30
35
40
60
80
100
120
140
160
180

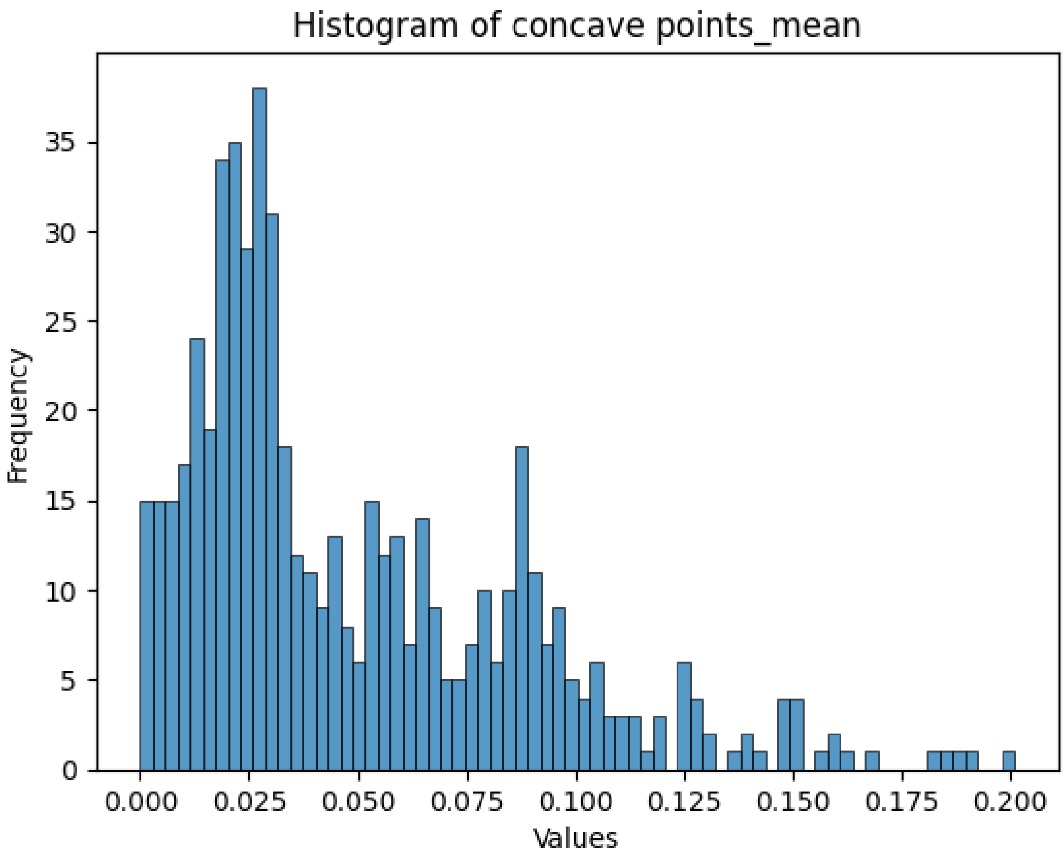

Histogram of area_mean
Frequency
Values
0
10
20
30
40
500
1000
1500
2000
2500


Histogram of smoothness_mean
Frequency
Values
0
5
10
15
20
25
30
0.06
0.08
0.10
0.12
0.14
0.16

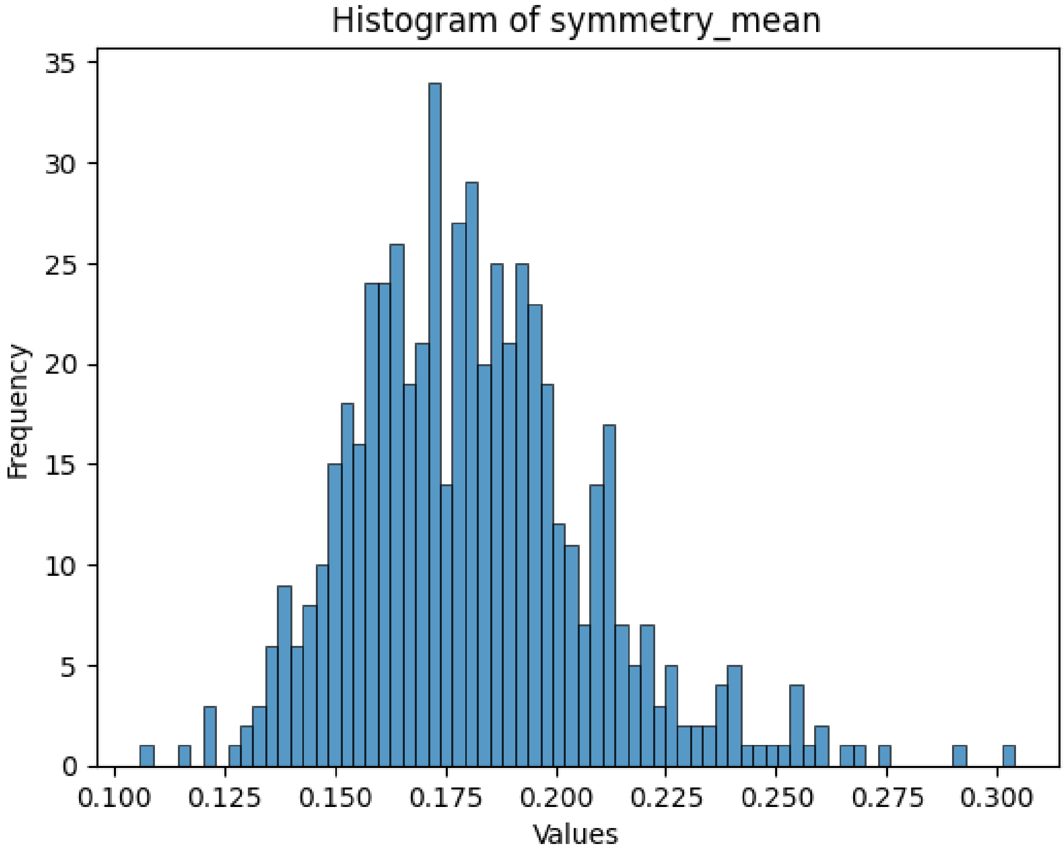


Histogram of compactness_mean
Frequency
Values
0
5
10
15
20
25
30
35
0.05
0.10
0.15
0.20
0.25
0.30
0.35

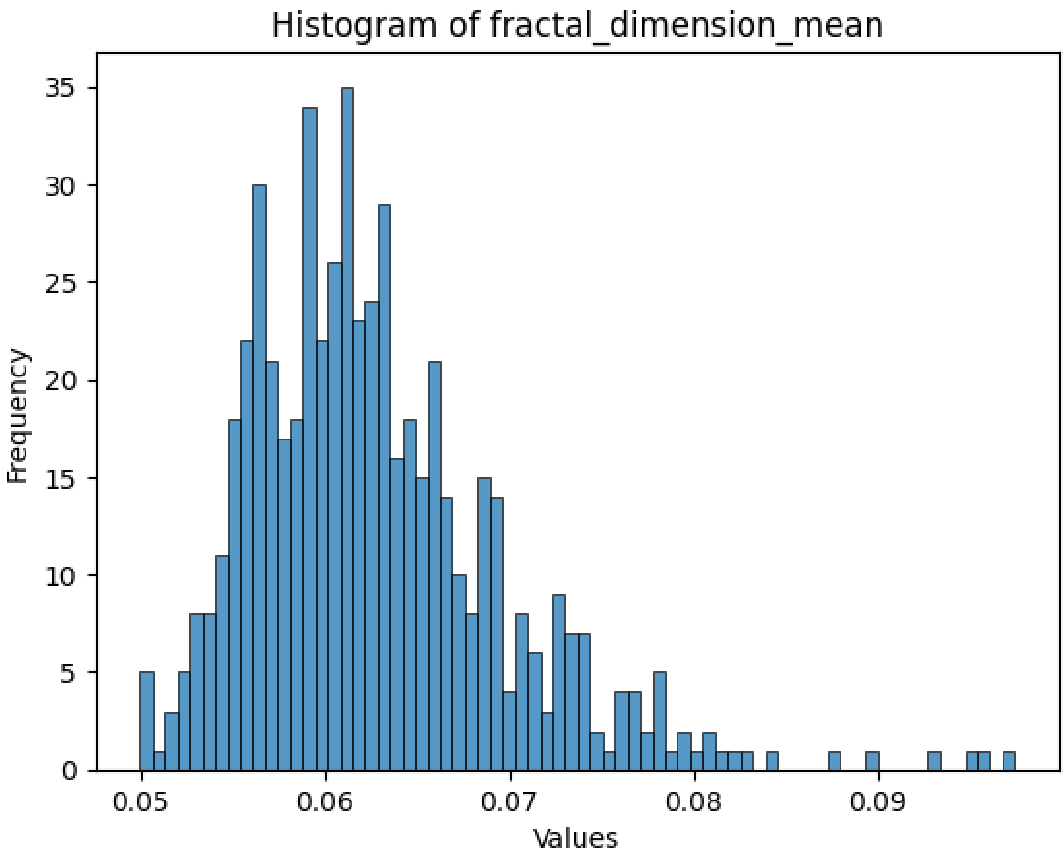

Histogram of concavity_mean
Frequency
Values
0
5
10
15
20
25
30
35
40
0.0
0.1
0.2
0.3
0.4

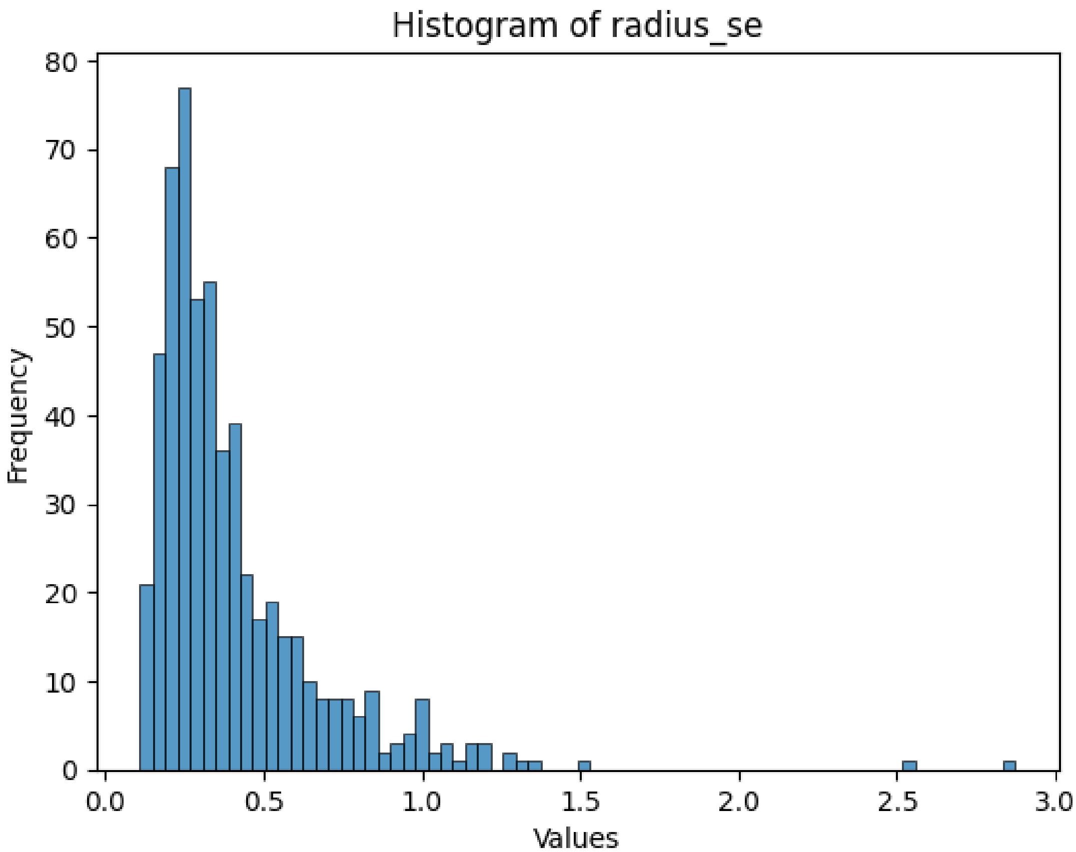


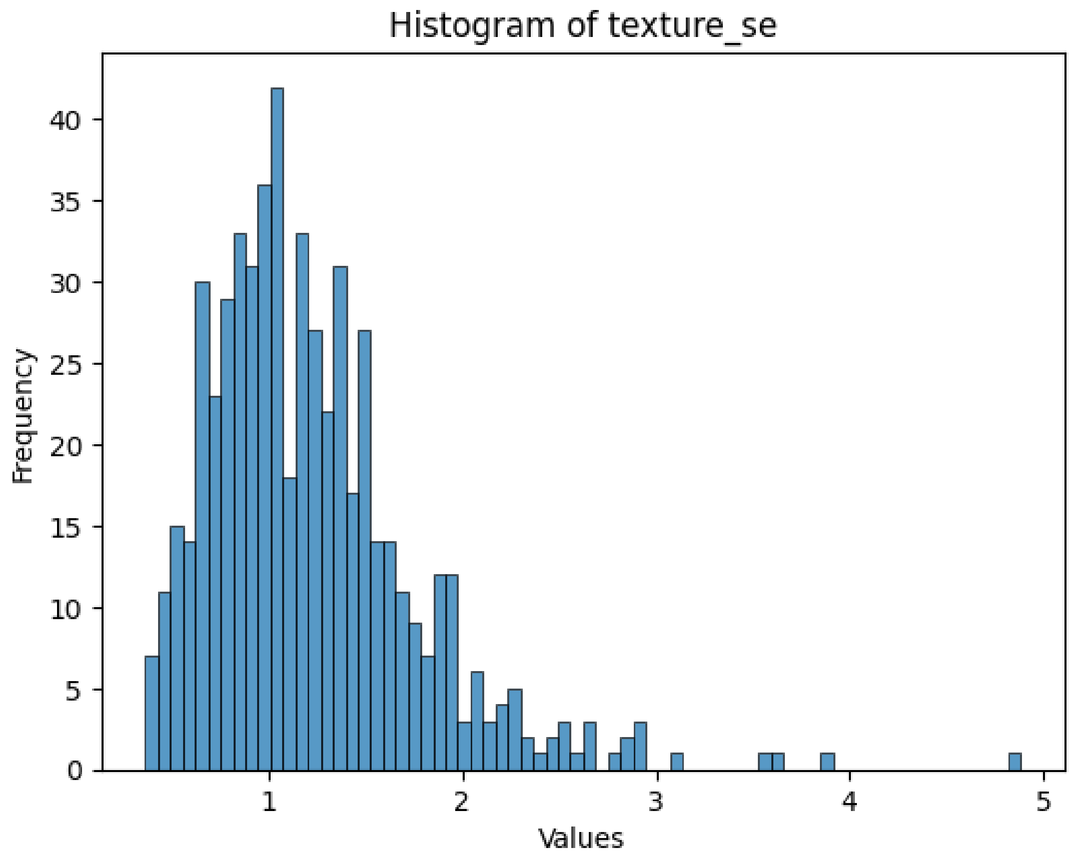
Histogram of concave points_mean
Frequency
Values
0
5
10
15
20
25
30
35
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

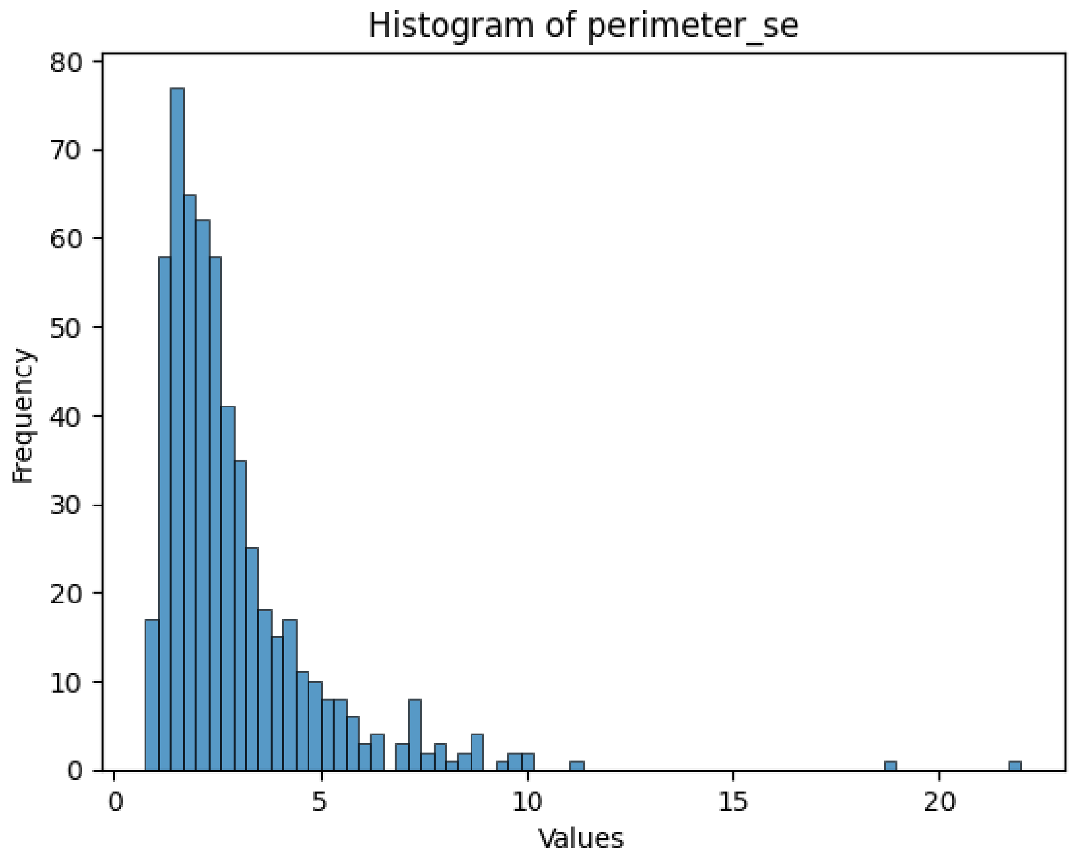


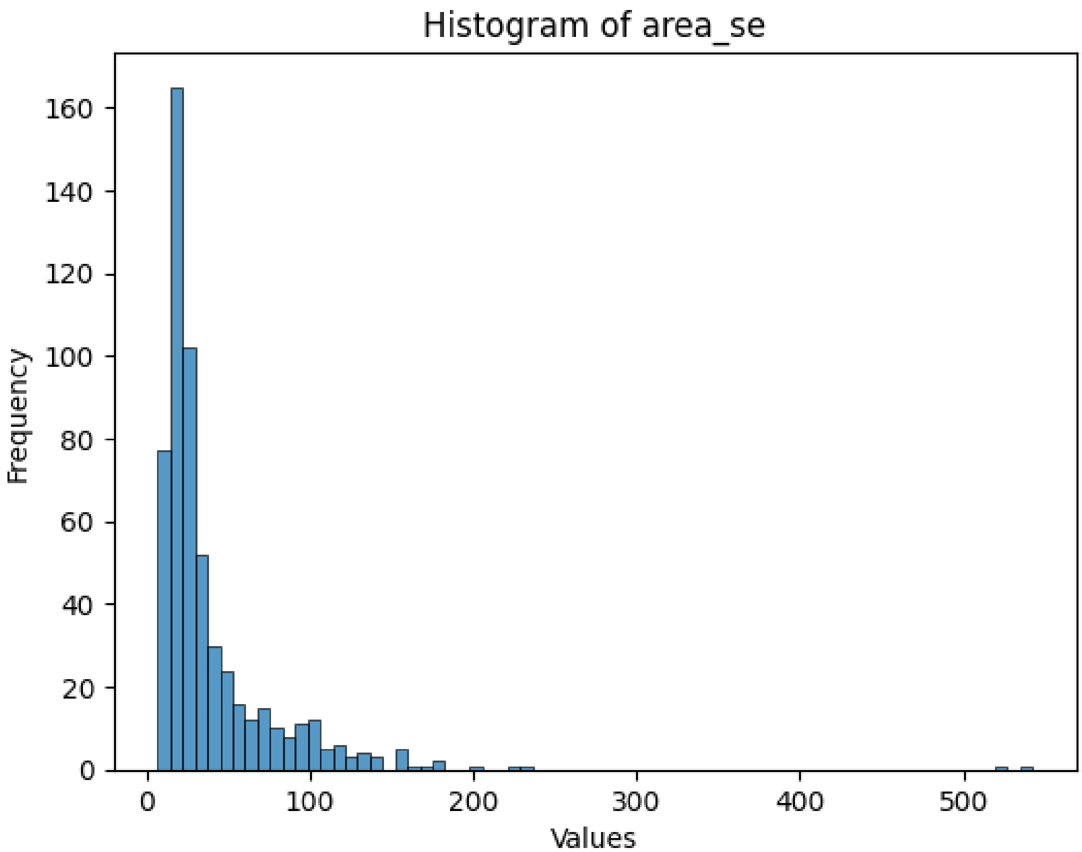
Histogram of symmetry_mean
Frequency
Values
0
5
10
15
20
25
30
35
0.100
0.125
0.150
0.175
0.200
0.225
0.250
0.275
0.300

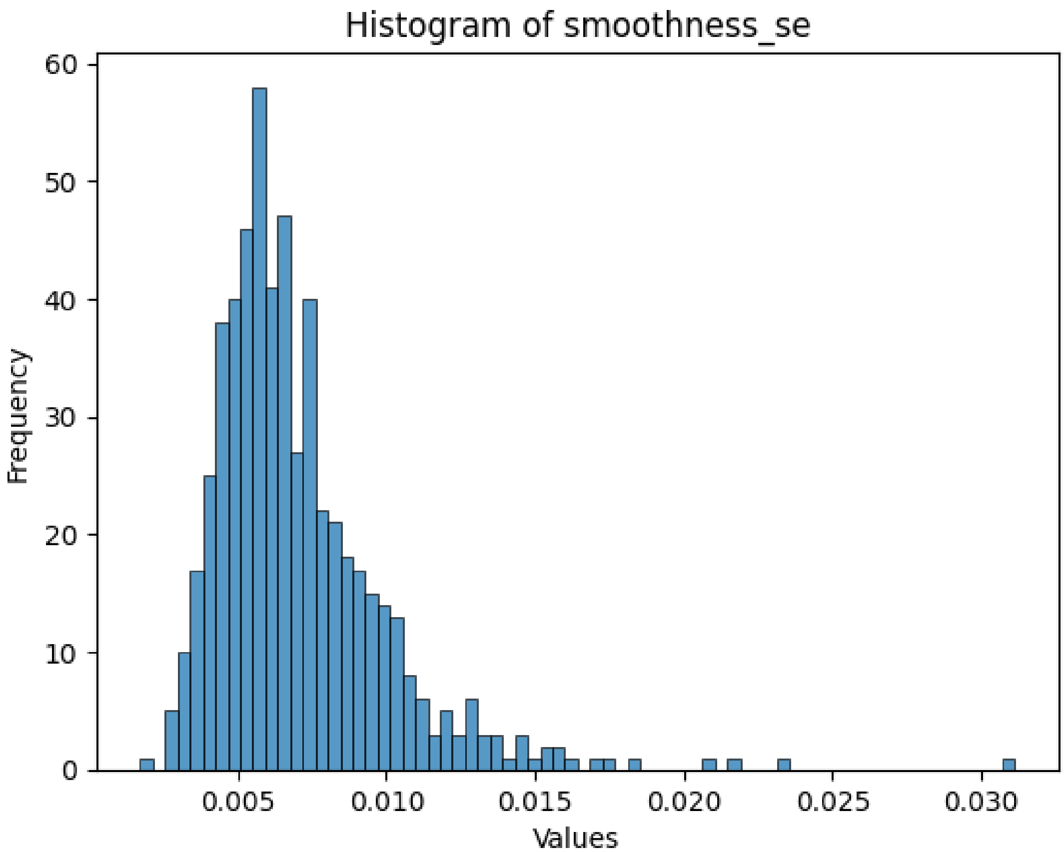

Histogram of fractal_dimension_mean
Frequency
Values
0
5
10
15
20
25
30
35
0.05
0.06
0.07
0.08
0.09


Histogram of radius_se
Frequency
Values
0
10
20
30
40
50
60
70
80
0.0
0.5
1.0
1.5
2.0
2.5
3.0

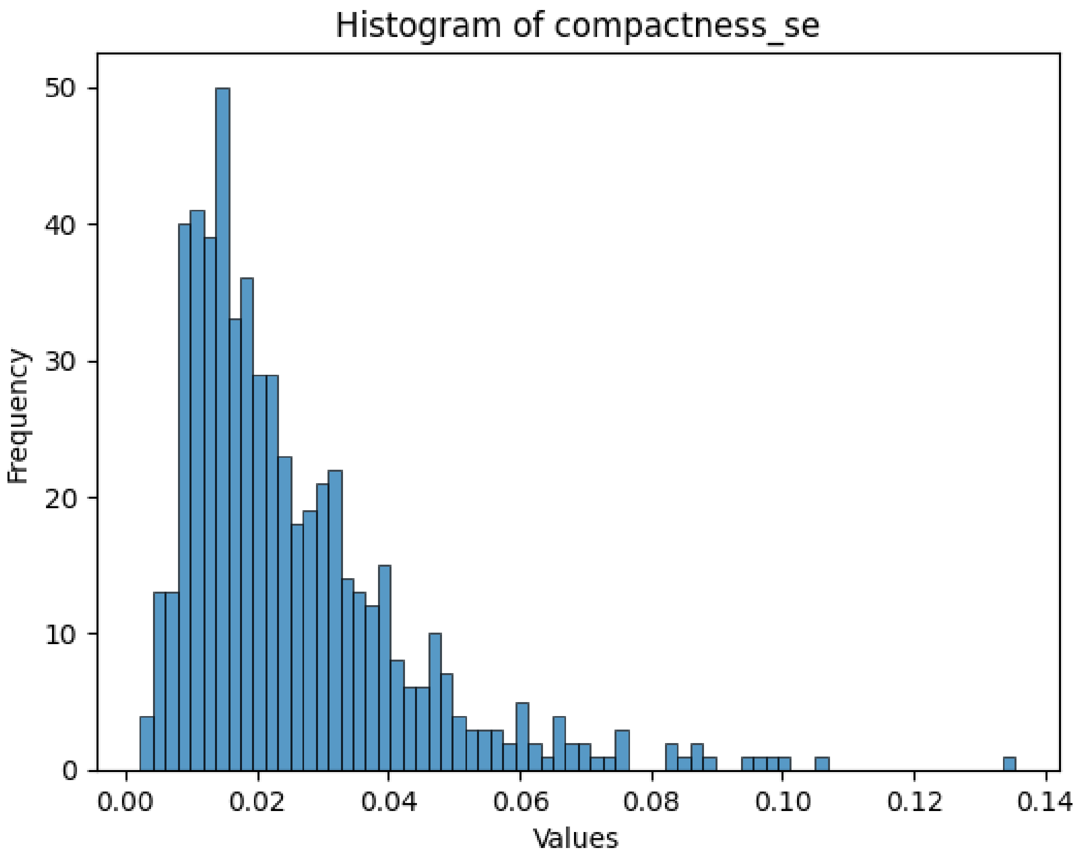

Histogram of texture_se
Frequency
0
5
10
15
20
25
30
35
40
1
2
3
4
5
Values

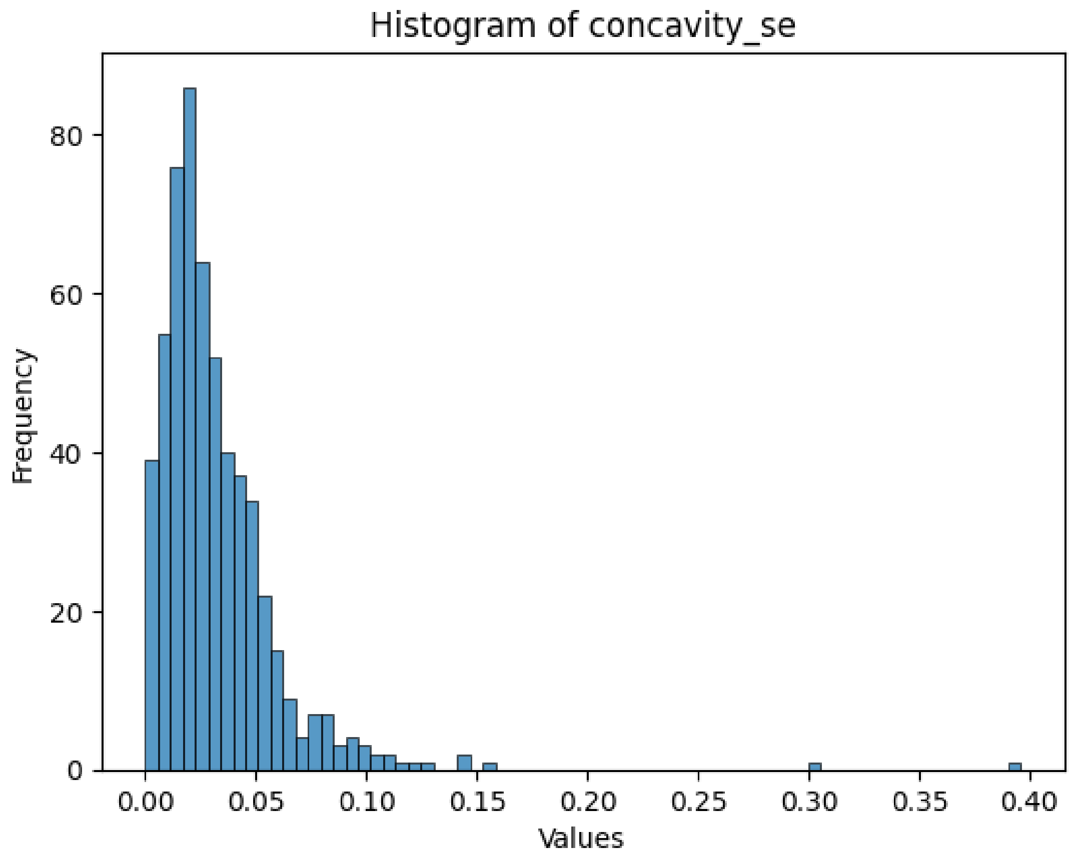

Histogram of perimeter_se
Frequency
Values
0
10
20
30
40
50
60
70
80
0
5
10
15
20

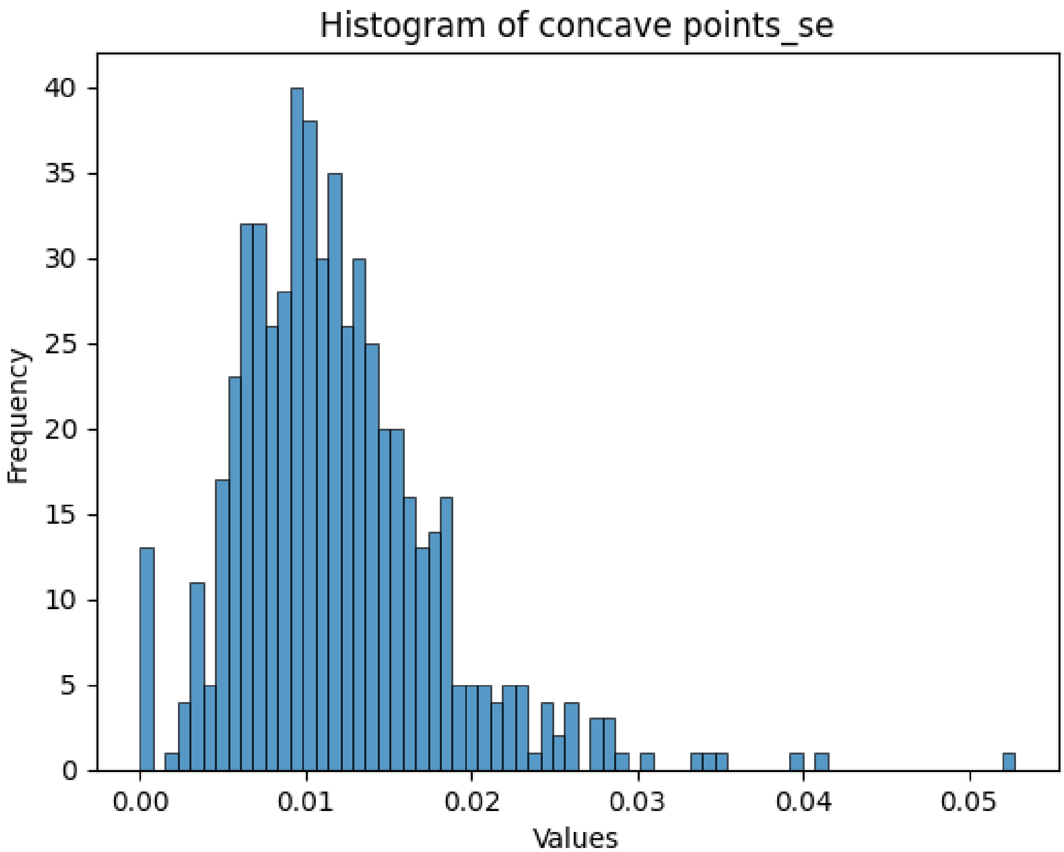

Histogram of area_se
Frequency
Values
0
20
40
60
80
100
120
140
160
0
100
200
300
400
500

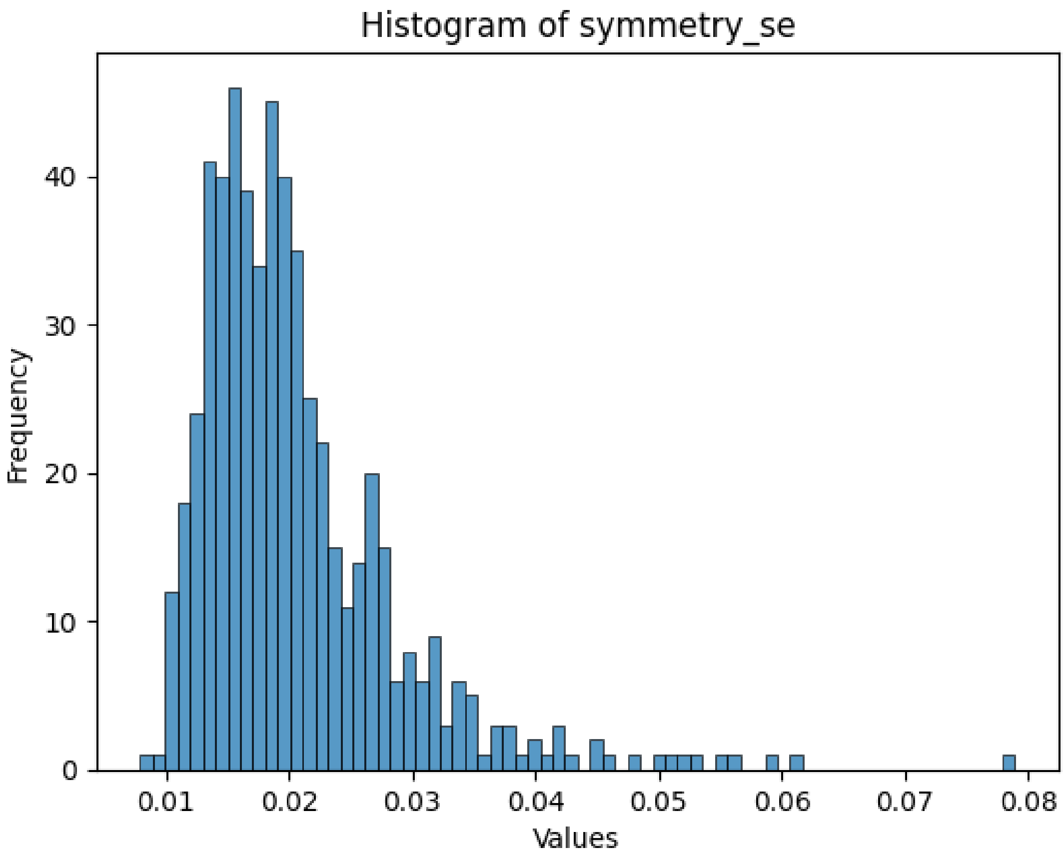


Histogram of smoothness_se
Frequency
Values
0
10
20
30
40
50
60
0.005
0.010
0.015
0.020
0.025
0.030

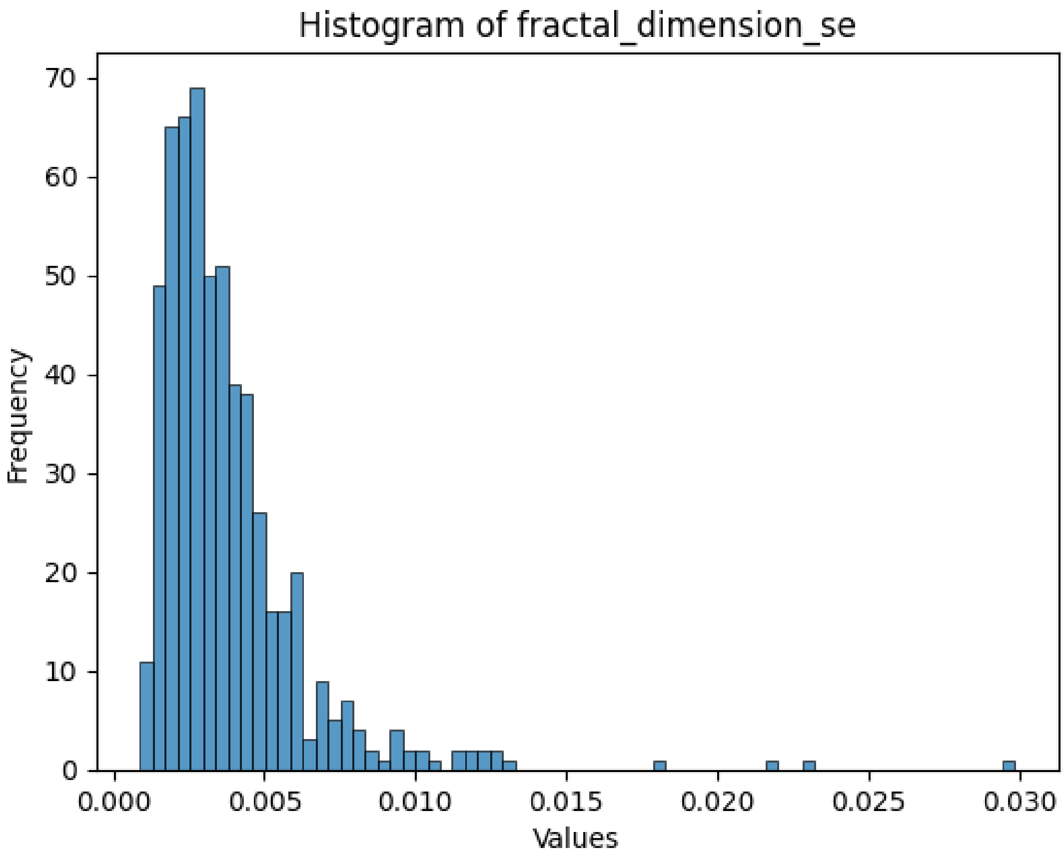

Histogram of compactness_se
Frequency
Values
0
10
20
30
40
50
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14


Histogram of concavity_se
Frequency
80
60
40
20
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Values

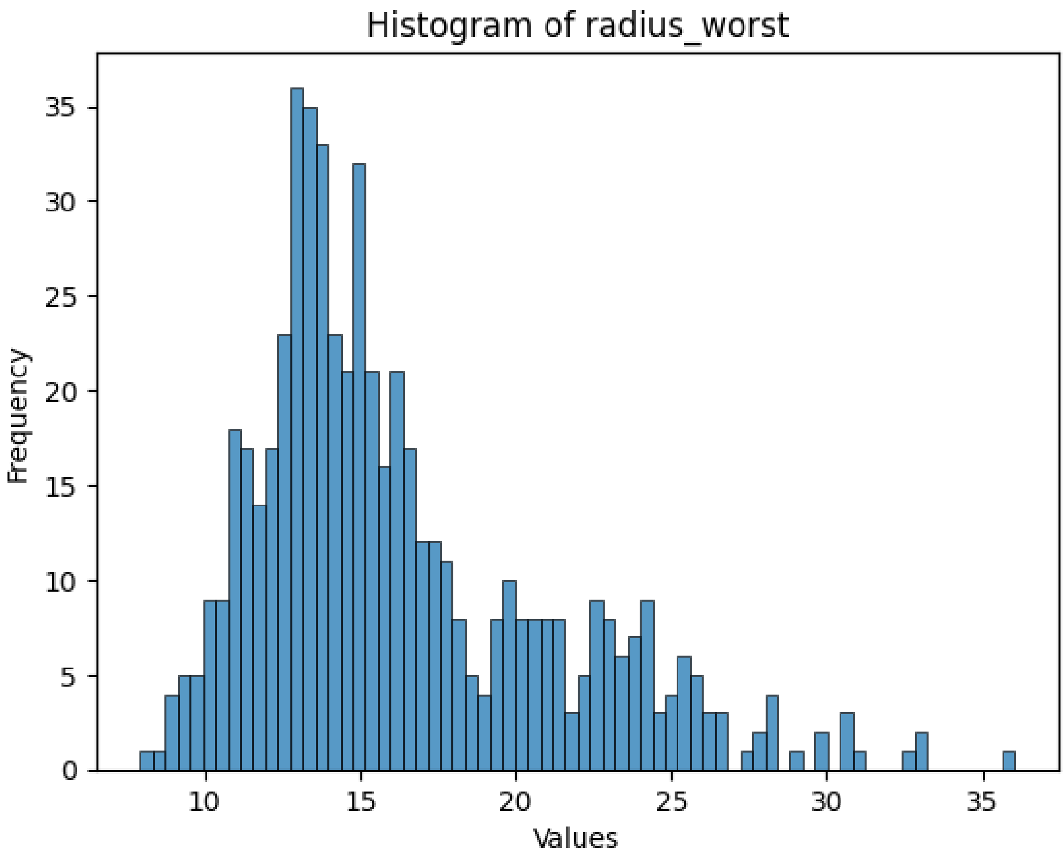

Histogram of concave points_se
Frequency
Values
0
5
10
15
20
25
30
35
40
0.00
0.01
0.02
0.03
0.04
0.05



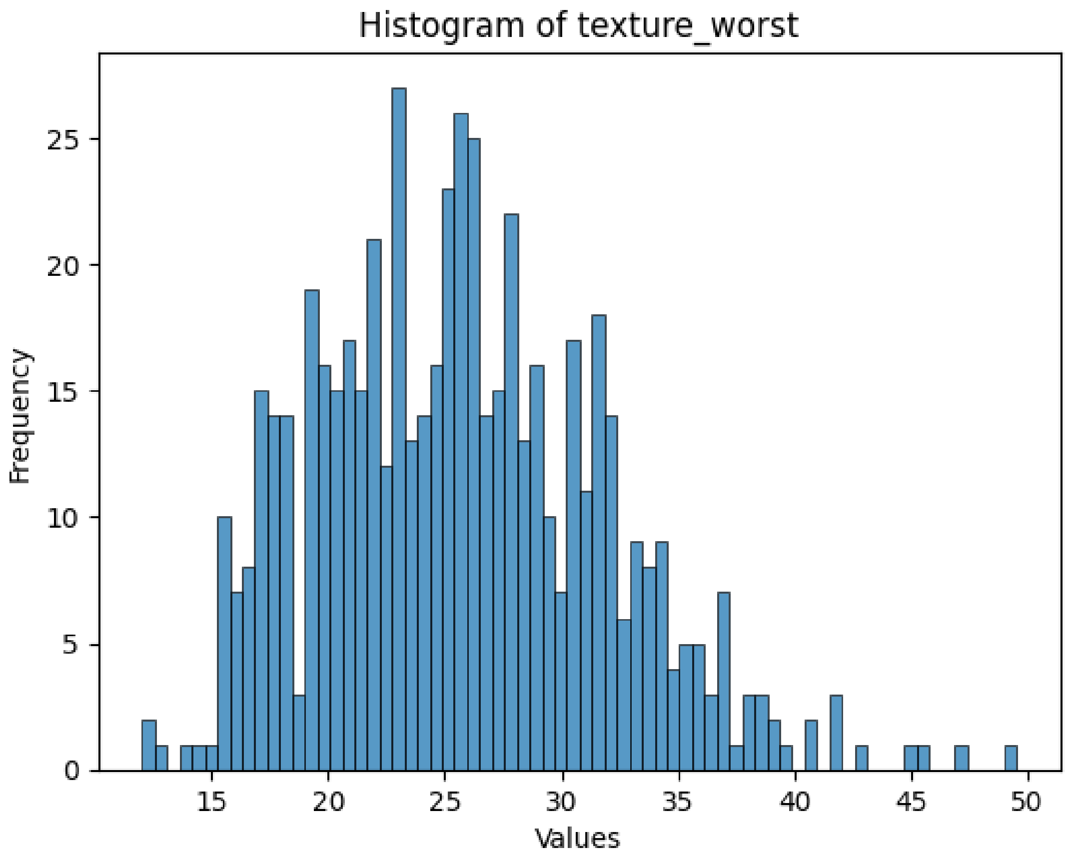
Histogram of symmetry_se
Frequency
Values
0
10
20
30
40
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08



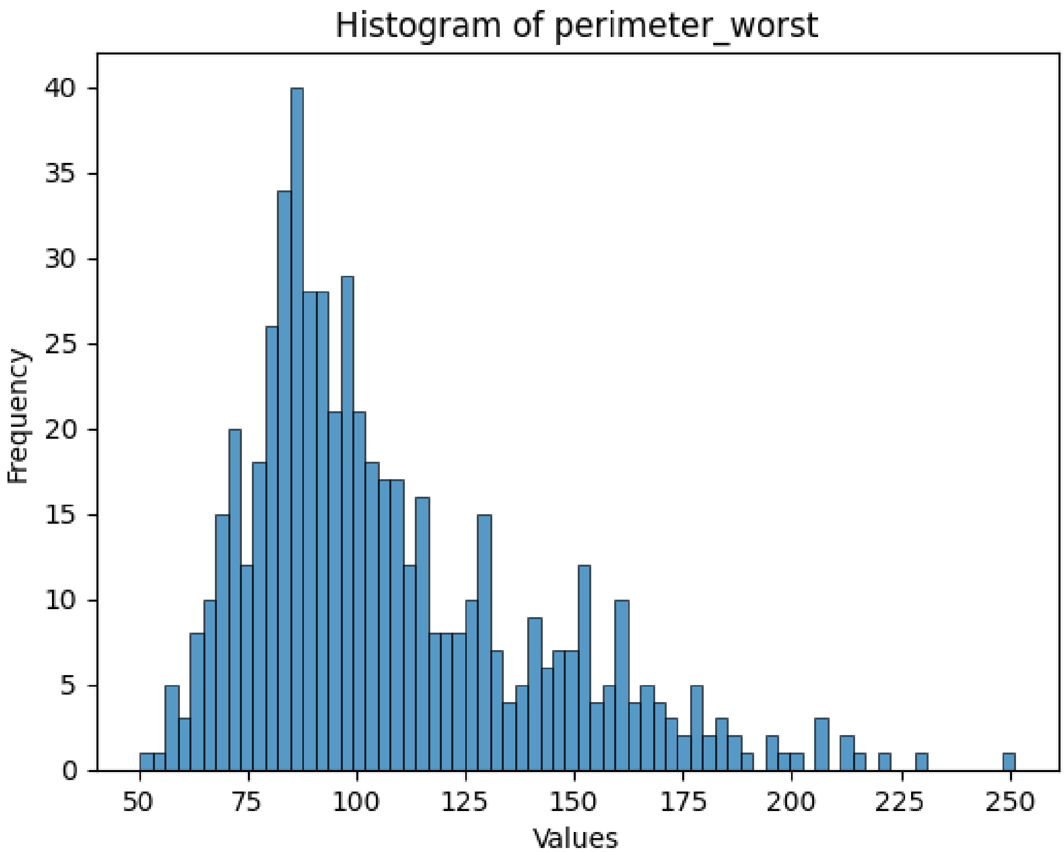
Histogram of fractal_dimension_se
Frequency
Values
0
10
20
30
40
50
60
70
0.000
0.005
0.010
0.015
0.020
0.025
0.030

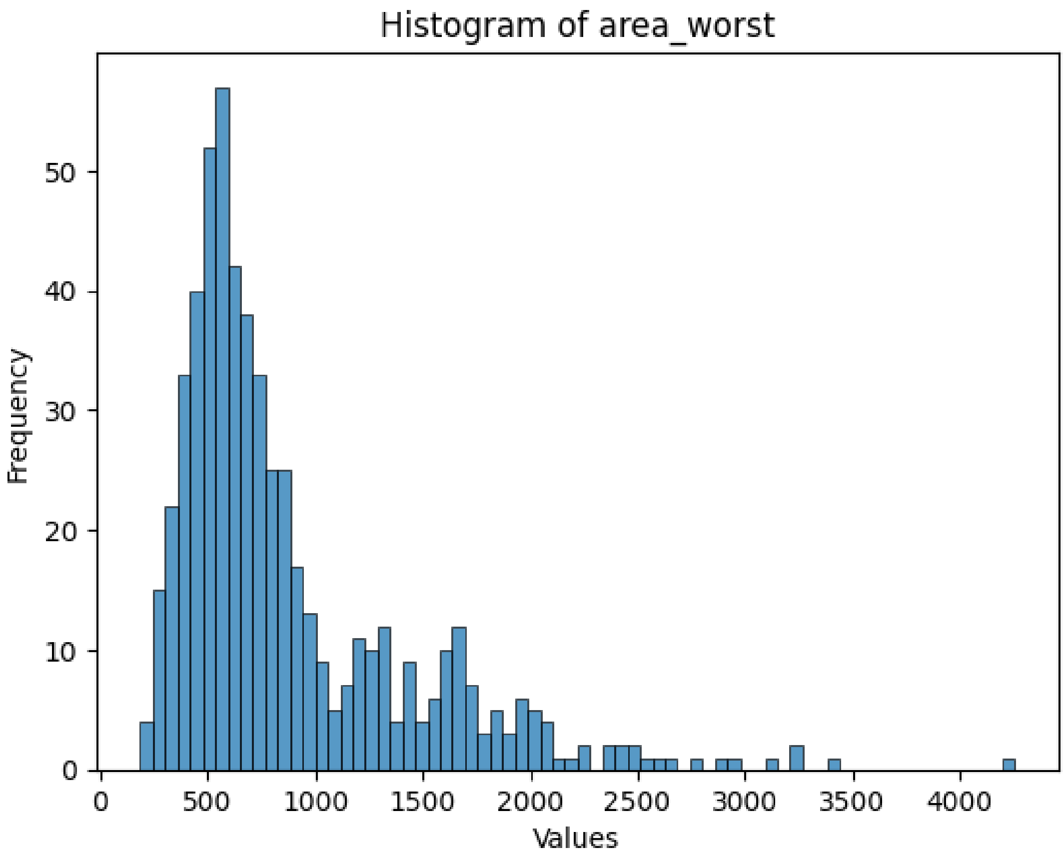


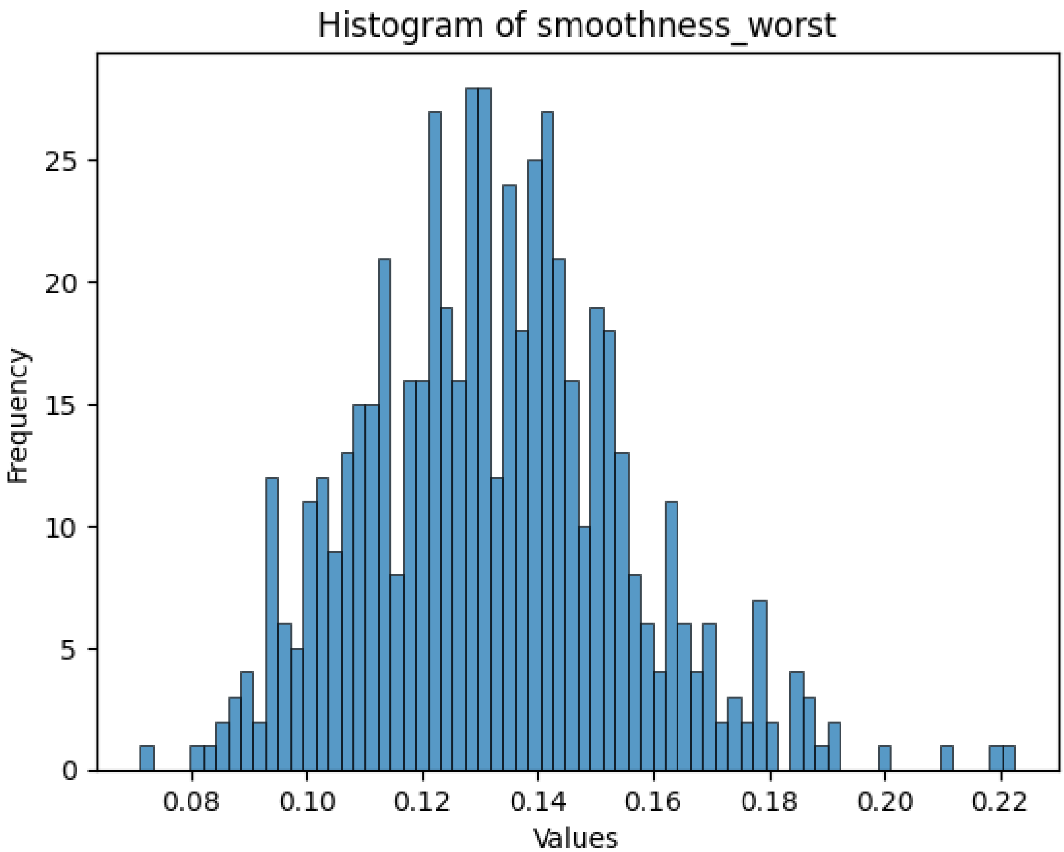
Histogram of radius_worst
Frequency
Values
0
5
10
15
20
25
30
35
10
15
20
25
30
35

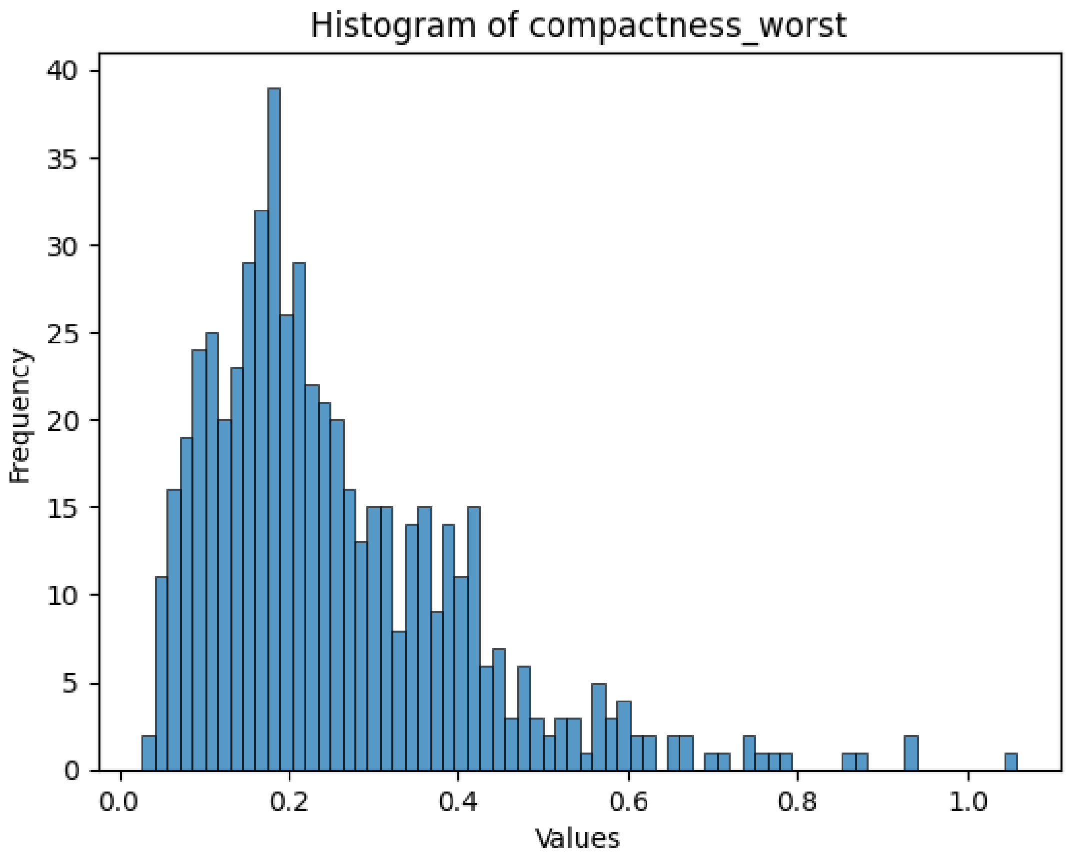


Histogram of texture_worst
Frequency
Values
0
5
10
15
20
25
15
20
25
30
35
40
45
50

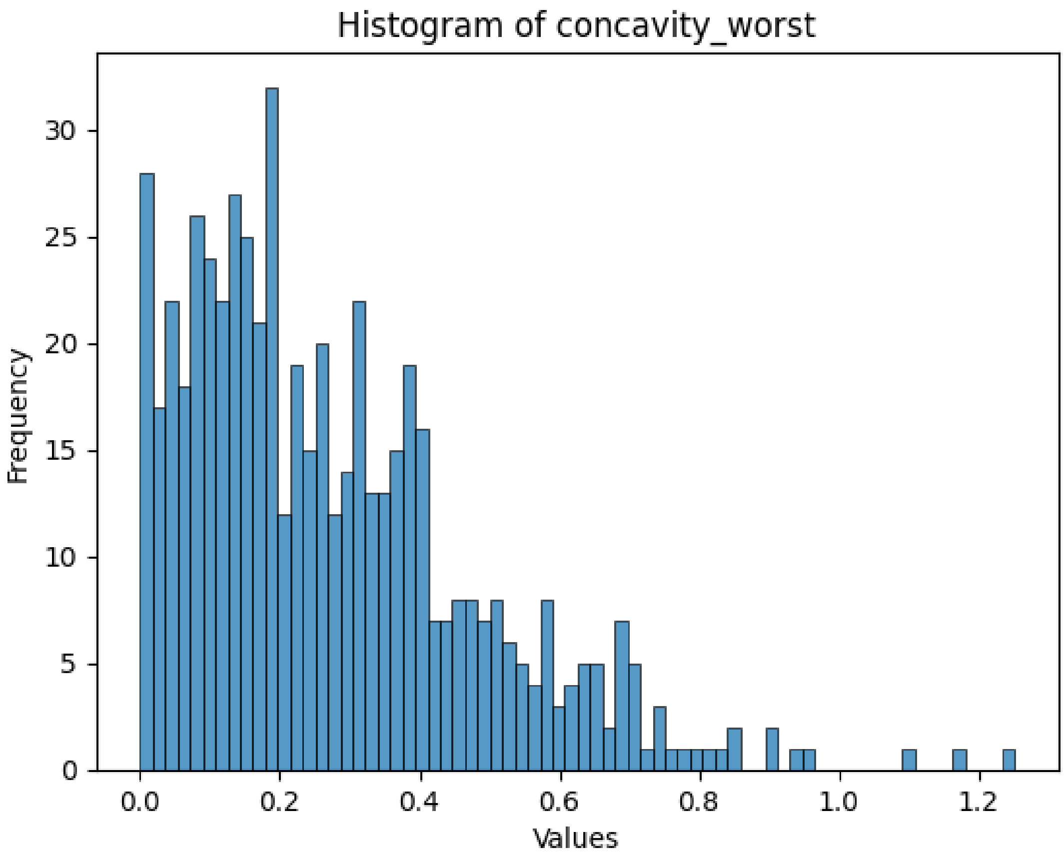

Histogram of perimeter_worst
Frequency
Values
0
5
10
15
20
25
30
35
40
50
75
100
125
150
175
200
225
250

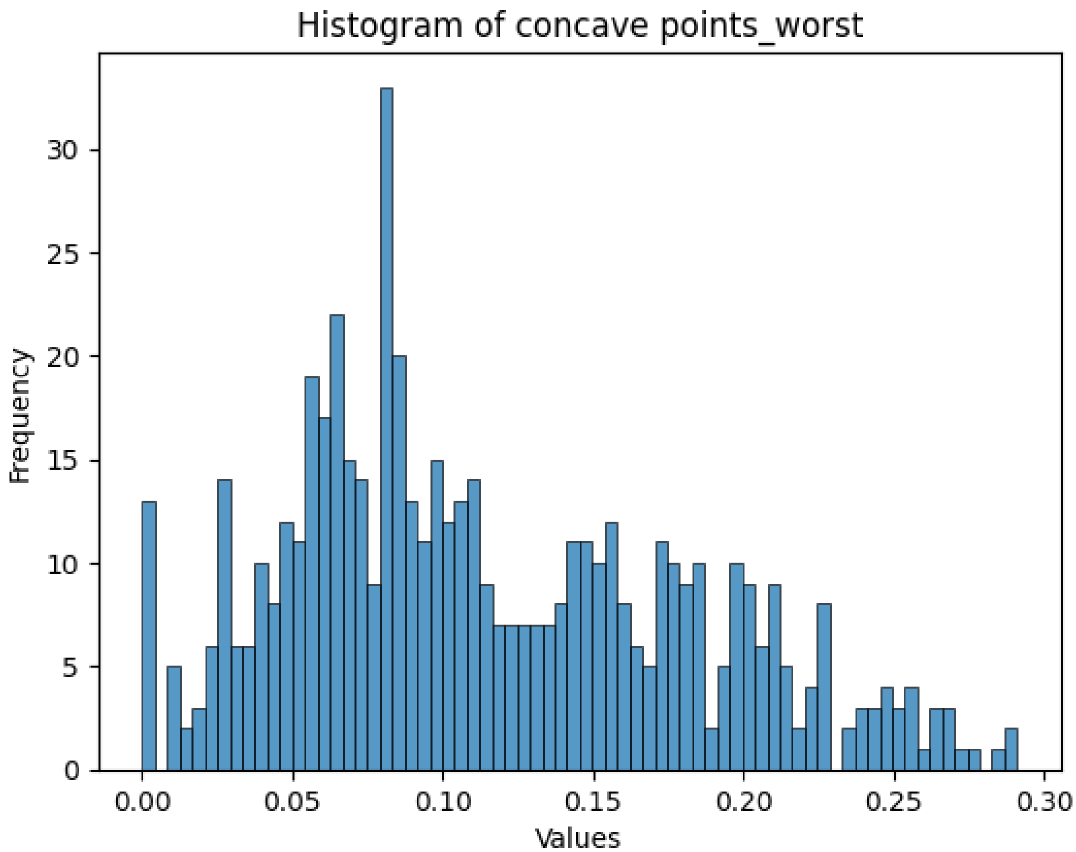

Histogram of area_worst
Frequency
Values
0
10
20
30
40
50
0
500
1000
1500
2000
2500
3000
3500
4000

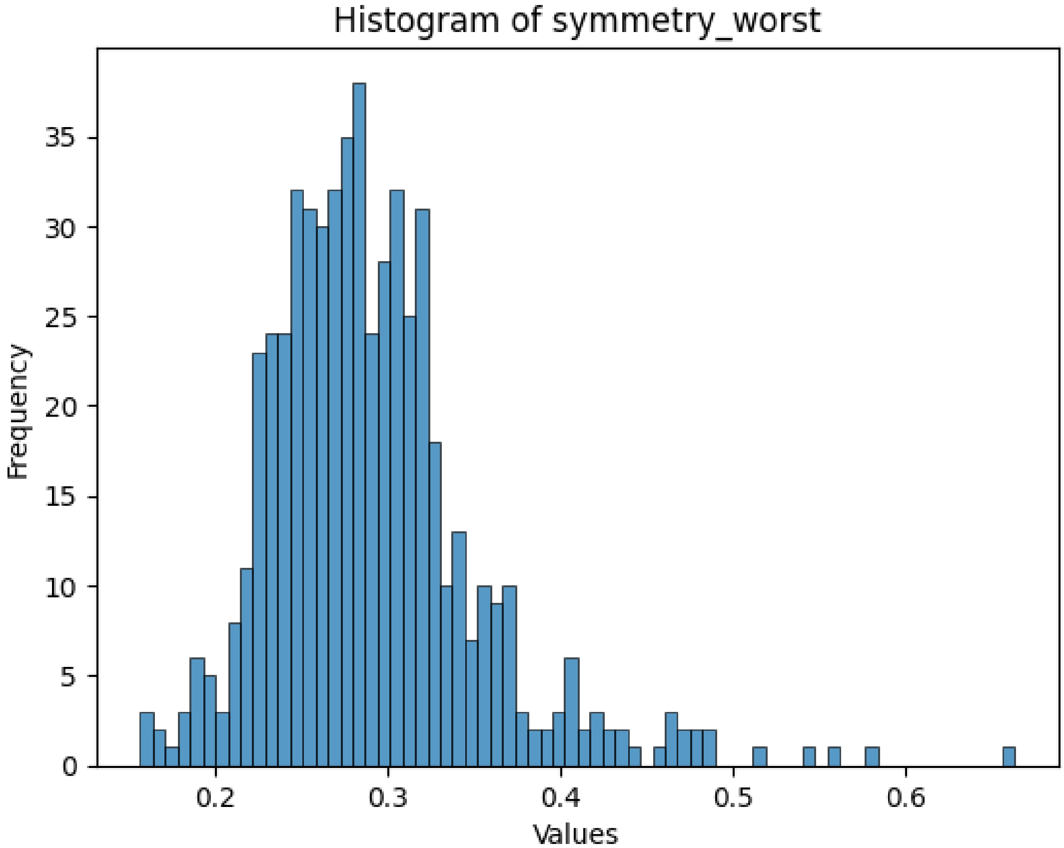

Histogram of smoothness_worst
Frequency
0
5
10
15
20
25
0.08
0.10
0.12
0.14
0.16
0.18
0.20
0.22
Values

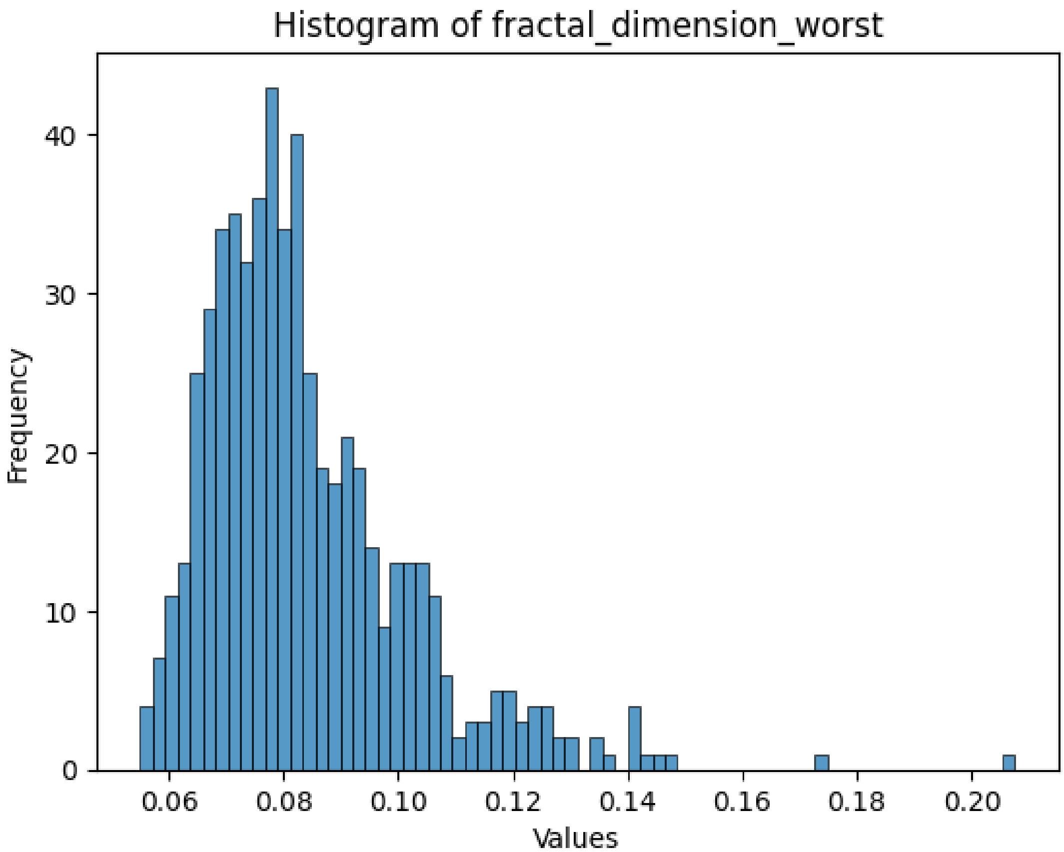


Histogram of compactness_worst
Frequency
Values
0
5
10
15
20
25
30
35
40
0.0
0.2
0.4
0.6
0.8
1.0

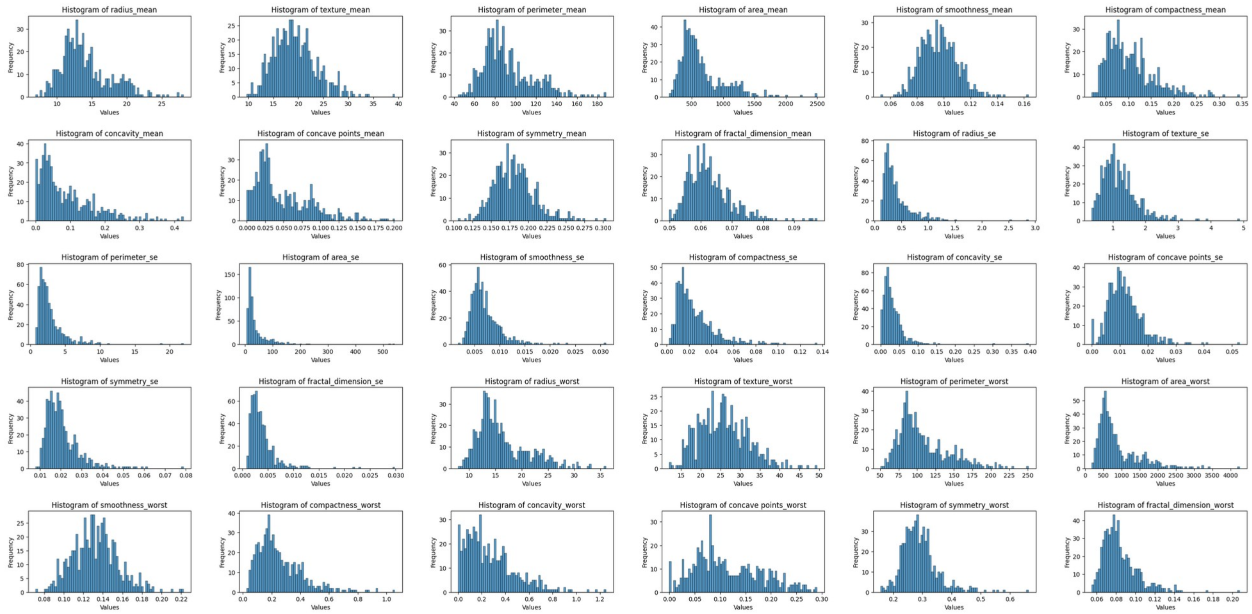

Histogram of concavity_worst
Frequency
Values
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Histogram of concave points_worst
Frequency
Values
0
5
10
15
20
25
30
0.00
0.05
0.10
0.15
0.20
0.25
0.30


Histogram of symmetry_worst
Frequency
Values
0
5
10
15
20
25
30
35
0.2
0.3
0.4
0.5
0.6




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

# Showing the Histograms as a Grid Format
n_rows = 5
n_cols = 6
fig, axes = plt.subplots(n_rows, n_cols, figsize=(30, 15))
fig.tight_layout(pad=5.0)
axes = axes.flatten()
for i, column in enumerate(X.columns):
    if i < len(axes):
        sns.histplot(X[column], bins=70, ax=axes[i])
        axes[i].set_title(f'Histogram of {column}')
        axes[i].set_xlabel('Values')
        axes[i].set_ylabel('Frequency')
    else:
        break
plt.show()
```



3.1. Normalization

```
scaler = MinMaxScaler()
# Fit and transform the data to normalize each feature between 0 and 1
df_scaled = scaler.fit_transform(X)
column_titles = X.columns.tolist()
data = pd.DataFrame(df_scaled, columns=column_titles)
data.describe()
```

```
       radius_mean  texture_mean  perimeter_mean   area_mean
smoothness_mean  \
count   569.000000    569.000000      569.000000  569.000000
569.000000
mean      0.338222      0.323965        0.332935    0.216920
0.394785
std       0.166787      0.145453        0.167915    0.149274
0.126967
min       0.000000      0.000000        0.000000    0.000000
0.000000
25%       0.223342      0.218465        0.216847    0.117413
0.304595
50%       0.302381      0.308759        0.293345    0.172895
0.390358
75%       0.416442      0.408860        0.416765    0.271135
0.475490
max       1.000000      1.000000        1.000000    1.000000
1.000000

       compactness_mean  concavity_mean  concave points_mean
symmetry_mean  \
count        569.000000      569.000000           569.000000
```

```
569.000000
mean              0.260601        0.208058             0.243137
0.379605
std               0.161992        0.186785             0.192857
0.138456
min               0.000000        0.000000             0.000000
0.000000
25%               0.139685        0.069260             0.100944
0.282323
50%               0.224679        0.144189             0.166501
0.369697
75%               0.340531        0.306232             0.367793
0.453030
max               1.000000        1.000000             1.000000
1.000000

       fractal_dimension_mean  ...  radius_worst  texture_worst  \
count              569.000000  ...    569.000000     569.000000
mean                 0.270379  ...      0.296663       0.363998
std                  0.148702  ...      0.171940       0.163813
min                  0.000000  ...      0.000000       0.000000
25%                  0.163016  ...      0.180719       0.241471
50%                  0.243892  ...      0.250445       0.356876
75%                  0.340354  ...      0.386339       0.471748
max                  1.000000  ...      1.000000       1.000000

       perimeter_worst  area_worst  smoothness_worst
compactness_worst  \
count       569.000000  569.000000        569.000000
569.000000
mean          0.283138    0.170906          0.404138
0.220212
std           0.167352    0.139932          0.150779
0.152649
min           0.000000    0.000000          0.000000
0.000000
25%           0.167837    0.081130          0.300007
0.116337
50%           0.235320    0.123206          0.397081
0.179110
75%           0.373475    0.220901          0.494156
0.302520
max           1.000000    1.000000          1.000000
1.000000

       concavity_worst  concave points_worst  symmetry_worst  \
count       569.000000            569.000000      569.000000
mean          0.217403              0.393836        0.263307
std           0.166633              0.225884        0.121954
min           0.000000              0.000000        0.000000
```

```
25%          0.091454               0.223127        0.185098
50%          0.181070               0.343402        0.247782
75%          0.305831               0.554639        0.318155
max          1.000000               1.000000        1.000000

       fractal_dimension_worst
count               569.000000
mean                  0.189596
std                   0.118466
min                   0.000000
25%                   0.107700
50%                   0.163977
75%                   0.242949
max                   1.000000

[8 rows x 30 columns]
```

```
data.info()
```

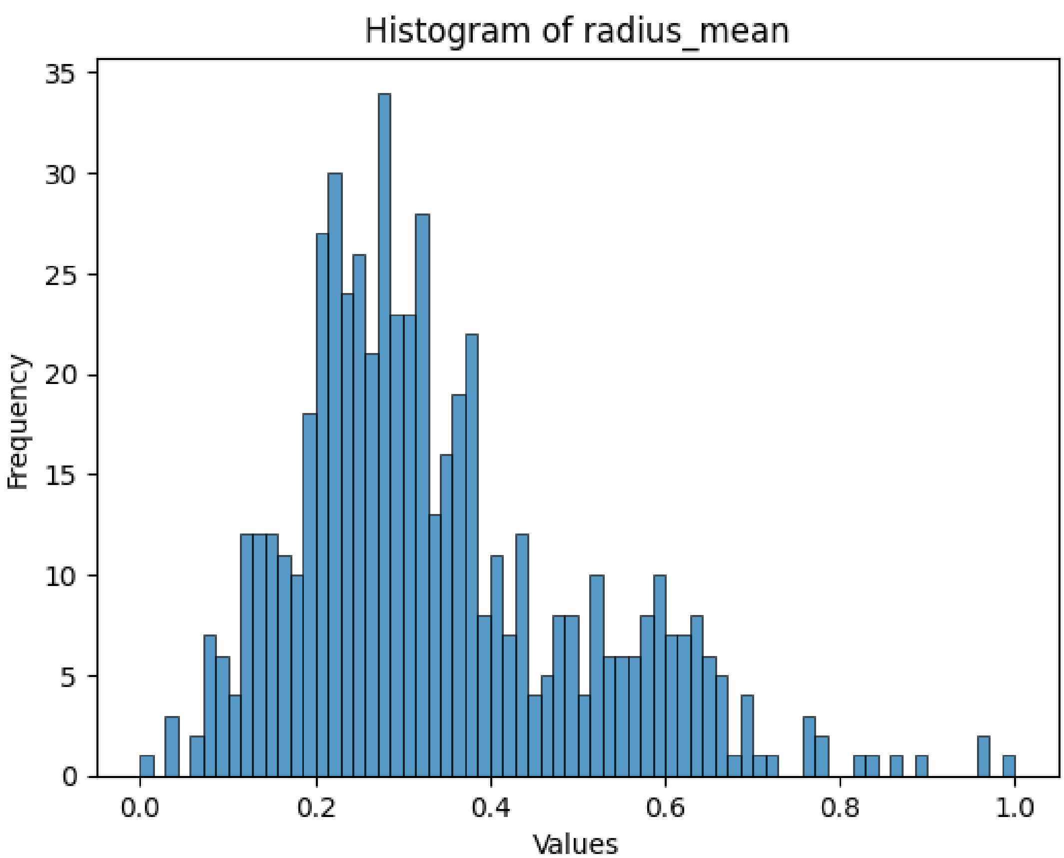
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   radius_mean              569 non-null    float64
 1   texture_mean             569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

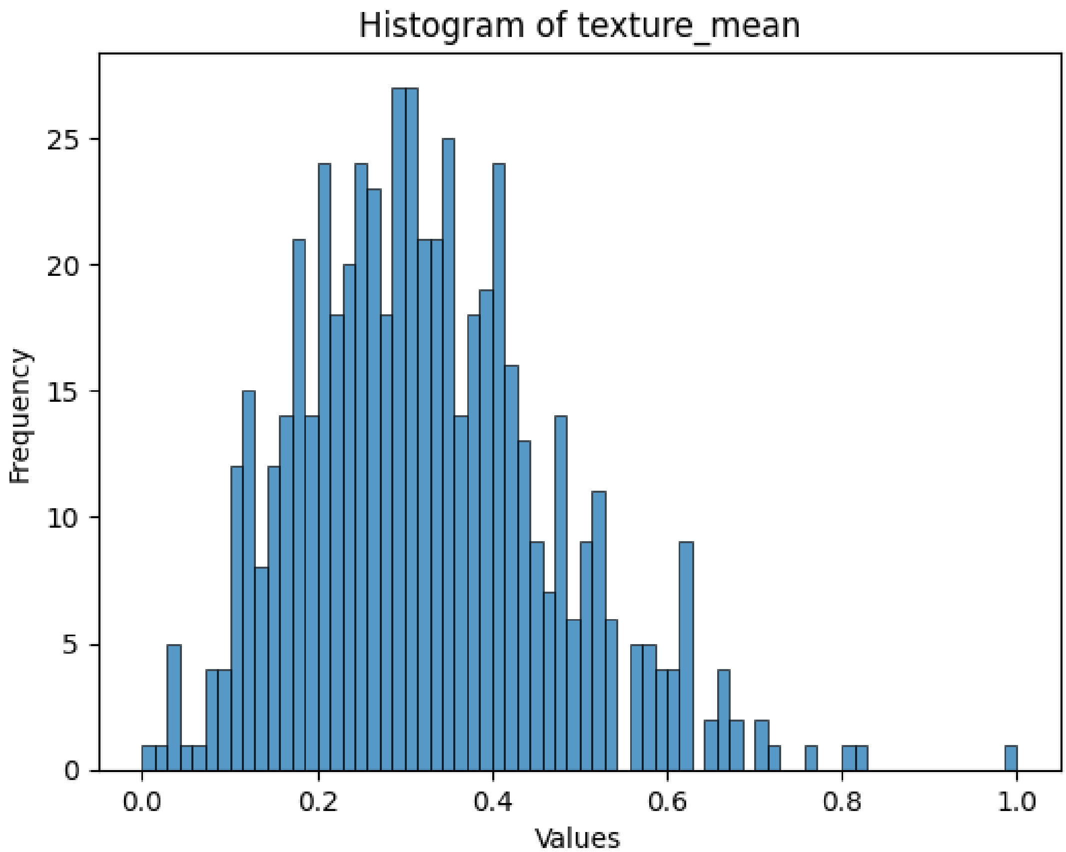
```
 26  concavity_worst          569 non-null    float64
 27  concave points_worst     569 non-null    float64
 28  symmetry_worst           569 non-null    float64
 29  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

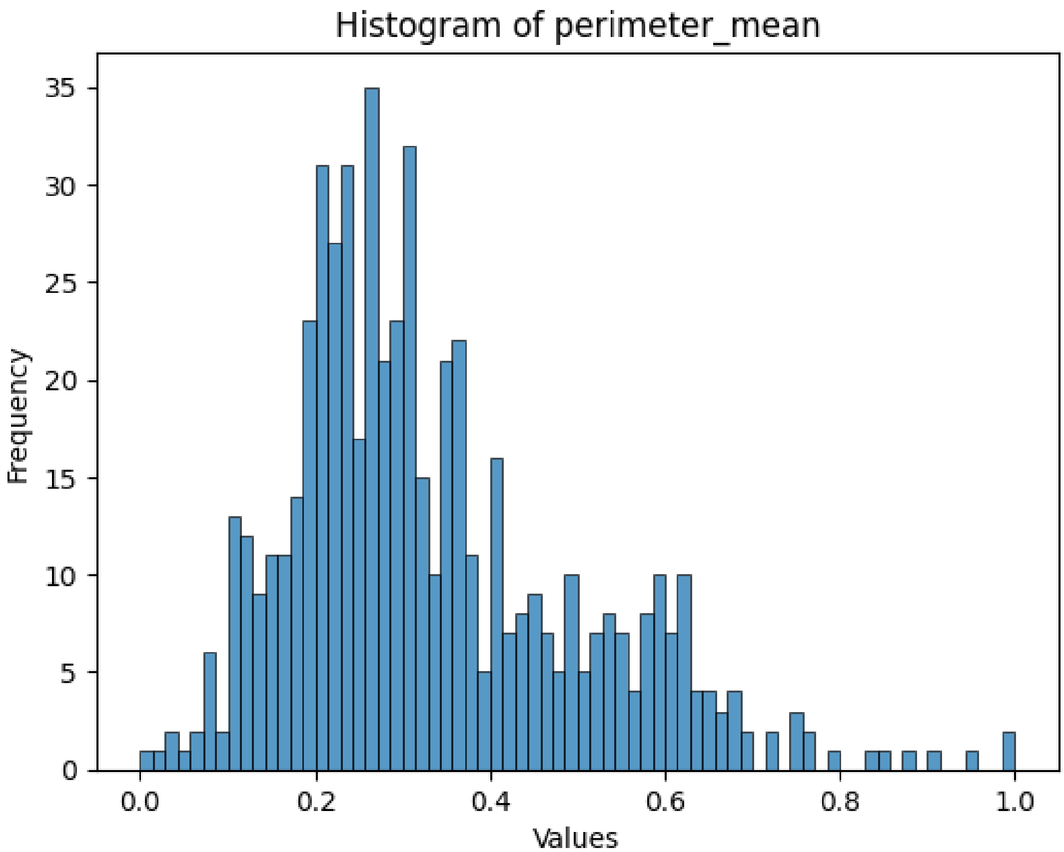
```
data.head()
```

```
   radius_mean  texture_mean  perimeter_mean  area_mean
smoothness_mean  \
0     0.521037      0.022658        0.545989   0.363733
0.593753
1     0.643144      0.272574        0.615783   0.501591
0.289880
2     0.601496      0.390260        0.595743   0.449417
0.514309
3     0.210090      0.360839        0.233501   0.102906
0.811321
4     0.629893      0.156578        0.630986   0.489290
0.430351

   compactness_mean  concavity_mean  concave points_mean
symmetry_mean  \
0          0.792037        0.703140             0.731113
0.686364
1          0.181768        0.203608             0.348757
0.379798
2          0.431017        0.462512             0.635686
0.509596
3          0.811361        0.565604             0.522863
0.776263
4          0.347893        0.463918             0.518390
0.378283

   fractal_dimension_mean  ...  radius_worst  texture_worst
perimeter_worst  \
0                0.605518  ...      0.620776       0.141525
0.668310
1                0.141323  ...      0.606901       0.303571
0.539818
2                0.211247  ...      0.556386       0.360075
0.508442
3                1.000000  ...      0.248310       0.385928
0.241347
4                0.186816  ...      0.519744       0.123934
0.506948

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0    0.450698          0.601136           0.619292         0.568610
```

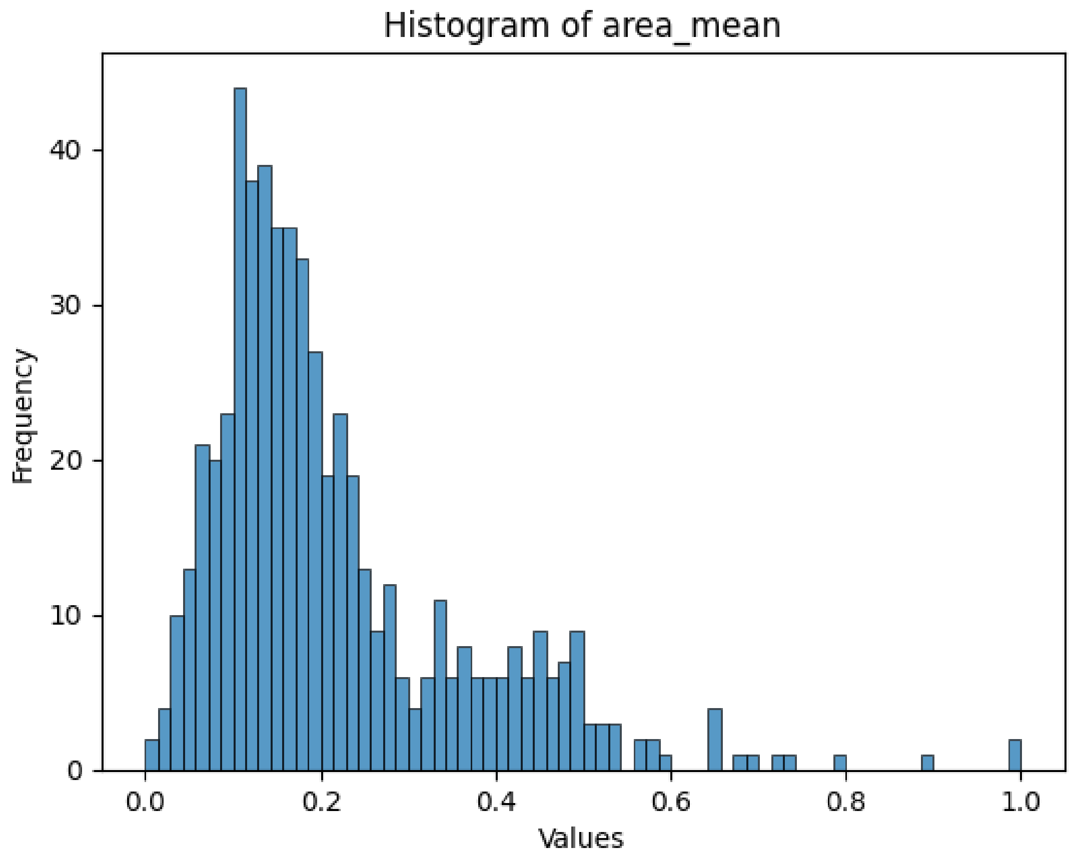
```
1    0.435214           0.347553            0.154563          0.192971
2    0.374508           0.483590            0.385375          0.359744
3    0.094008           0.915472            0.814012          0.548642
4    0.341575           0.437364            0.172415          0.319489

   concave points_worst  symmetry_worst  fractal_dimension_worst
0              0.912027        0.598462                 0.418864
1              0.639175        0.233590                 0.222878
2              0.835052        0.403706                 0.213433
3              0.884880        1.000000                 0.773711
4              0.558419        0.157500                 0.142595

[5 rows x 30 columns]

for column in data.columns:
    sns.histplot(data[column], bins=70)  # Adjust the number of bins
as needed
    plt.title(f'Histogram of {column}')
    plt.xlabel('Values')
    plt.ylabel('Frequency')
    plt.show()
```



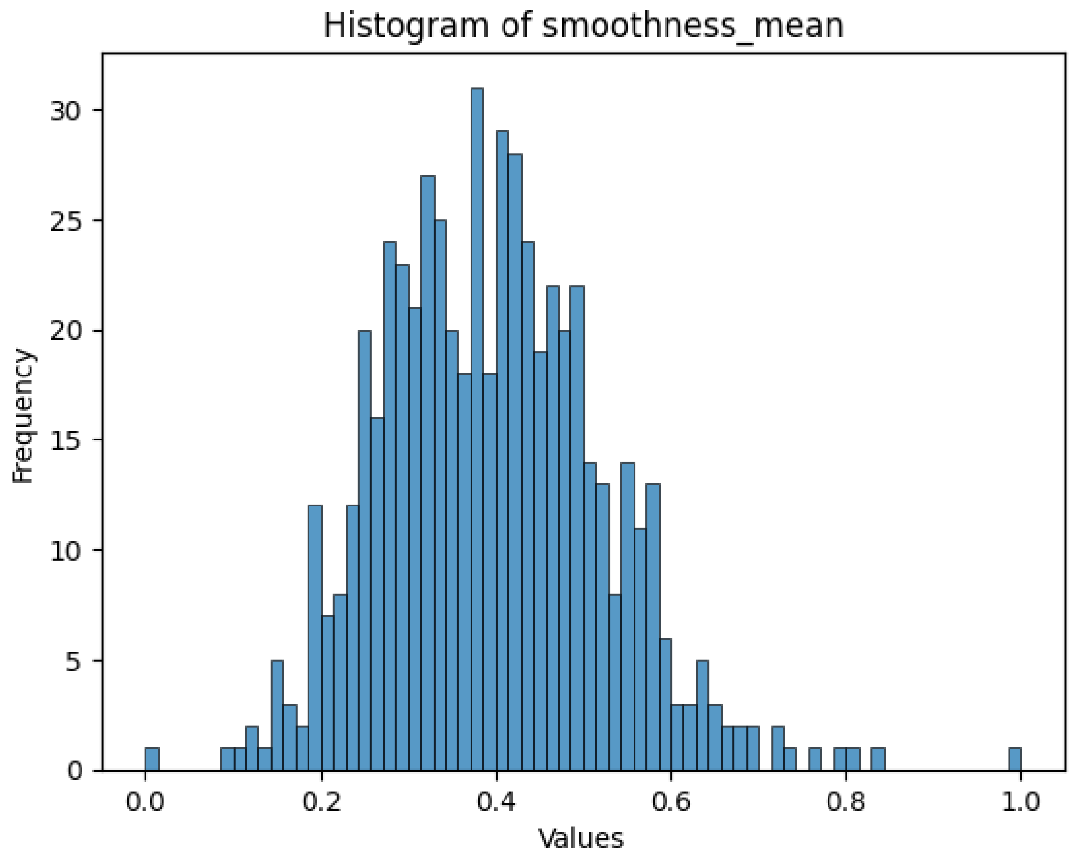

Histogram of texture_mean
Frequency
Values
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0

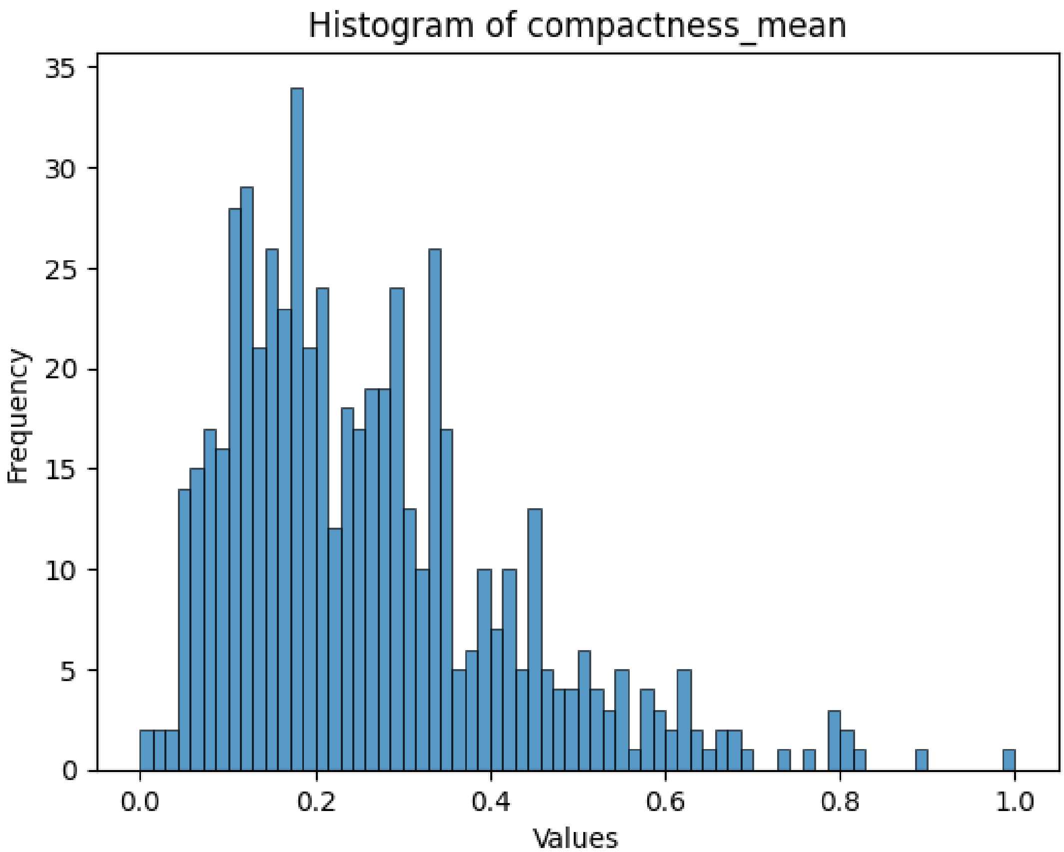


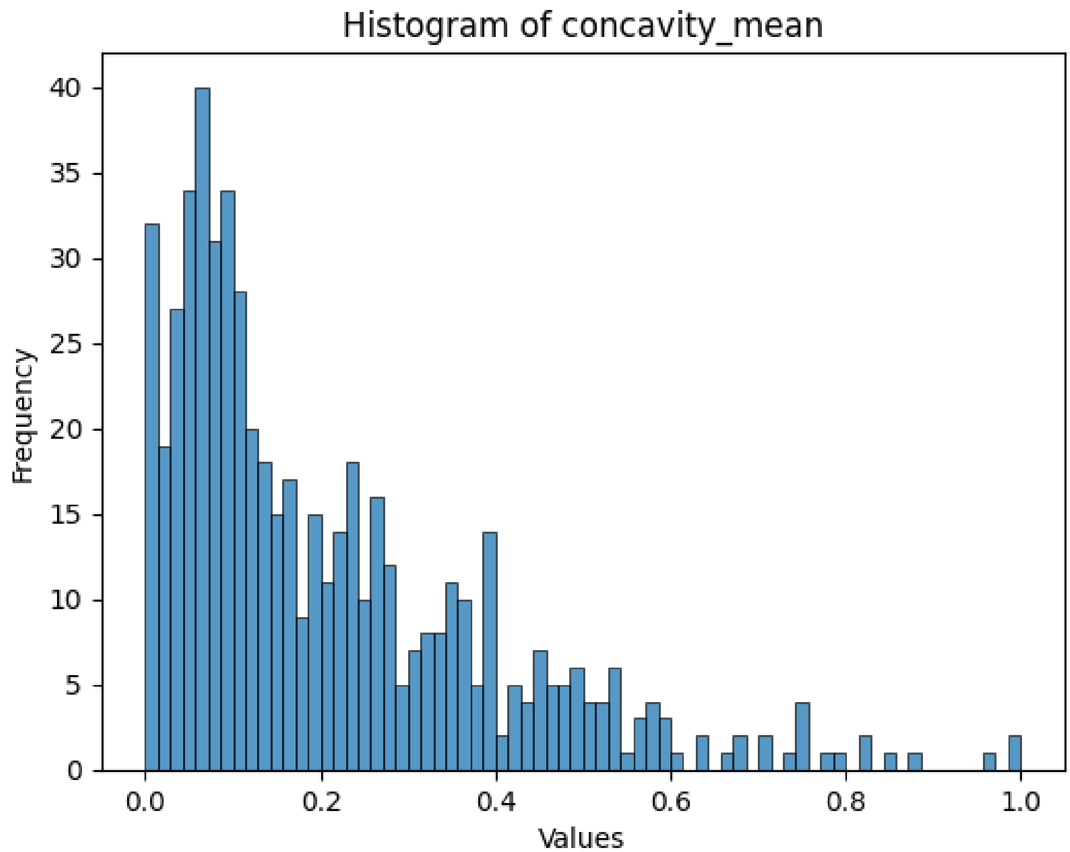
Histogram of perimeter_mean
Frequency
Values
0
5
10
15
20
25
30
35
0.0
0.2
0.4
0.6
0.8
1.0



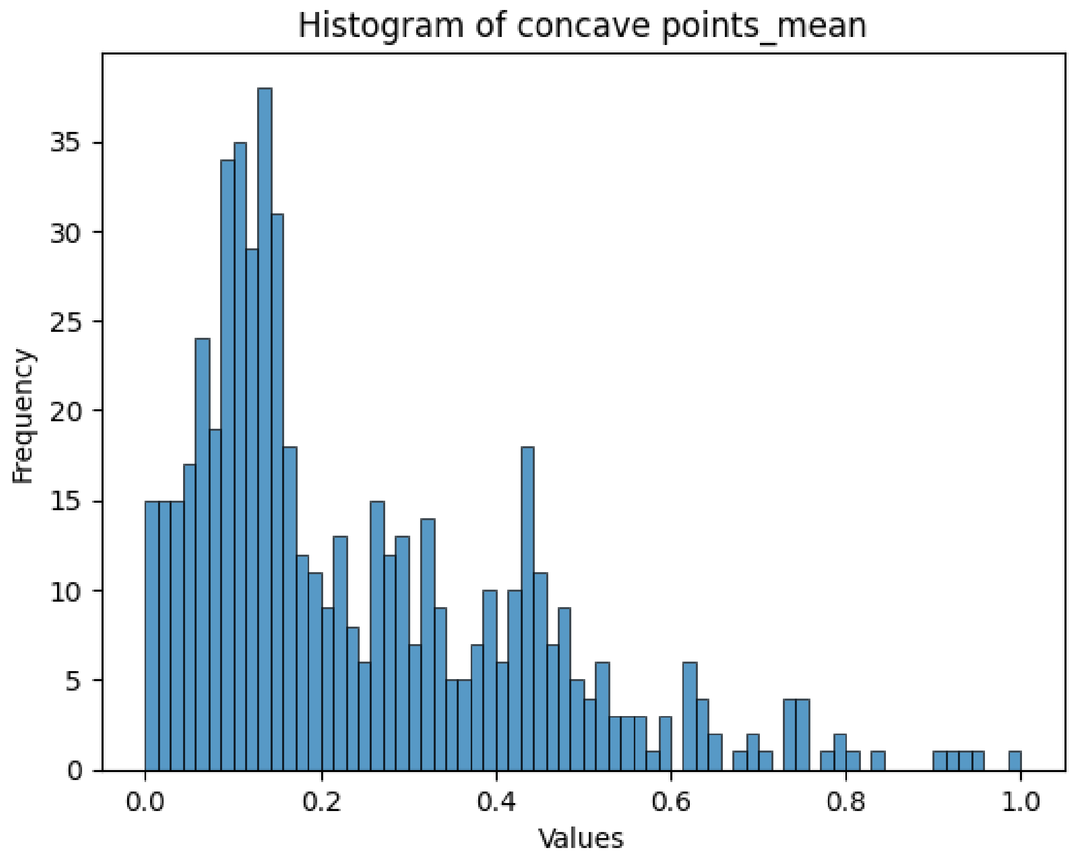
Histogram of area_mean
Frequency
Values
0
10
20
30
40
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of smoothness_mean
Frequency
Values
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0

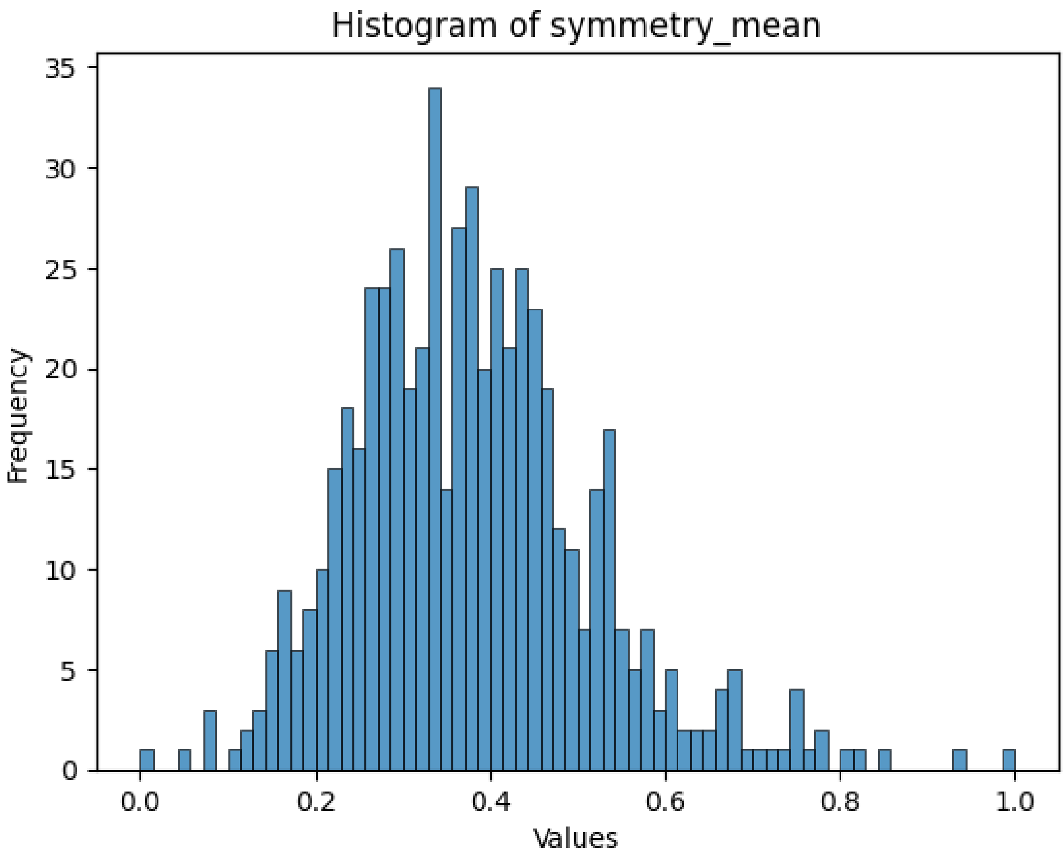

Histogram of compactness_mean
Frequency
Values
0
5
10
15
20
25
30
35
0.0
0.2
0.4
0.6
0.8
1.0

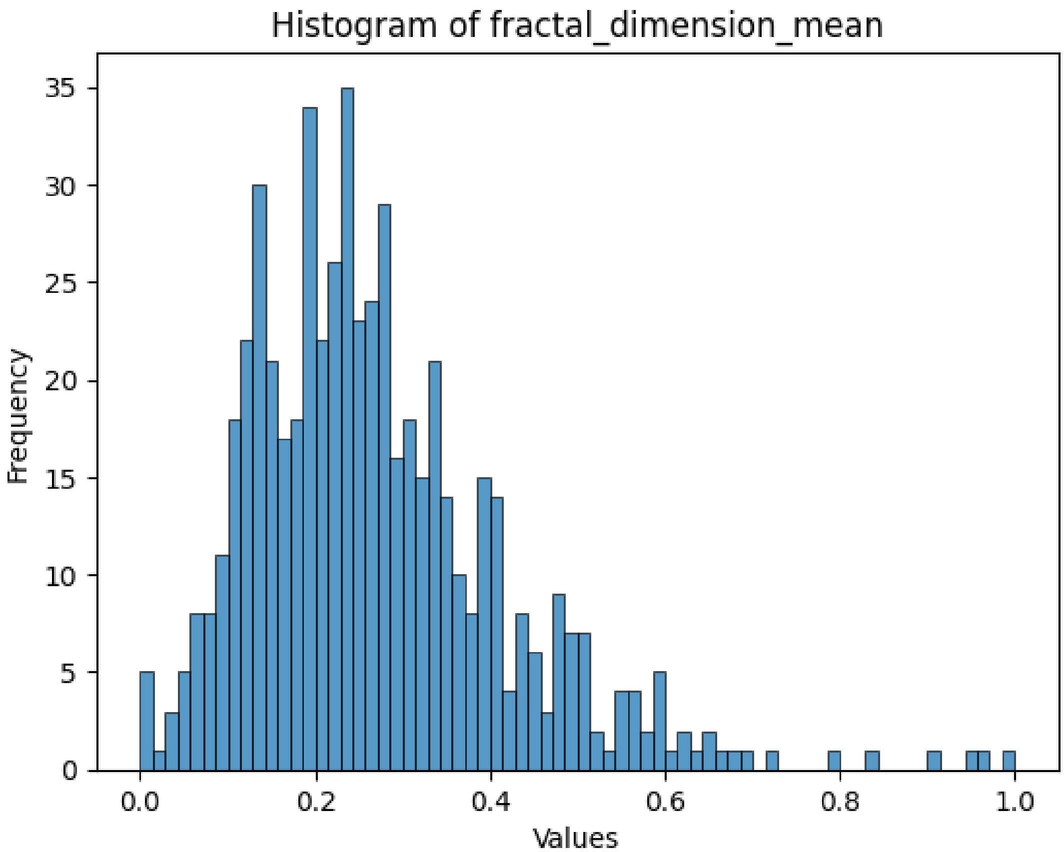


Histogram of concavity_mean
Frequency
Values
0
5
10
15
20
25
30
35
40
0.0
0.2
0.4
0.6
0.8
1.0

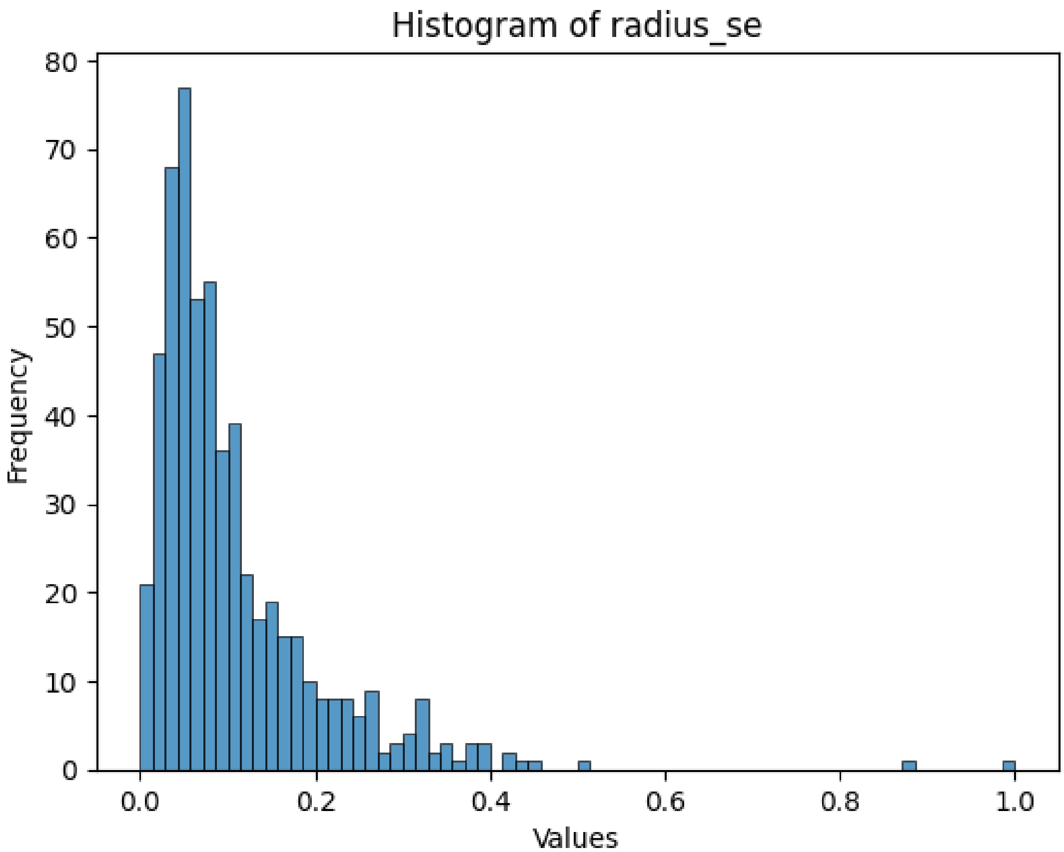


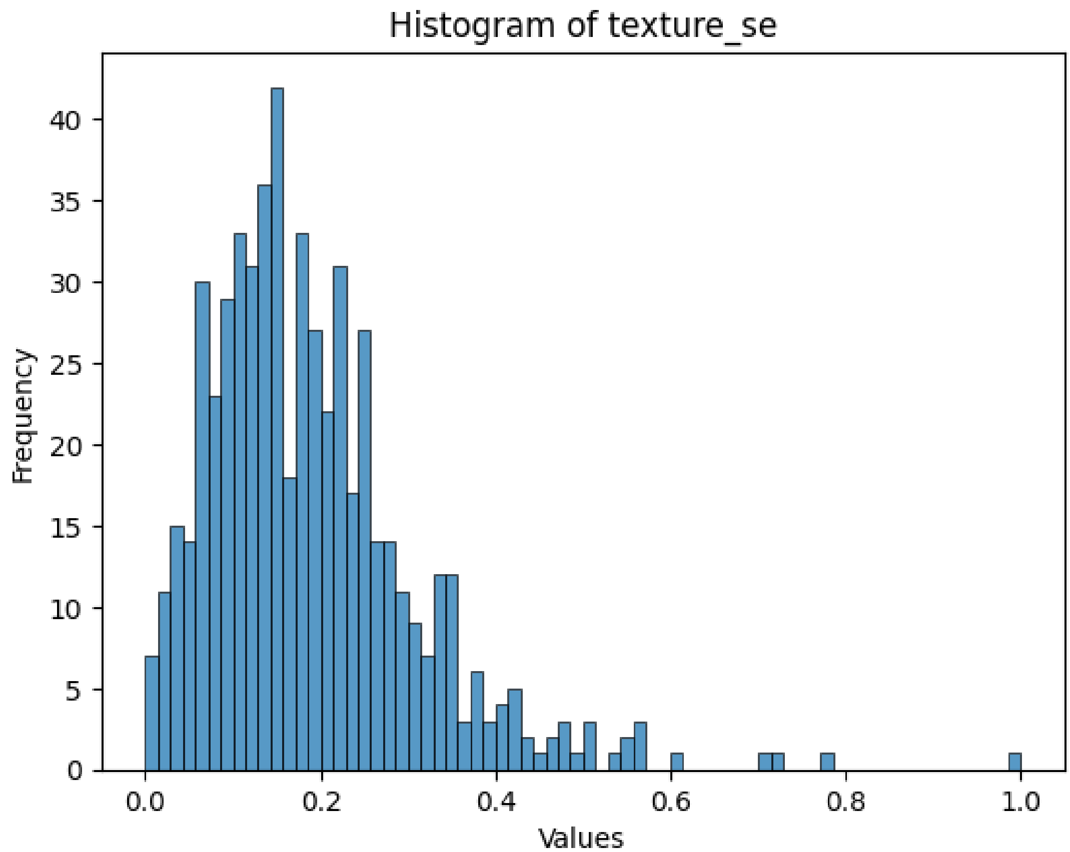
Histogram of concave points_mean
Frequency
Values
0
5
10
15
20
25
30
35
0.0
0.2
0.4
0.6
0.8
1.0

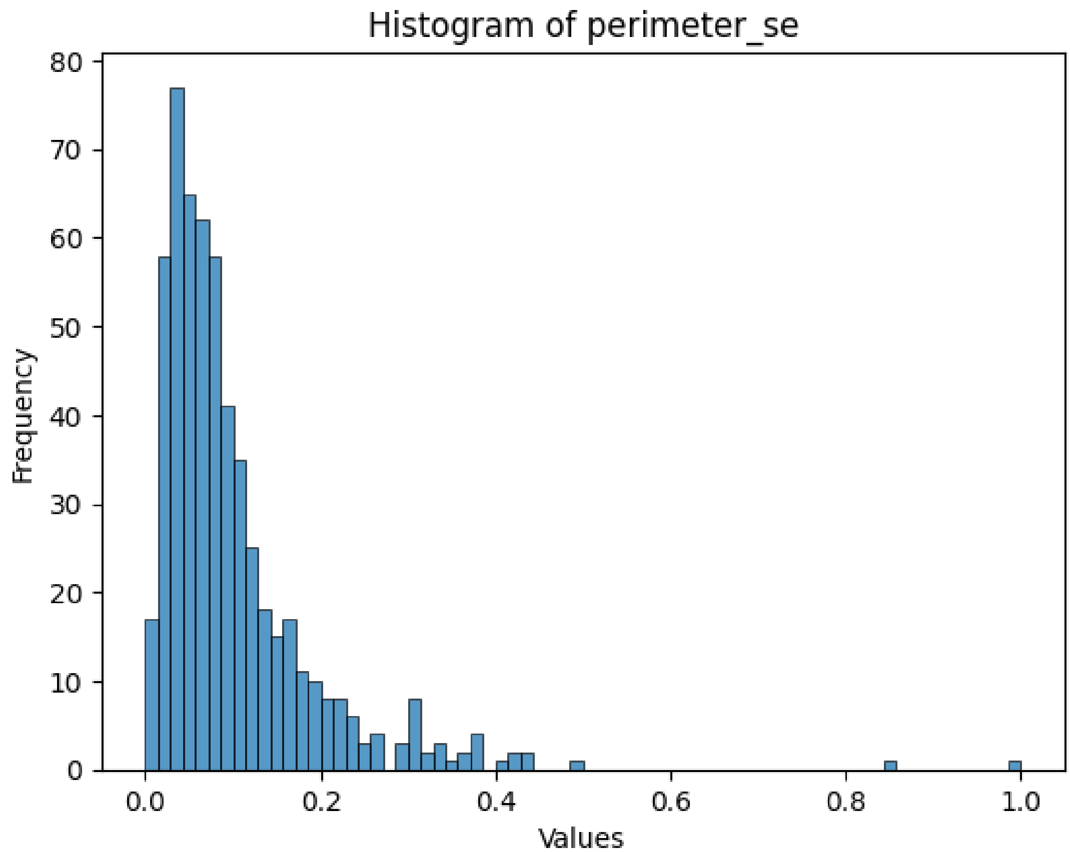

Histogram of symmetry_mean
Frequency
Values
0
5
10
15
20
25
30
35
0.0
0.2
0.4
0.6
0.8
1.0

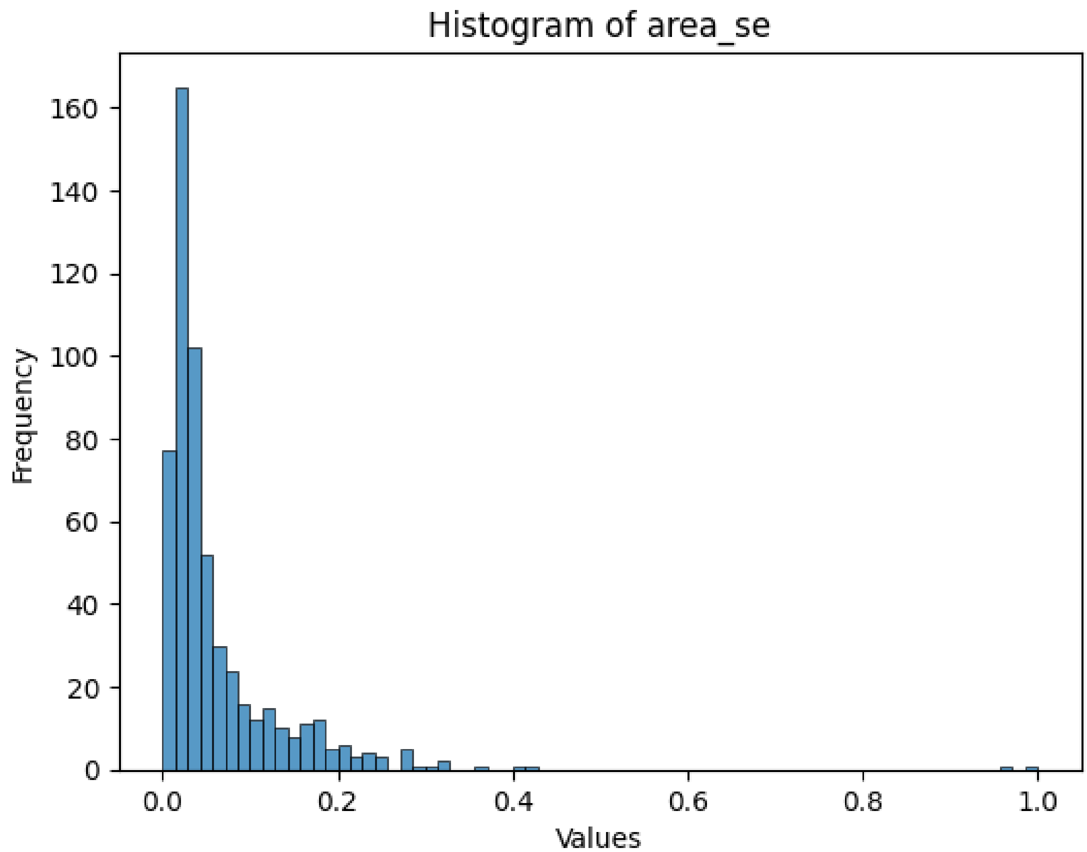


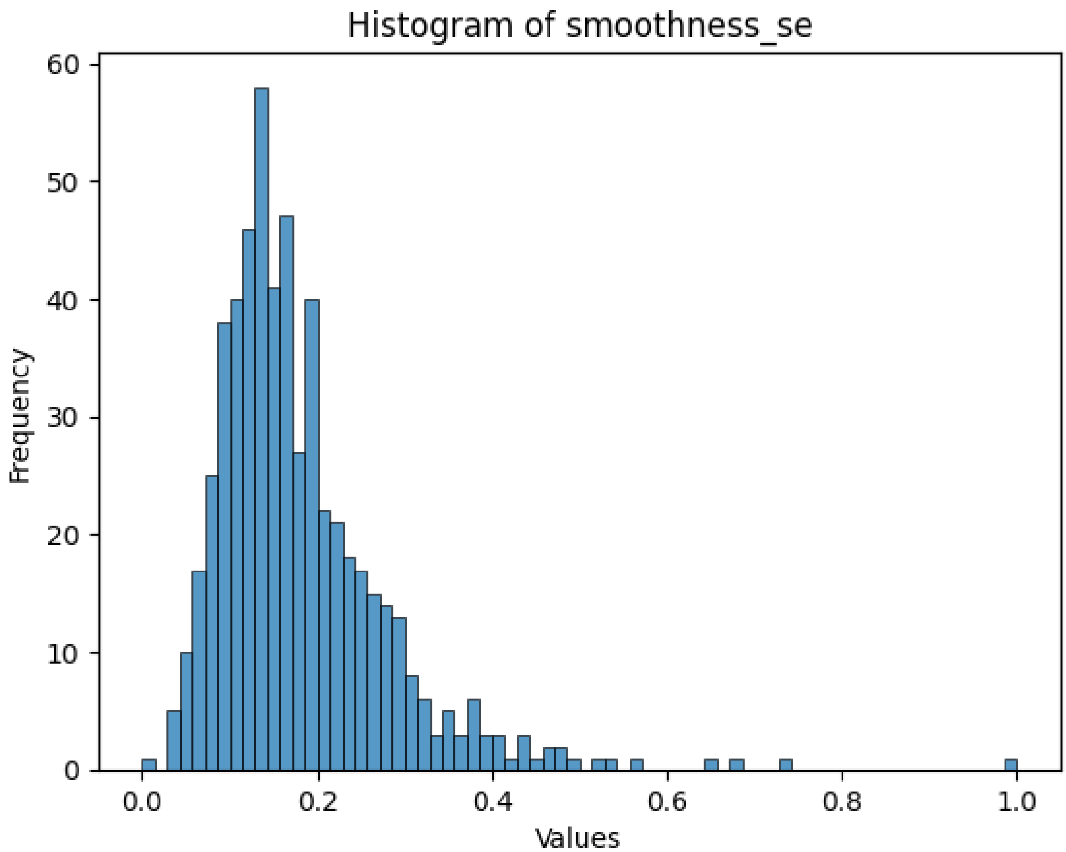
Histogram of fractal_dimension_mean
Frequency
Values
0
5
10
15
20
25
30
35
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of radius_se
Frequency
Values
0
10
20
30
40
50
60
70
80
0.0
0.2
0.4
0.6
0.8
1.0

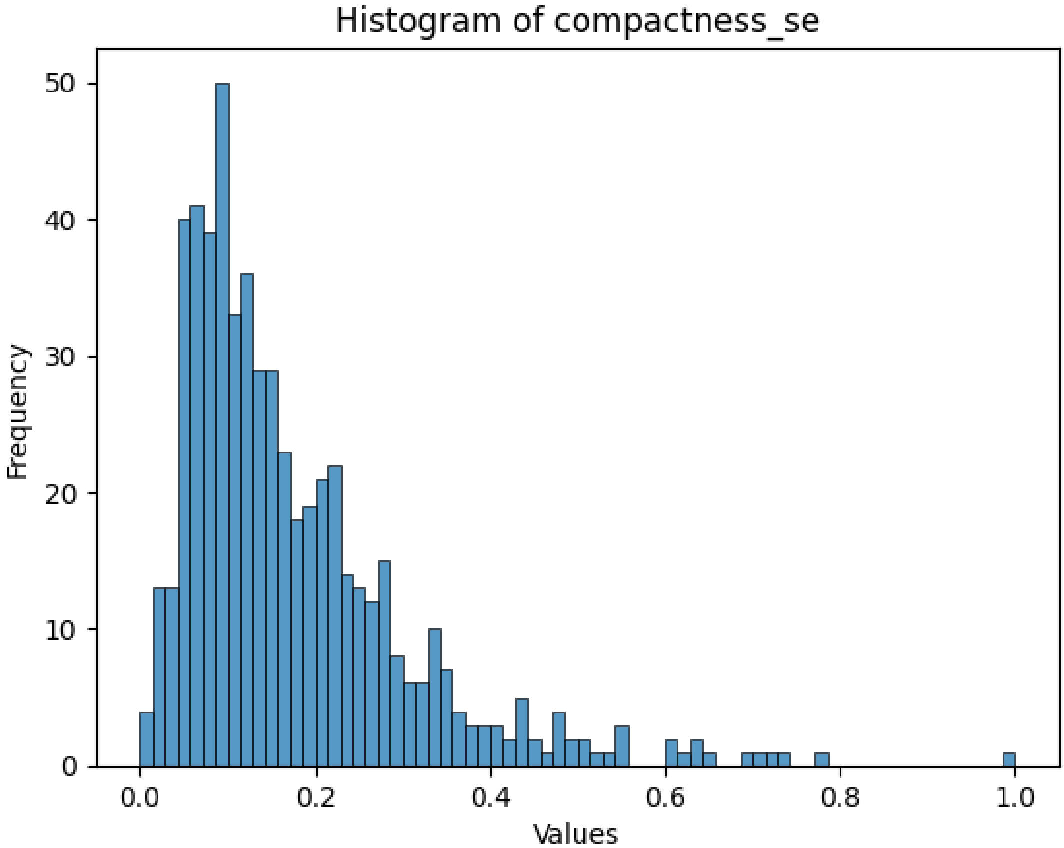

Histogram of texture_se
Frequency
Values
0
5
10
15
20
25
30
35
40
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of perimeter_se
Frequency
Values
0
10
20
30
40
50
60
70
80
0.0
0.2
0.4
0.6
0.8
1.0

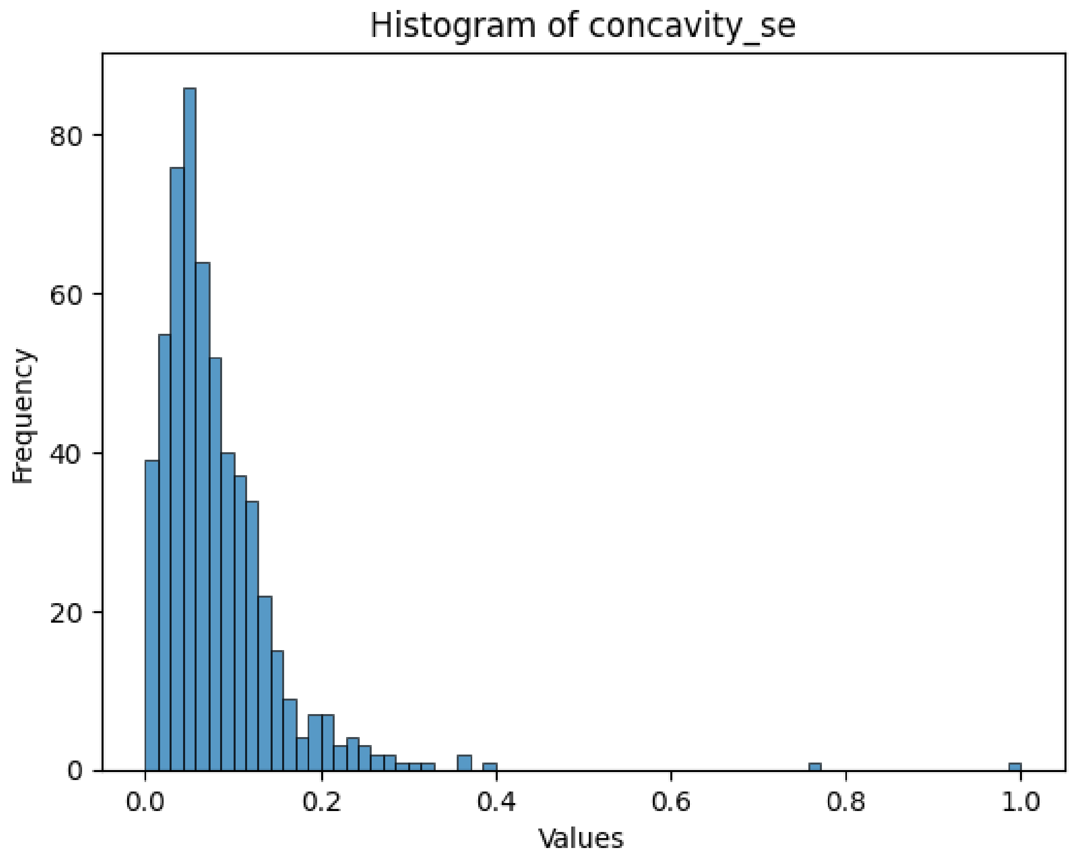

Histogram of area_se
Frequency
0
20
40
60
80
100
120
140
160
0.0
0.2
0.4
0.6
0.8
1.0
Values

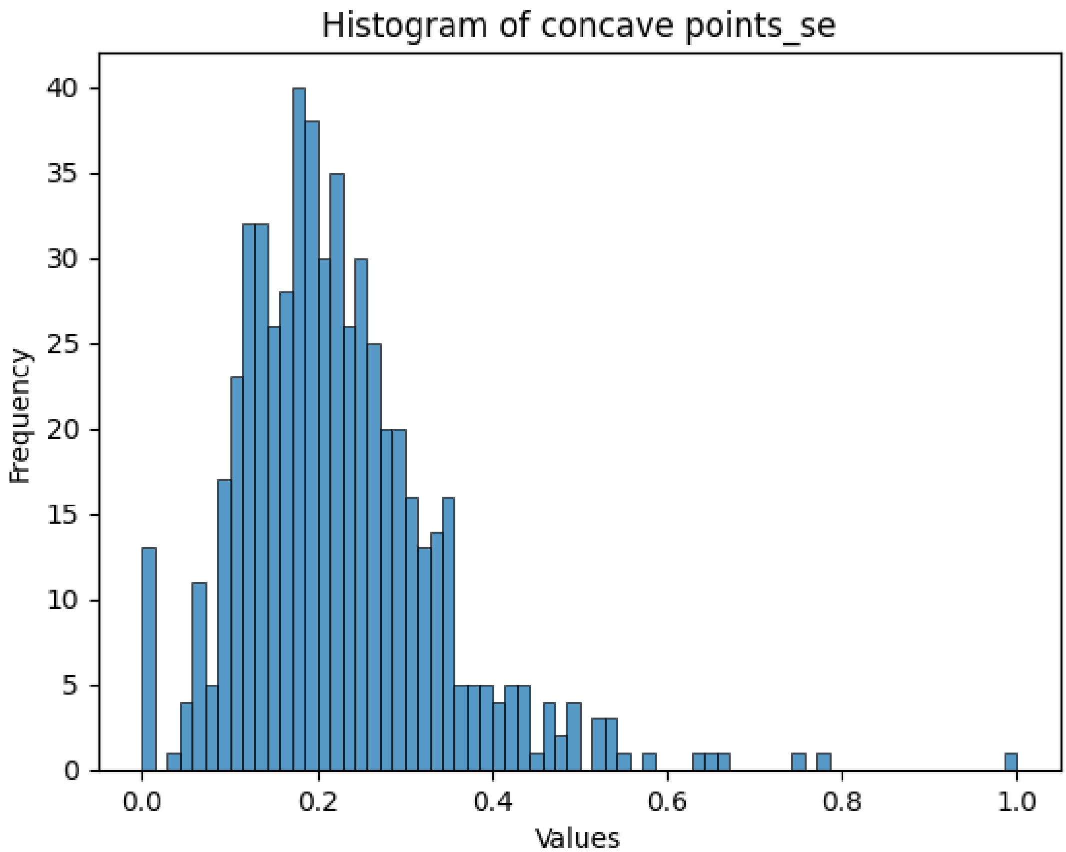


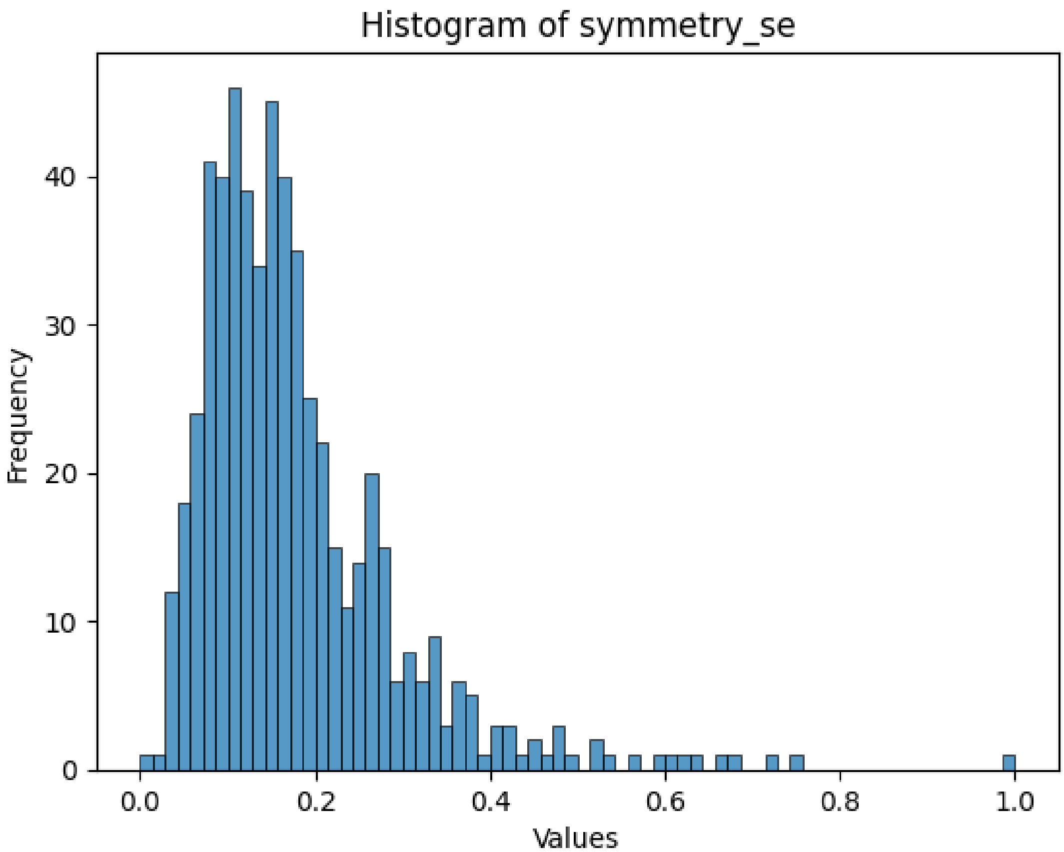
Histogram of smoothness_se
Frequency
Values
0
10
20
30
40
50
60
0.0
0.2
0.4
0.6
0.8
1.0

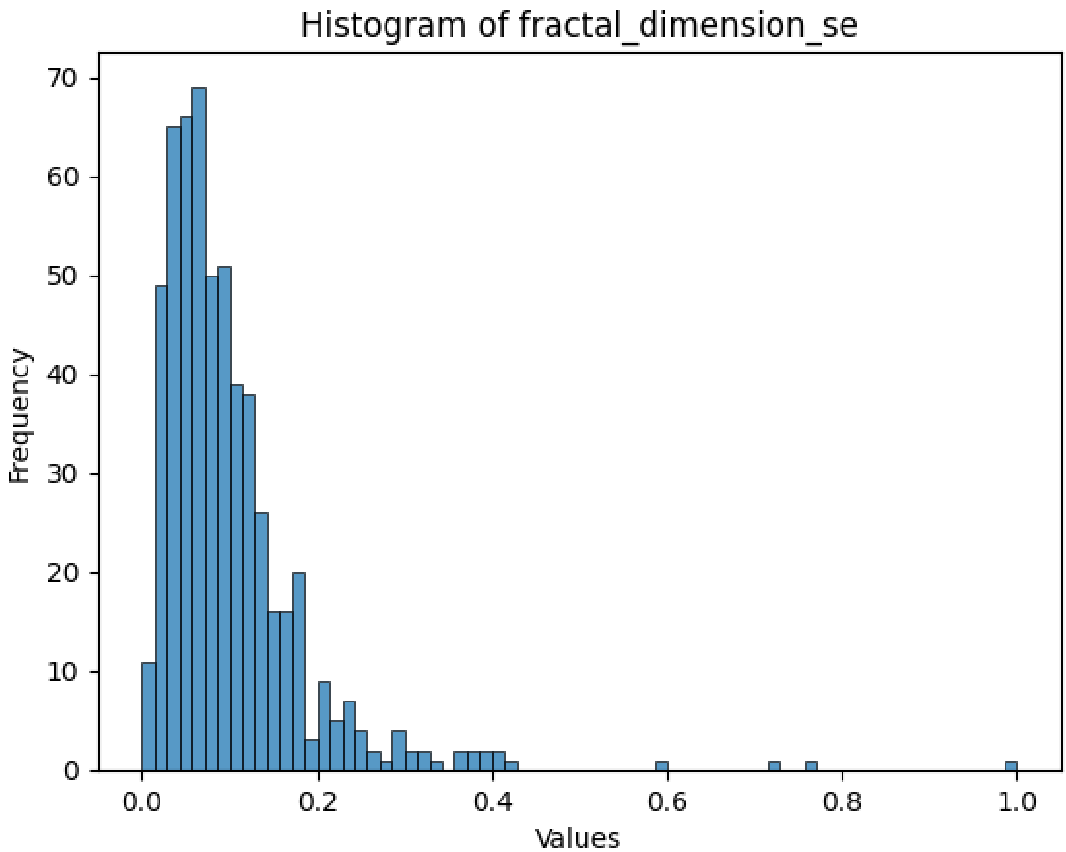


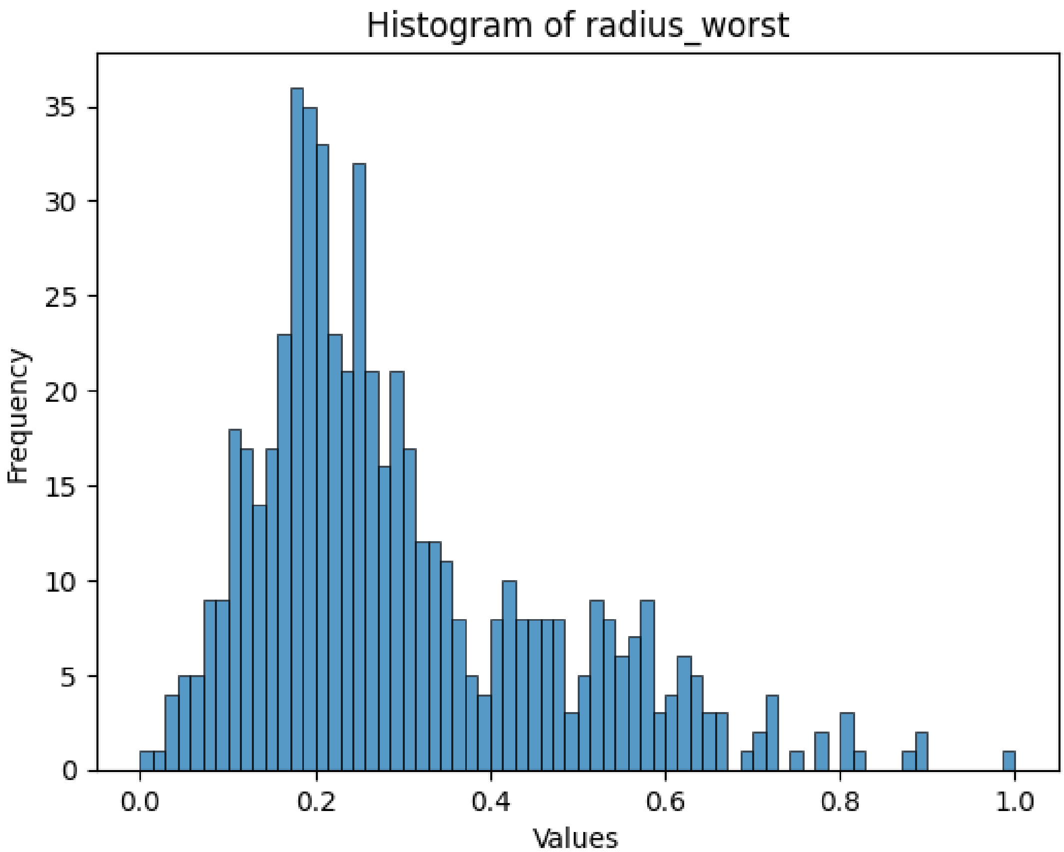
Histogram of compactness_se
Frequency
Values
0
10
20
30
40
50
0.0
0.2
0.4
0.6
0.8
1.0

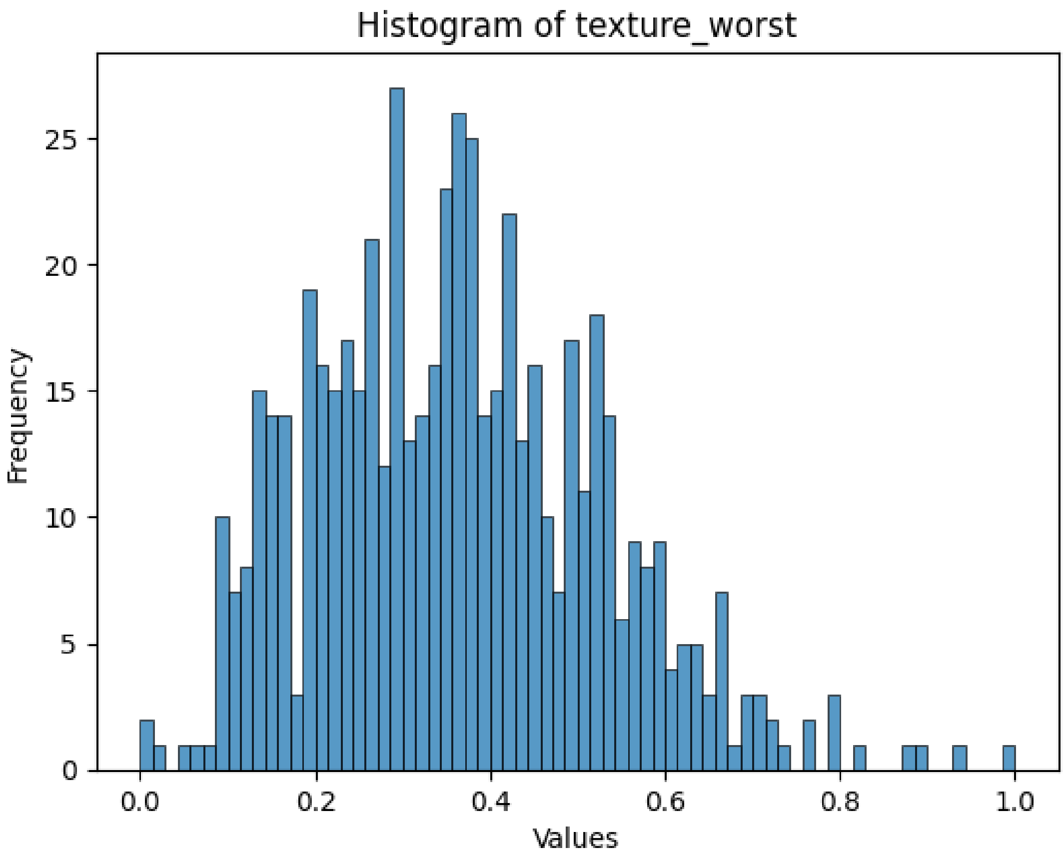

Histogram of concavity_se
Frequency
Values
0
20
40
60
80
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of concave points_se
Frequency
Values
0
5
10
15
20
25
30
35
40
0.0
0.2
0.4
0.6
0.8
1.0

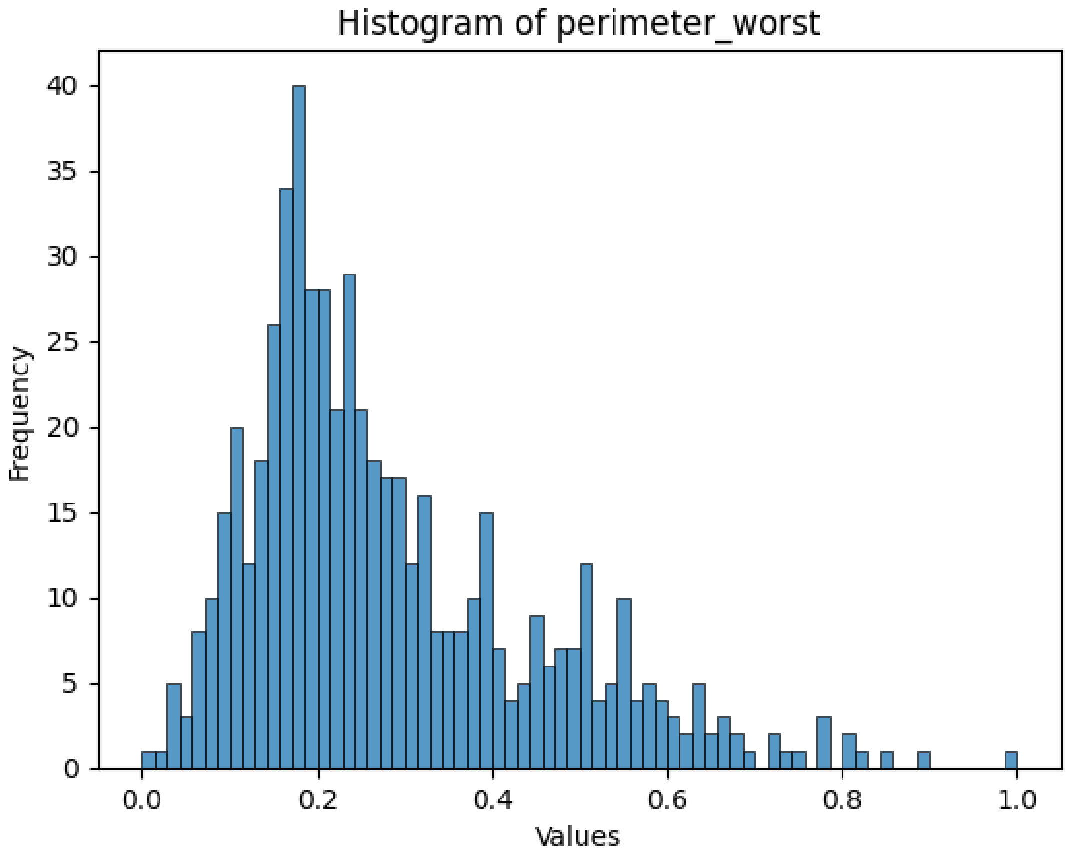

Histogram of symmetry_se
Frequency
Values
0
10
20
30
40
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of fractal_dimension_se
Frequency
Values
0
10
20
30
40
50
60
70
0.0
0.2
0.4
0.6
0.8
1.0

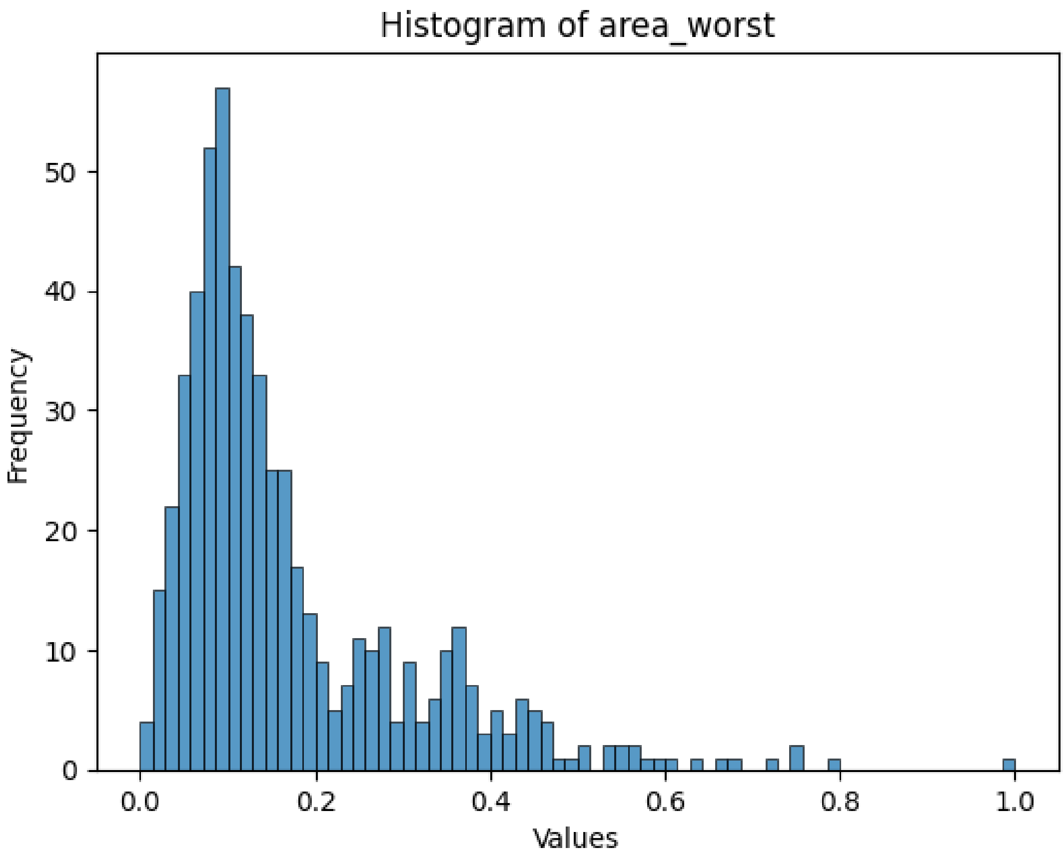

Histogram of radius_worst
Frequency
Values
0
5
10
15
20
25
30
35
0.0
0.2
0.4
0.6
0.8
1.0

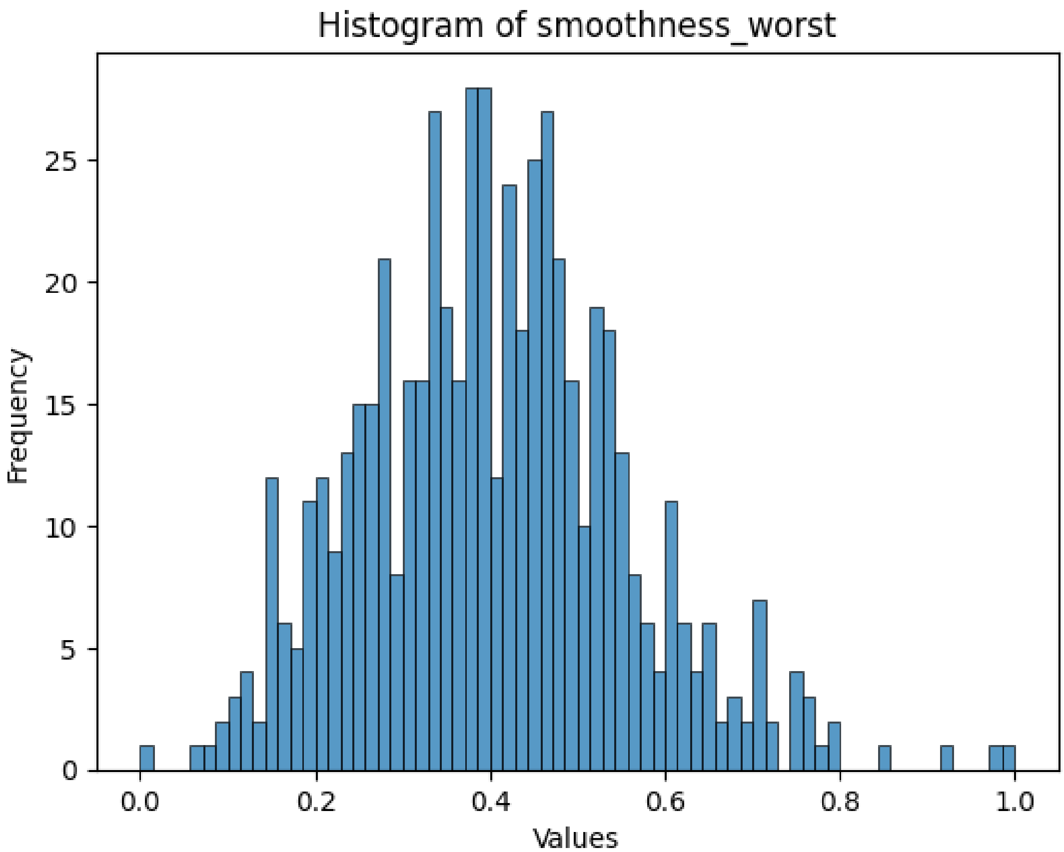


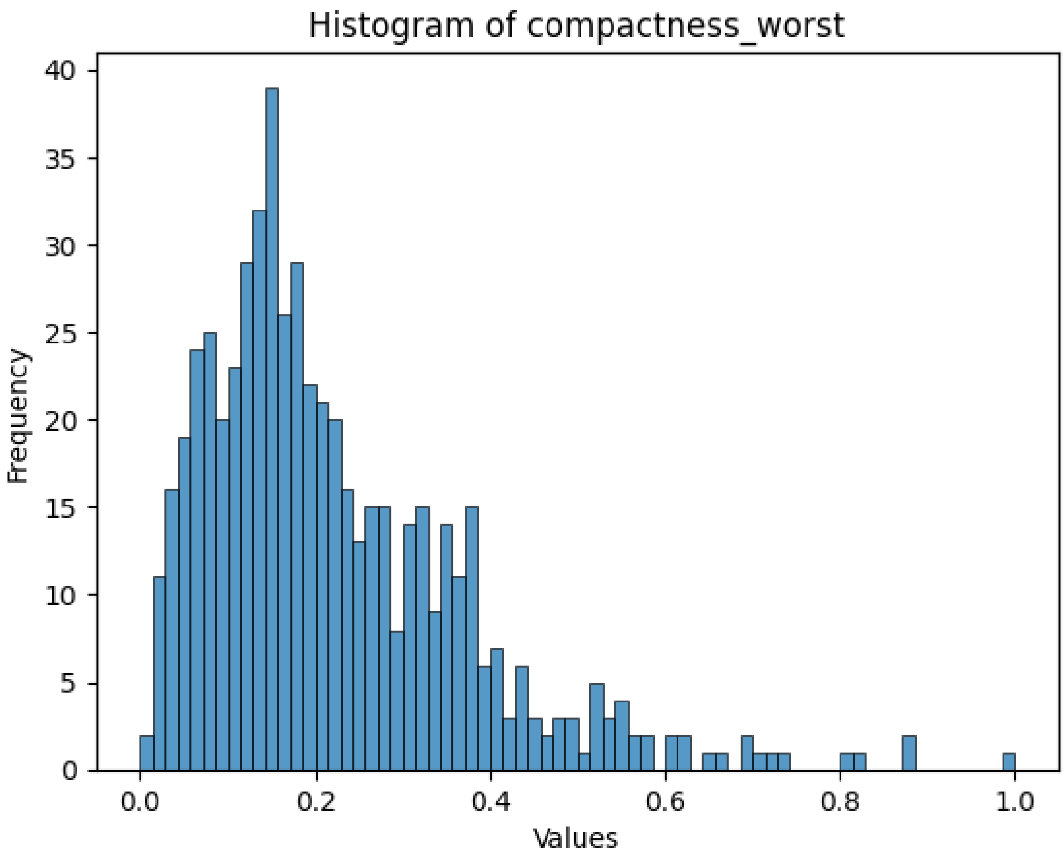
Histogram of texture_worst
Frequency
Values
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0

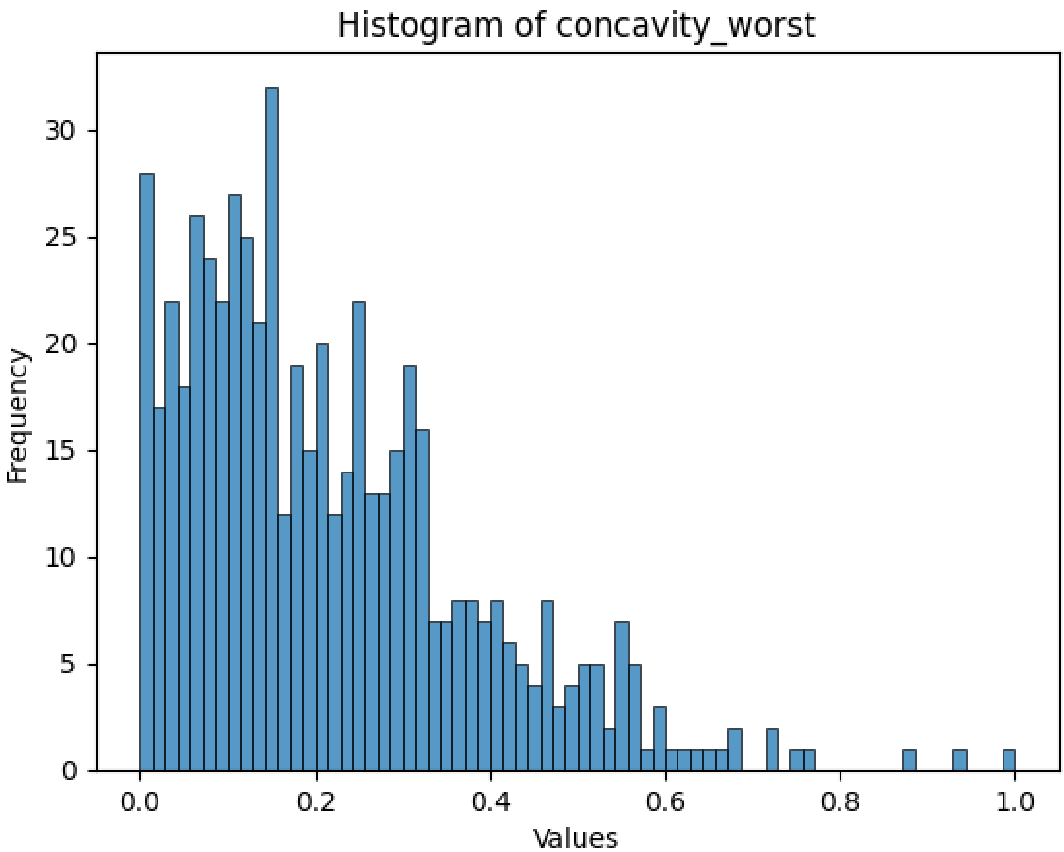


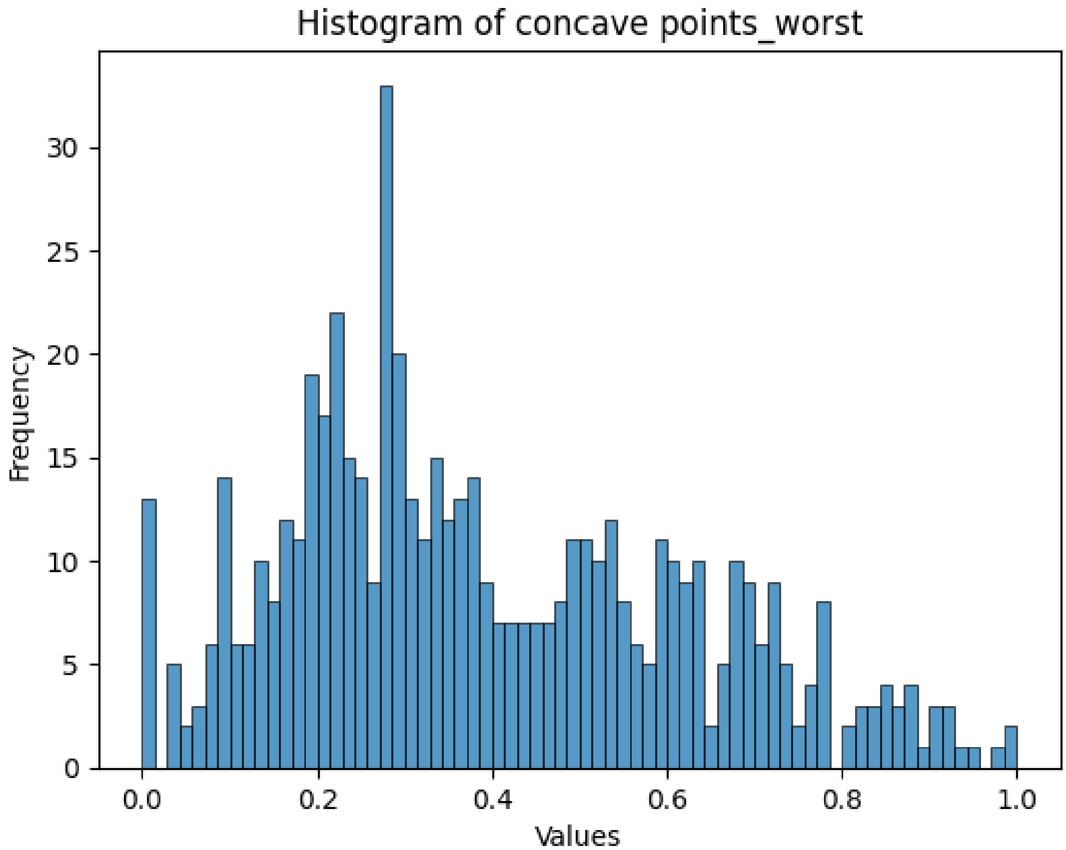
Histogram of perimeter_worst
Frequency
Values
0
5
10
15
20
25
30
35
40
0.0
0.2
0.4
0.6
0.8
1.0

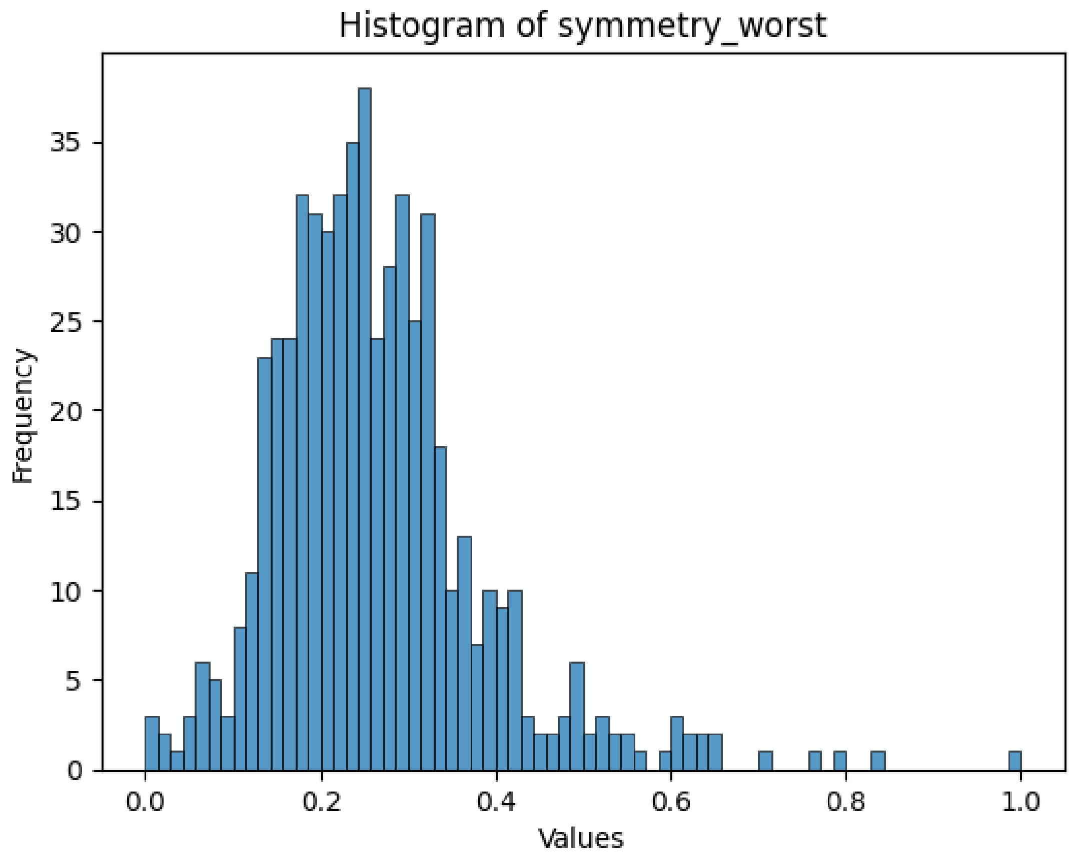

Histogram of area_worst
Frequency
0
10
20
30
40
50
0.0
0.2
0.4
0.6
0.8
1.0
Values

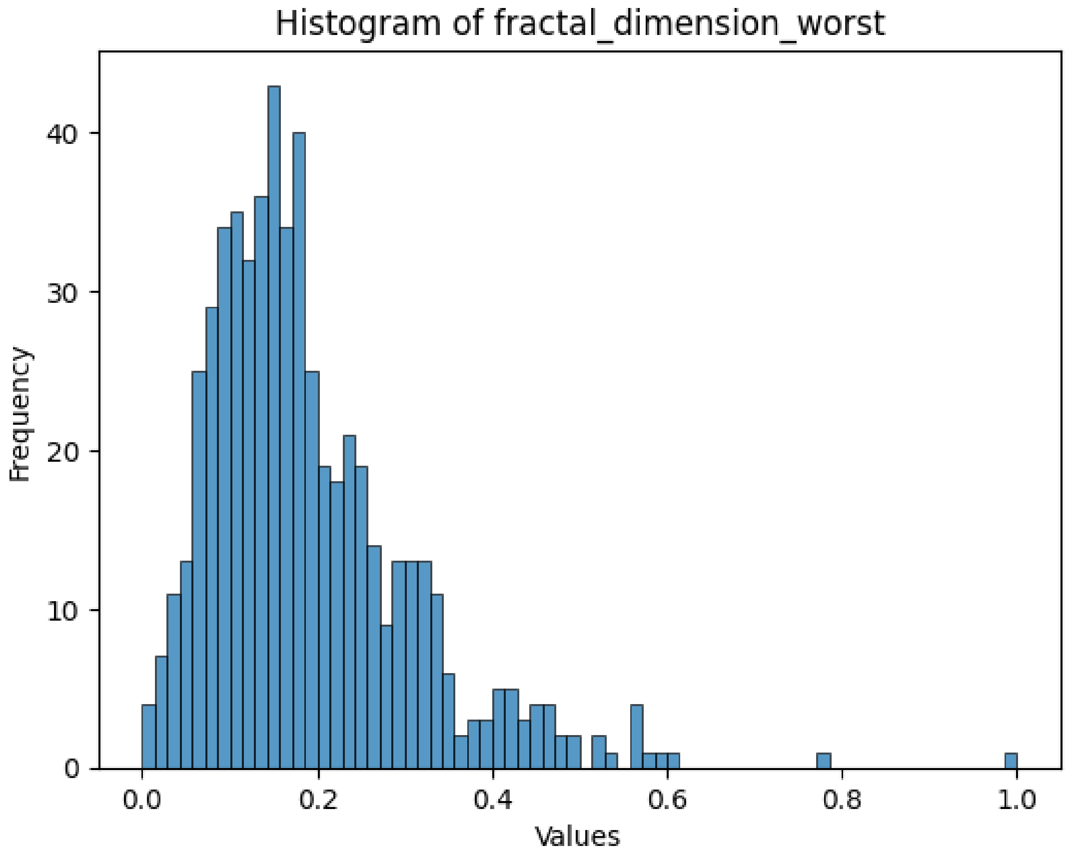

Histogram of smoothness_worst
Frequency
Values
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of compactness_worst
Frequency
Values
0
5
10
15
20
25
30
35
40
0.0
0.2
0.4
0.6
0.8
1.0

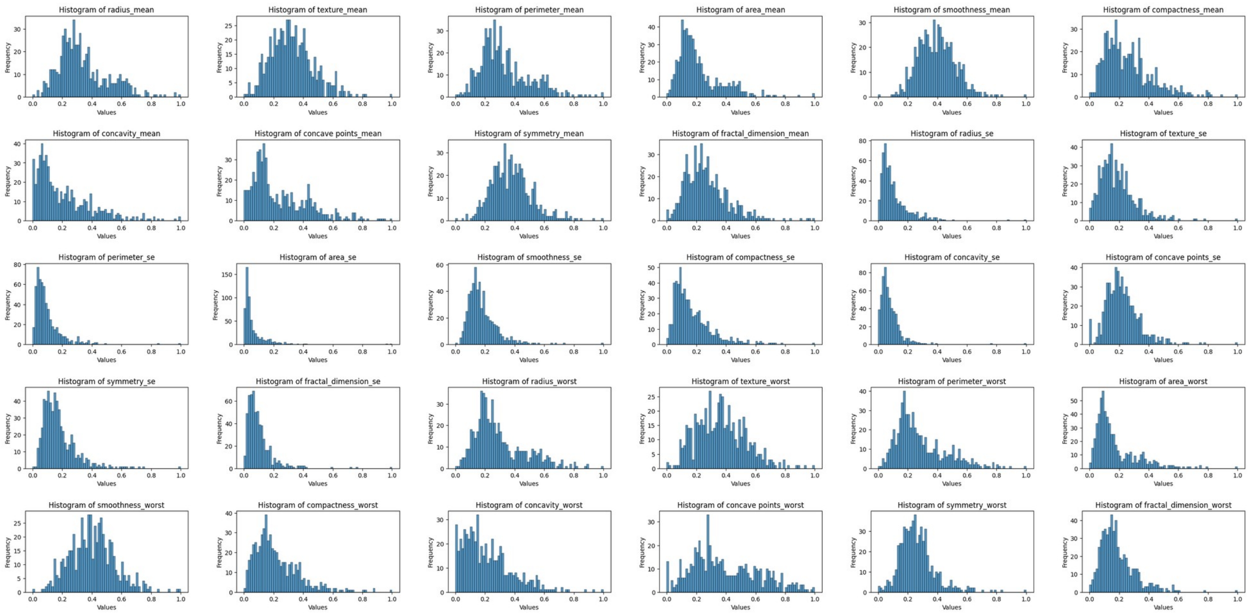

Histogram of concavity_worst
Frequency
Values
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of concave points_worst
Frequency
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0
Values


Histogram of symmetry_worst
Frequency
Values
0
5
10
15
20
25
30
35
0.0
0.2
0.4
0.6
0.8
1.0




```python
# Showing the Histograms as a Grid Format
n_rows = 5
n_cols = 6
fig, axes = plt.subplots(n_rows, n_cols, figsize=(30, 15))
fig.tight_layout(pad=5.0)
axes = axes.flatten()
for i, column in enumerate(data.columns):
    if i < len(axes):
        sns.histplot(data[column], bins=70, ax=axes[i])
        axes[i].set_title(f'Histogram of {column}')
        axes[i].set_xlabel('Values')
        axes[i].set_ylabel('Frequency')
    else:
        break
plt.show()
```

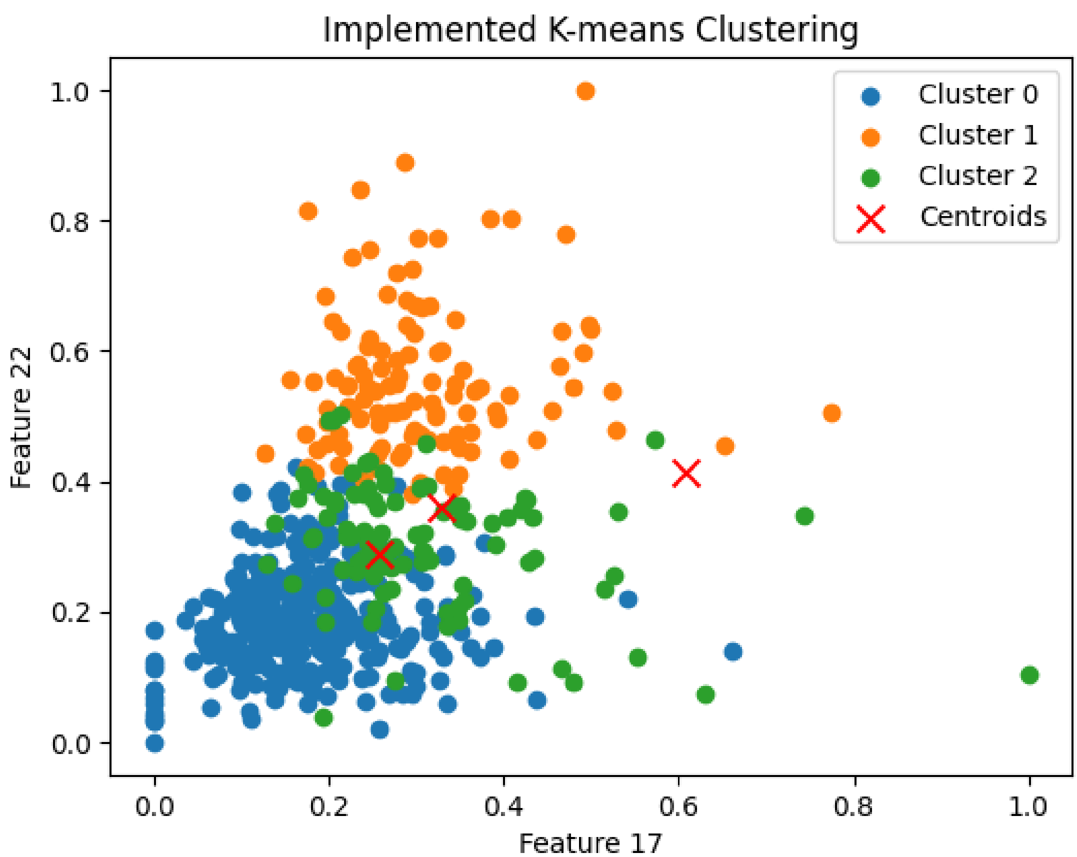


4.) K-means Clustering

K-means Implemented Function

```
# Manually implemented KMeans function
def custom_kmeans(data, k, max_iter=300, n_init=10, random_state=42):
    np.random.seed(random_state)

    # Randomly initialize centroids
    # This is done by selecting K-Unique Datapoints from the Dataset
    centroids = data[np.random.choice(data.shape[0], k, replace=False)]

    for _ in range(max_iter):
        # Assign each data point to the nearest centroid
        labels = np.argmin(np.linalg.norm(data[:, np.newaxis] - centroids, axis=2), axis=1)

        # Update centroids based on the mean of data points in each cluster
        new_centroids = np.array([data[labels == j].mean(axis=0) for j in range(k)])

        # If centroids don't change significantly, break
        if np.allclose(centroids, new_centroids):
            break

        centroids = new_centroids

    # Calculate the sum of squared errors (SSE)
```

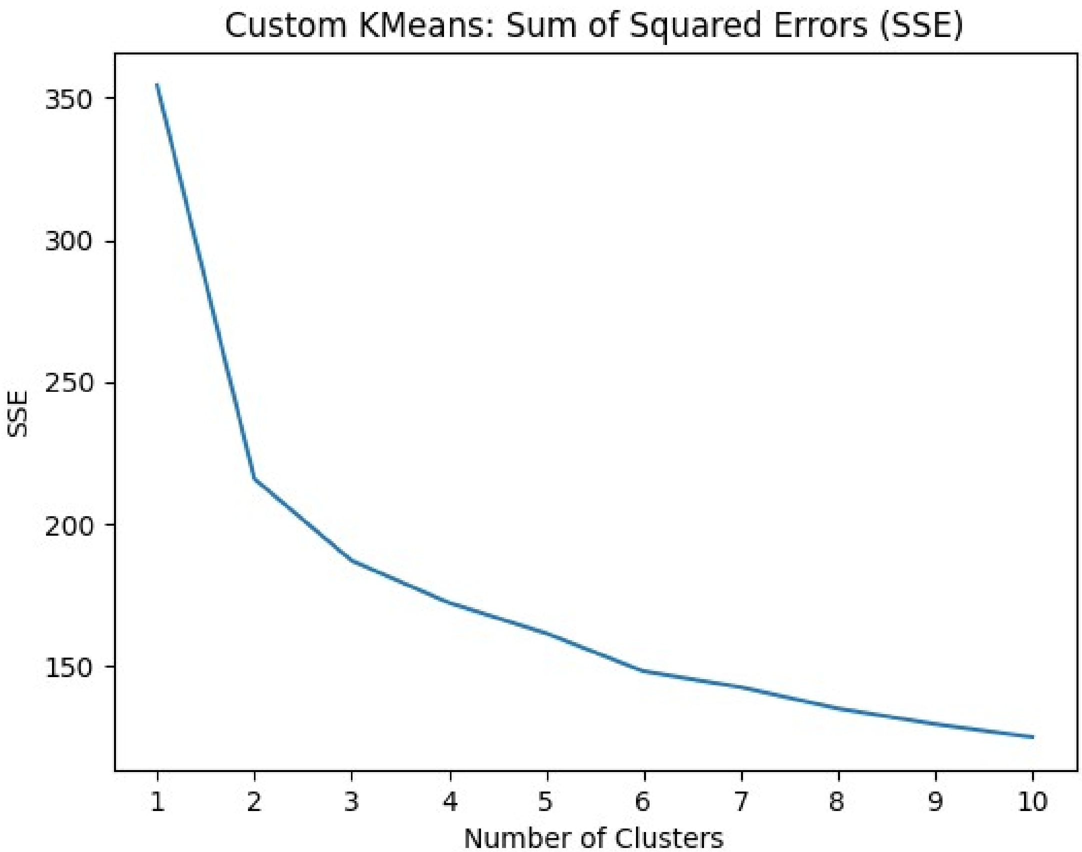
```
    sse = np.sum([np.sum((data[labels == j] - centroids[j])**2) for j
in range(k)])

    return labels, centroids, sse

k = 3 # Number of clusters
labels, centroids, sse = custom_kmeans(df_scaled, k)

import random
a = random.randint(0,29)
b = random.randint(0,29)

# Step 2: Visualization
# Scatter plot for each cluster
for i in range(k):
    plt.scatter(df_scaled[labels == i, a], df_scaled[labels == i, b],
label=f'Cluster {i}')

# Plot the centroids
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='x',
s=100, label='Centroids')

plt.title('Implemented K-means Clustering')
plt.xlabel(f'Feature {a}')
plt.ylabel(f'Feature {b}')
plt.legend()
plt.show()
print("SSE =", sse)
```




```
SSE = 187.03025264441408
```

No PCA

```python
# Create a list to hold SSE values for each k
sse_custom = []

# Iterate over different values of k
for k in range(1, 11):
    labels, centroids, sse = custom_kmeans(df_scaled, k)
    sse_custom.append(sse)

# Visualize results
plt.plot(range(1, 11), sse_custom)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Custom KMeans: Sum of Squared Errors (SSE)")
plt.show()
```




```
temp = pd.DataFrame(sse_custom)
temp.head(10)

            0
0  354.436613
1  215.838320
2  187.030253
3  172.172287
4  161.445437
5  148.049740
6  142.450486
7  134.925103
8  129.431043
9  124.852063

#find the elbow point programmatically
kl = KneeLocator(range(1, 11), sse_custom, curve="convex",
direction="decreasing")
k_optimal = kl.elbow
k_optimal

3
```

```
labels_custom, centroids_custom, sse_optimal= custom_kmeans(df_scaled,
k_optimal)

# View cluster assignments for each observation
print("Cluster Assignments (custom implementation):", labels_custom)

# Print the centroid centers
print("Centroid Centers (custom implementation):")
print(centroids_custom)

# Print SSE for k=3
print(f"SSE for k=3: {sse_optimal}")
```

```
Cluster Assignments (custom implementation): [1 1 1 2 1 2 1 2 2 2 0 2
1 0 2 2 0 2 1 0 0 0 2 1 1 1 2 1 2 1 1 2 1 1 2 2 2
 0 0 2 0 2 1 2 2 1 0 2 0 0 0 0 0 1 0 0 1 2 0 0 0 0 2 0 2 2 0 0 2 0 1 0
2 0
 0 1 0 1 1 0 0 2 1 1 0 1 0 1 0 2 0 0 0 0 2 1 0 0 0 2 0 0 0 0 0 2 0 0 1
0 0
 0 2 0 0 0 0 2 2 0 0 1 1 0 0 0 0 1 2 1 0 2 2 0 1 0 0 0 2 0 0 0 0 0 0 0
2 0
 0 0 0 2 2 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 0 2 0 0 0 2 2 0 0 1 1 0
0 0
 0 1 0 0 0 2 0 0 2 2 0 2 0 1 2 0 1 1 2 0 0 0 0 2 0 1 0 1 2 2 2 0 0 1 1
0 0
 0 2 0 0 0 0 0 2 2 0 0 1 0 0 1 1 0 1 0 0 2 0 1 0 0 2 0 0 1 0 1 1 1 2 1
2 2
 2 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 2 0 0 0 0 0 0 2 0 0 0
0 0
 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 1 0 1 0 0 0 0 2 2 2
0 0
 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 2 1 2 0 0 2 0 0 0 0 0 0 0 0 1 1 0
1 1
 2 0 1 1 0 0 2 0 0 2 0 0 0 0 0 0 0 0 0 1 0 0 2 1 0 0 0 0 0 0 2 0 0 0 0
0 0
 0 1 0 0 0 0 0 0 0 0 1 0 0 0 2 0 0 0 0 0 0 0 0 2 0 1 1 0 2 0 0 0 0 2 1
0 0
 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 2 0 0 1 2 0 0 0 0 0 0 0 0 0
2 0
 0 0 0 0 2 0 1 0 0 0 0 1 0 0 0 2 0 1 1 0 2 0 1 2 2 0 0 0 2 0 0 2 0 0 0
1 1
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 2 2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 2 1 1 1 1 1 0]
Centroid Centers (custom implementation):
[[0.25724828 0.28710408 0.24811606 0.14533507 0.3504499  0.16938359
  0.09500438 0.12511818 0.33460925 0.24360571 0.06453055 0.18689068
  0.05899374 0.02911313 0.17803893 0.12252456 0.05391389 0.17487179
  0.17145216 0.07725531 0.20673403 0.31839935 0.19261204 0.1005868
  0.348234   0.13842304 0.12103526 0.25288623 0.22316342 0.14417805]
```

```
 [0.60746655 0.41285811 0.60412204 0.46401606 0.44084025 0.40220719
  0.42966261 0.52203076 0.43287338 0.22057784 0.24153734 0.19412401
  0.22440445 0.17826652 0.16831466 0.23117247 0.11045207 0.30779724
  0.17608086 0.10831564 0.58101464 0.45240158 0.5575379  0.40473191
  0.45265847 0.31918203 0.36195658 0.67762641 0.29811009 0.19798636]
 [0.32859804 0.35886949 0.33550506 0.19783095 0.50776948 0.43840645
  0.37465026 0.36156033 0.48666351 0.42916096 0.1058611  0.19287455
  0.10536427 0.05379246 0.20764    0.30346698 0.14576599 0.30774294
  0.20571187 0.17697134 0.30308831 0.43233331 0.30341816 0.16253973
  0.55775221 0.41230944 0.41113968 0.59277599 0.37365916 0.35059811]]
SSE for k=3: 187.03025264441408
```

```python
# Append cluster assignments to the original DataFrame
df_with_clusters = df.copy()  # Create a copy to avoid modifying the
original DataFrame
df_with_clusters['Cluster'] = labels_custom

# View the updated DataFrame
print("Updated DataFrame with Cluster Assignments:")
print(df_with_clusters.head(20))
```

```
Updated DataFrame with Cluster Assignments:
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          M        17.99         10.38          122.80     1001.0
1          M        20.57         17.77          132.90     1326.0
2          M        19.69         21.25          130.00     1203.0
3          M        11.42         20.38           77.58      386.1
4          M        20.29         14.34          135.10     1297.0
5          M        12.45         15.70           82.57      477.1
6          M        18.25         19.98          119.60     1040.0
7          M        13.71         20.83           90.20      577.9
8          M        13.00         21.82           87.50      519.8
9          M        12.46         24.04           83.97      475.9
10         M        16.02         23.24          102.70      797.8
11         M        15.78         17.89          103.60      781.0
12         M        19.17         24.80          132.40     1123.0
13         M        15.85         23.95          103.70      782.7
14         M        13.73         22.61           93.60      578.3
15         M        14.54         27.54           96.73      658.8
16         M        14.68         20.13           94.74      684.5
17         M        16.13         20.68          108.10      798.8
18         M        19.81         22.15          130.00     1260.0
19         B        13.54         14.36           87.46      566.3

    smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0           0.11840           0.27760         0.30010
0.14710
1           0.08474           0.07864         0.08690
0.07017
```

```
2           0.10960            0.15990          0.19740
0.12790
3           0.14250            0.28390          0.24140
0.10520
4           0.10030            0.13280          0.19800
0.10430
5           0.12780            0.17000          0.15780
0.08089
6           0.09463            0.10900          0.11270
0.07400
7           0.11890            0.16450          0.09366
0.05985
8           0.12730            0.19320          0.18590
0.09353
9           0.11860            0.23960          0.22730
0.08543
10          0.08206            0.06669          0.03299
0.03323
11          0.09710            0.12920          0.09954
0.06606
12          0.09740            0.24580          0.20650
0.11180
13          0.08401            0.10020          0.09938
0.05364
14          0.11310            0.22930          0.21280
0.08025
15          0.11390            0.15950          0.16390
0.07364
16          0.09867            0.07200          0.07395
0.05259
17          0.11700            0.20220          0.17220
0.10280
18          0.09831            0.10270          0.14790
0.09498
19          0.09779            0.08129          0.06664
0.04781

    symmetry_mean  ...  texture_worst  perimeter_worst  area_worst  \
0          0.2419  ...          17.33           184.60      2019.0
1          0.1812  ...          23.41           158.80      1956.0
2          0.2069  ...          25.53           152.50      1709.0
3          0.2597  ...          26.50            98.87       567.7
4          0.1809  ...          16.67           152.20      1575.0
5          0.2087  ...          23.75           103.40       741.6
6          0.1794  ...          27.66           153.20      1606.0
7          0.2196  ...          28.14           110.60       897.0
8          0.2350  ...          30.73           106.20       739.3
9          0.2030  ...          40.68            97.65       711.4
10         0.1528  ...          33.88           123.80      1150.0
```

```
11         0.1842  ...          27.28           136.50      1299.0
12         0.2397  ...          29.94           151.70      1332.0
13         0.1847  ...          27.66           112.00       876.5
14         0.2069  ...          32.01           108.80       697.7
15         0.2303  ...          37.13           124.10       943.2
16         0.1586  ...          30.88           123.40      1138.0
17         0.2164  ...          31.48           136.80      1315.0
18         0.1582  ...          30.88           186.80      2398.0
19         0.1885  ...          19.26            99.70       711.2

    smoothness_worst  compactness_worst  concavity_worst  \
0             0.1622             0.6656           0.7119
1             0.1238             0.1866           0.2416
2             0.1444             0.4245           0.4504
3             0.2098             0.8663           0.6869
4             0.1374             0.2050           0.4000
5             0.1791             0.5249           0.5355
6             0.1442             0.2576           0.3784
7             0.1654             0.3682           0.2678
8             0.1703             0.5401           0.5390
9             0.1853             1.0580           1.1050
10            0.1181             0.1551           0.1459
11            0.1396             0.5609           0.3965
12            0.1037             0.3903           0.3639
13            0.1131             0.1924           0.2322
14            0.1651             0.7725           0.6943
15            0.1678             0.6577           0.7026
16            0.1464             0.1871           0.2914
17            0.1789             0.4233           0.4784
18            0.1512             0.3150           0.5372
19            0.1440             0.1773           0.2390

    concave points_worst  symmetry_worst  fractal_dimension_worst
Cluster
0                0.26540          0.4601                  0.11890
1
1                0.18600          0.2750                  0.08902
1
2                0.24300          0.3613                  0.08758
1
3                0.25750          0.6638                  0.17300
2
4                0.16250          0.2364                  0.07678
1
5                0.17410          0.3985                  0.12440
2
6                0.19320          0.3063                  0.08368
1
7                0.15560          0.3196                  0.11510
```

```
2
8                0.20600          0.4378                    0.10720
2
9                0.22100          0.4366                    0.20750
2
10               0.09975          0.2948                    0.08452
0
11               0.18100          0.3792                    0.10480
2
12               0.17670          0.3176                    0.10230
1
13               0.11190          0.2809                    0.06287
0
14               0.22080          0.3596                    0.14310
2
15               0.17120          0.4218                    0.13410
2
16               0.16090          0.3029                    0.08216
0
17               0.20730          0.3706                    0.11420
2
18               0.23880          0.2768                    0.07615
1
19               0.12880          0.2977                    0.07259
0

[20 rows x 32 columns]
```

PCA

PCA Function

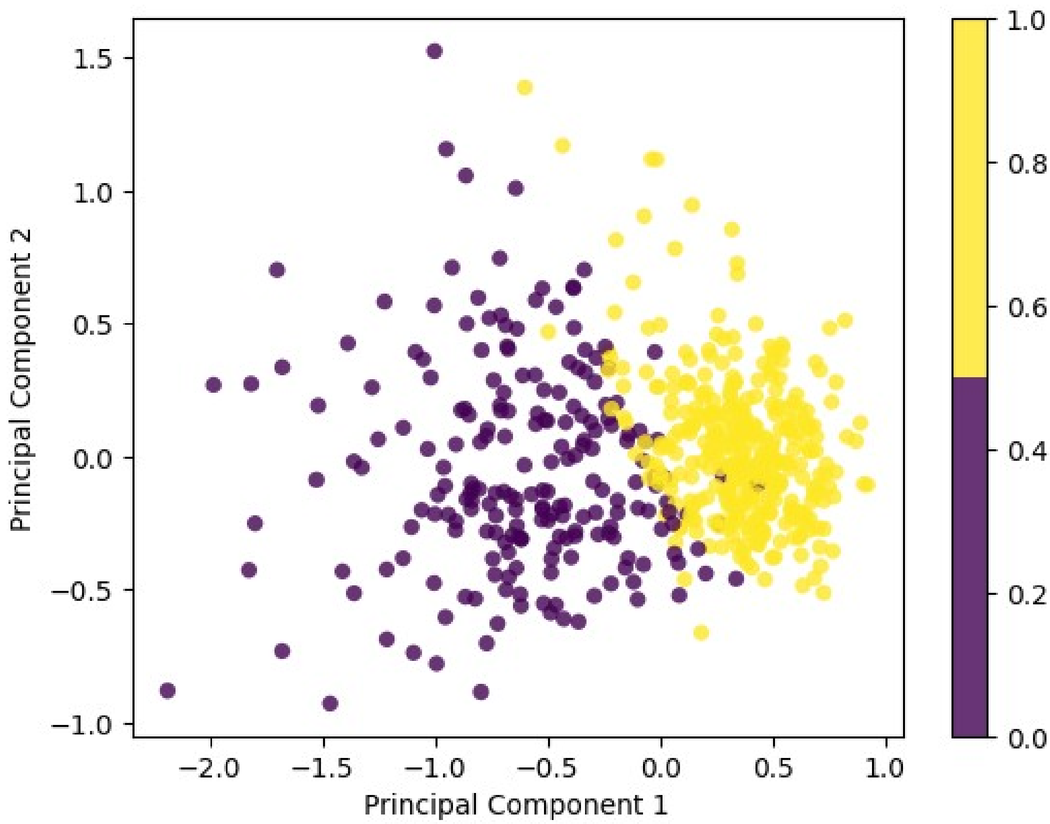
```
###PCA###
class PCA:

    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.componentsT = None
        self.mean = None
        self.eigenvalues = []
        self.slected_eigenvalues = []
        self.eigenvectors = None

    def fit(self, X):
      # mean centering
      self.mean = np.mean(X, axis=0)
      X = X -  self.mean
```

```
      # covariance, functions needs samples as columns
      cov = np.cov(X.T)

      # eigenvectors, eigenvalues
      self.eigenvalues, self.eigenvectors = np.linalg.eig(cov)

      # eigenvectors v = [:, i] column vector, transpose this for
easier calculations
      self.eigenvectors = self.eigenvectors.T

      # sort eigenvectors
      idxs = np.argsort(self.eigenvalues)[::-1]
      self.eigenvalues = self.eigenvalues[idxs]
      self.eigenvectors = self.eigenvectors[idxs]

      self.components = self.eigenvectors[:self.n_components]

    def transform(self, X):
        # projects data
        X = X - self.mean
        return np.dot(X,self.components.T)

    def explained_variance_ratio (self):
      return self.eigenvalues / np.sum(self.eigenvalues)

    def explained_variance_ratio_selectecd_pca (self):
      return self.slected_eigenvalues /
np.sum(self.slected_eigenvalues)

    def select_PCA(self,n):
      self.components = n
      self.slected_eigenvalues = self.eigenvalues[:self.n_components]
      self.components = self.eigenvectors[:self.n_components]

# Project the data onto the 2 primary principal components
pca = PCA(2)
pca.fit(df_scaled)
X_projected = pca.transform(df_scaled)

print("Shape of X:", df_scaled.shape)
print("Shape of transformed X:", X_projected.shape)

x1 = X_projected[:, 0]
x2 = X_projected[:, 1]

color_mapping = {'M': 0, 'B': 1}
colors = [color_mapping[label] for label in y]


plt.scatter(
    x1, x2, c=colors, edgecolor="none", alpha=0.8,
```

```
cmap=plt.cm.get_cmap("viridis", 2)
)

plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar()
plt.show()
```

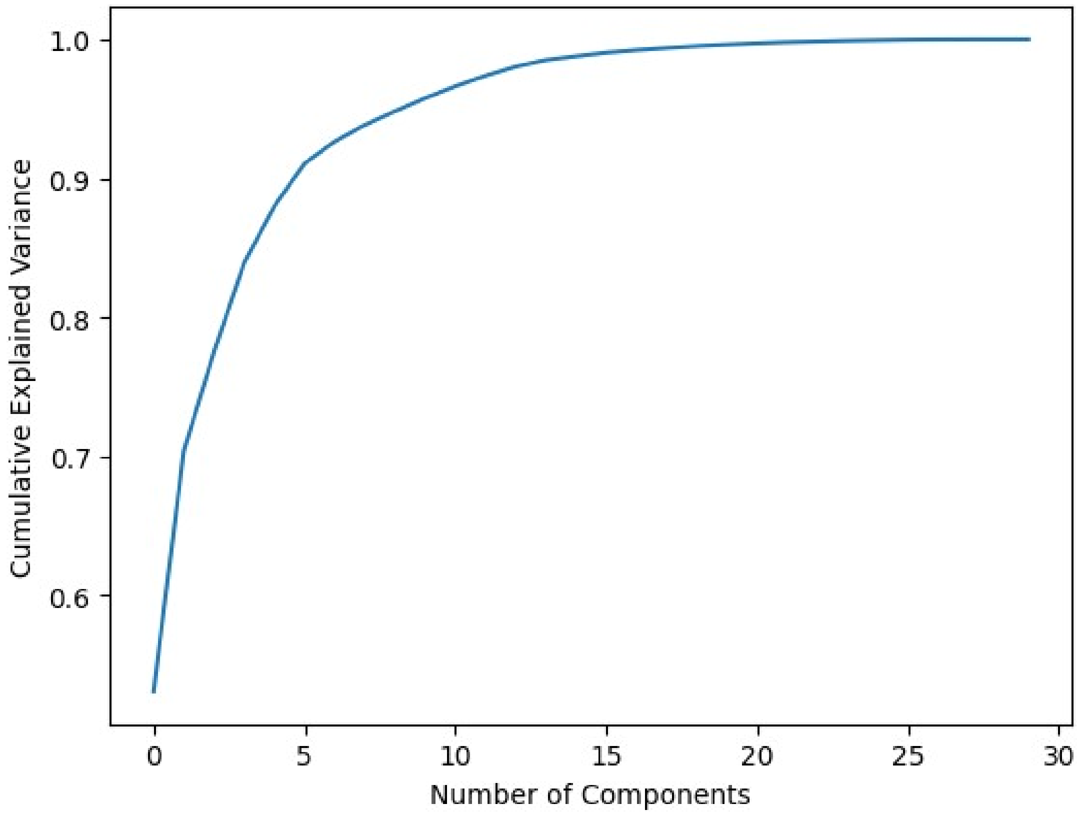
```
Shape of X: (569, 30)
Shape of transformed X: (569, 2)
```

```
<ipython-input-23-552d73299791>:17: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
two minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  x1, x2, c=colors, edgecolor="none", alpha=0.8,
cmap=plt.cm.get_cmap("viridis", 2)
```




```
explained_var_ratio = pca.explained_variance_ratio()
cumulative_var_ratio = np.cumsum(explained_var_ratio)

plt.plot(cumulative_var_ratio)
plt.xlabel('Number of Components')
```

```
plt.ylabel('Cumulative Explained Variance')
plt.show()
```




```
# Project the data onto the 2 primary principal components
pca2 = PCA(3)
pca2.fit(df_scaled)
X_projected2 = pca2.transform(df_scaled)

print("Shape of X:", df_scaled.shape)
print("Shape of transformed X:", X_projected2.shape)

x1_2 = X_projected2[:, 0]
x2_2 = X_projected2[:, 1]

color_mapping = {'M': 0, 'B': 1}
```

```
Shape of X: (569, 30)
Shape of transformed X: (569, 3)
```

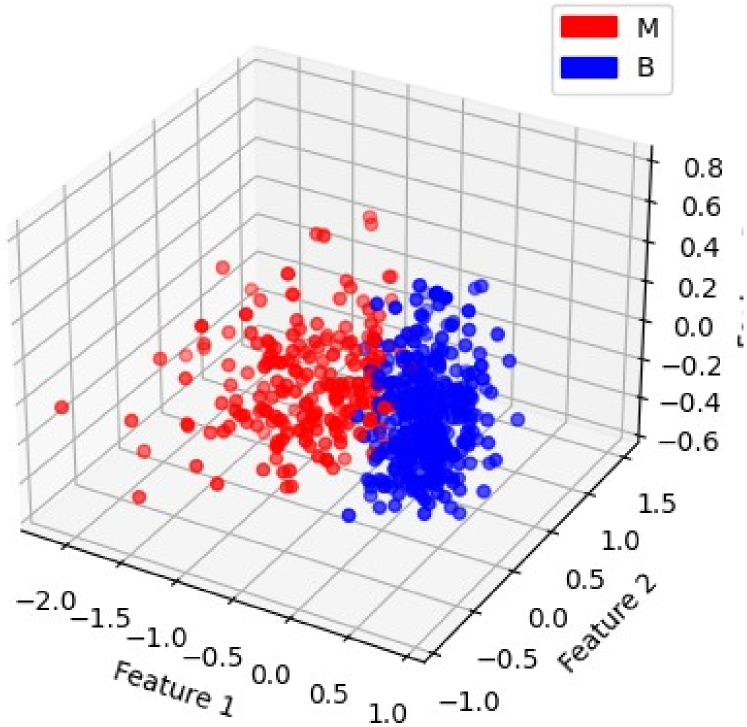
```
column_titles = ['Feature1', 'Feature2','Feature3']
df_projected = pd.DataFrame(X_projected2, columns=column_titles)
df_projected.head()
```

```
   Feature1  Feature2  Feature3
0 -1.387021  0.426895 -0.541703
1 -0.462308 -0.556947 -0.205175
2 -0.954621 -0.109701 -0.147848
3 -1.000816  1.525089 -0.053271
4 -0.626828 -0.302471 -0.409336
```

```
df_projected['target']=y
df_projected.head()
```

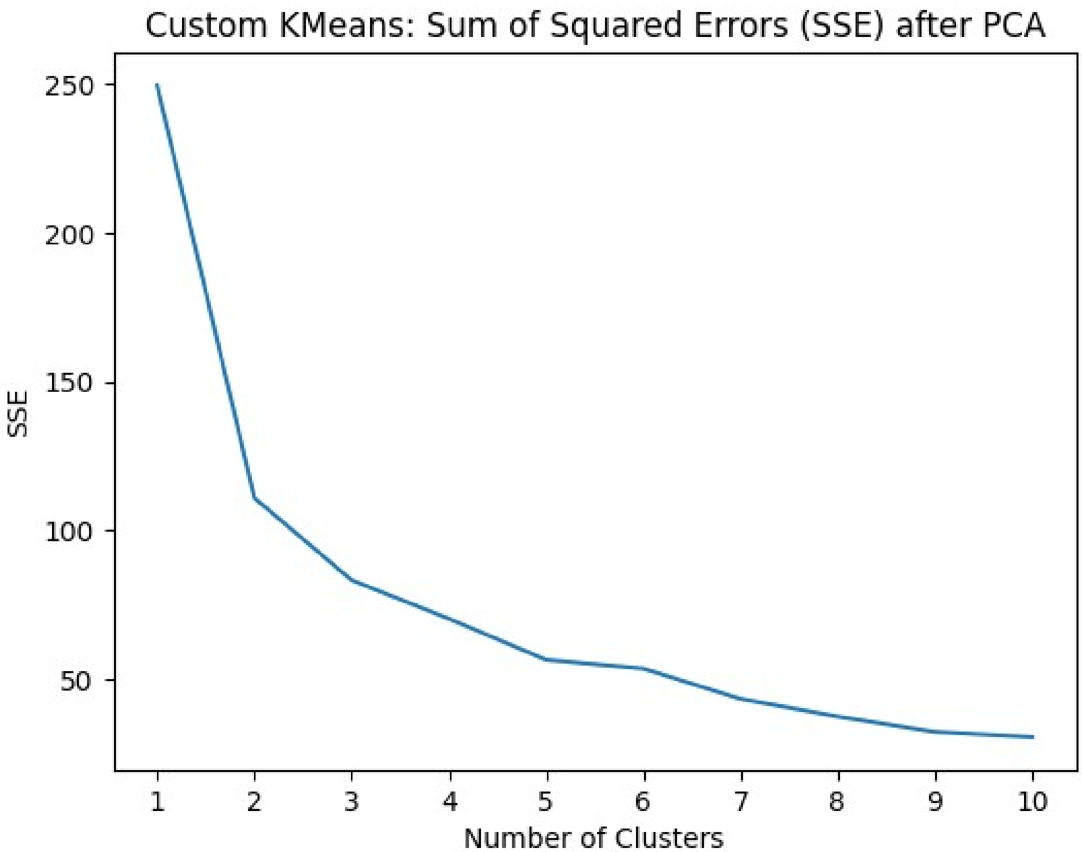
```
   Feature1  Feature2  Feature3 target
0 -1.387021  0.426895 -0.541703      M
1 -0.462308 -0.556947 -0.205175      M
2 -0.954621 -0.109701 -0.147848      M
3 -1.000816  1.525089 -0.053271      M
4 -0.626828 -0.302471 -0.409336      M
```

```
import matplotlib.pyplot as plt
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

# Assuming df_projected is your DataFrame containing the data

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Extracting features for plotting
x = df_projected['Feature1']
y = df_projected['Feature2']
z = df_projected['Feature3']

# Use Seaborn to define colors based on 'target'
colors = df_projected['target'].map({'M': 'red', 'B': 'blue'})

# Plotting the 3D scatter plot with Matplotlib
ax.scatter(x, y, z, c=colors, marker='o')

# Set labels for axes
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')

# Create a legend for the 'M' and 'B' categories
import matplotlib.patches as mpatches
M_patch = mpatches.Patch(color='red', label='M')
B_patch = mpatches.Patch(color='blue', label='B')
plt.legend(handles=[M_patch, B_patch])

plt.show()
```



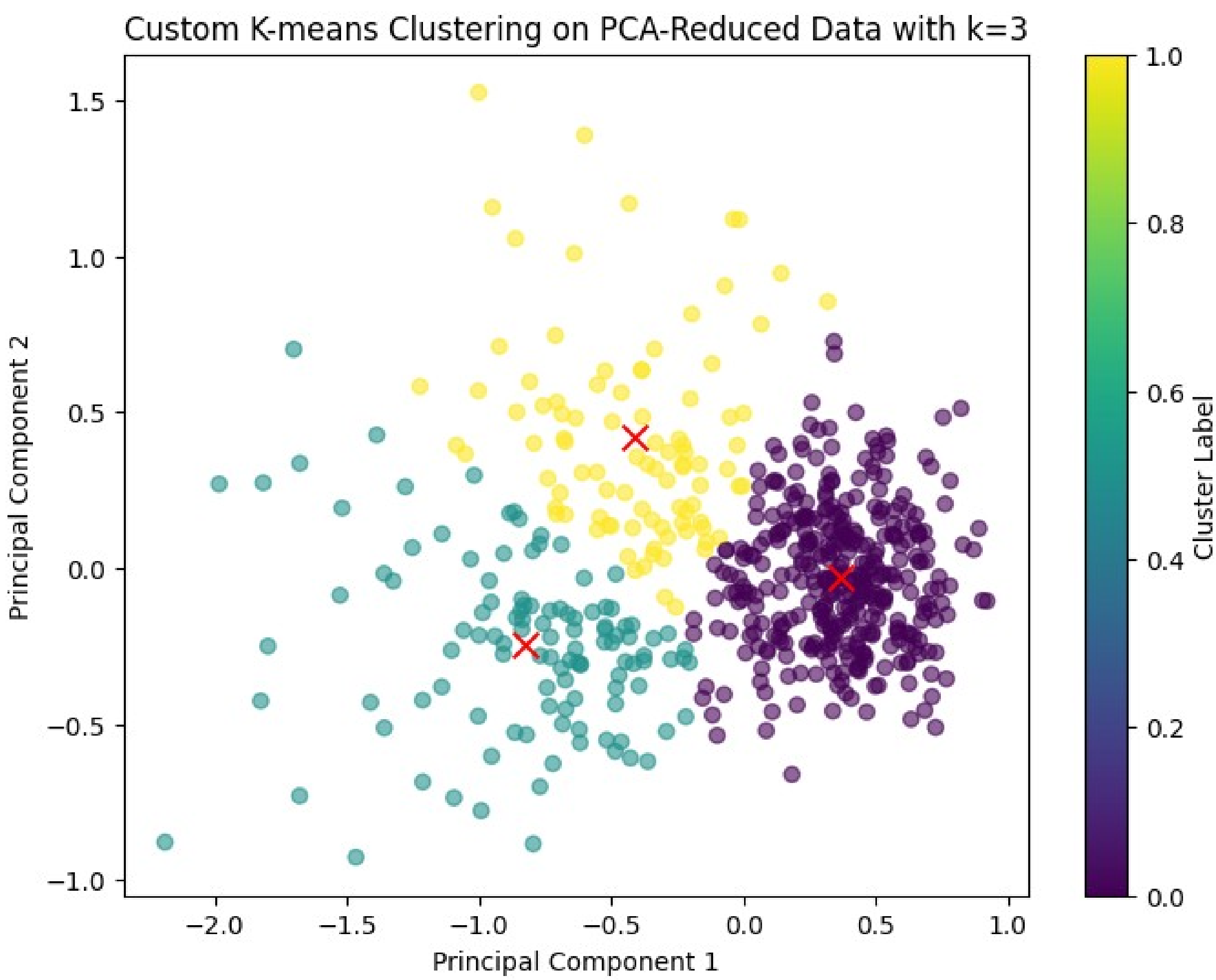
K-means w/ PCA

```
# Project the data onto the 2 primary principal components
pca_kmeans = PCA(2)
pca_kmeans.fit(df_scaled)
X_projected_kmeans = pca_kmeans.transform(df_scaled)

# Create a list to hold SSE values for each k
sse_custom_PCA = []

# Iterate over different values of k
for k in range(1, 11):
    labels, centroids, sse = custom_kmeans(X_projected_kmeans, k)
    sse_custom_PCA.append(sse)

# Visualize results
plt.plot(range(1, 11), sse_custom_PCA)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Custom KMeans: Sum of Squared Errors (SSE) after PCA")
plt.show()
```




```
temp = pd.DataFrame(sse_custom_PCA)
temp.head(10)

            0
0  249.456667
1  110.971509
2   83.371971
3   70.435200
4   56.645198
5   53.659692
6   43.533109
7   37.564503
8   32.438924
9   30.690678

#find the elbow point programmatically
k2 = KneeLocator(range(1, 11), sse_custom, curve="convex",
direction="decreasing")
k_optimal_PCA = k2.elbow
k_optimal_PCA

3
```

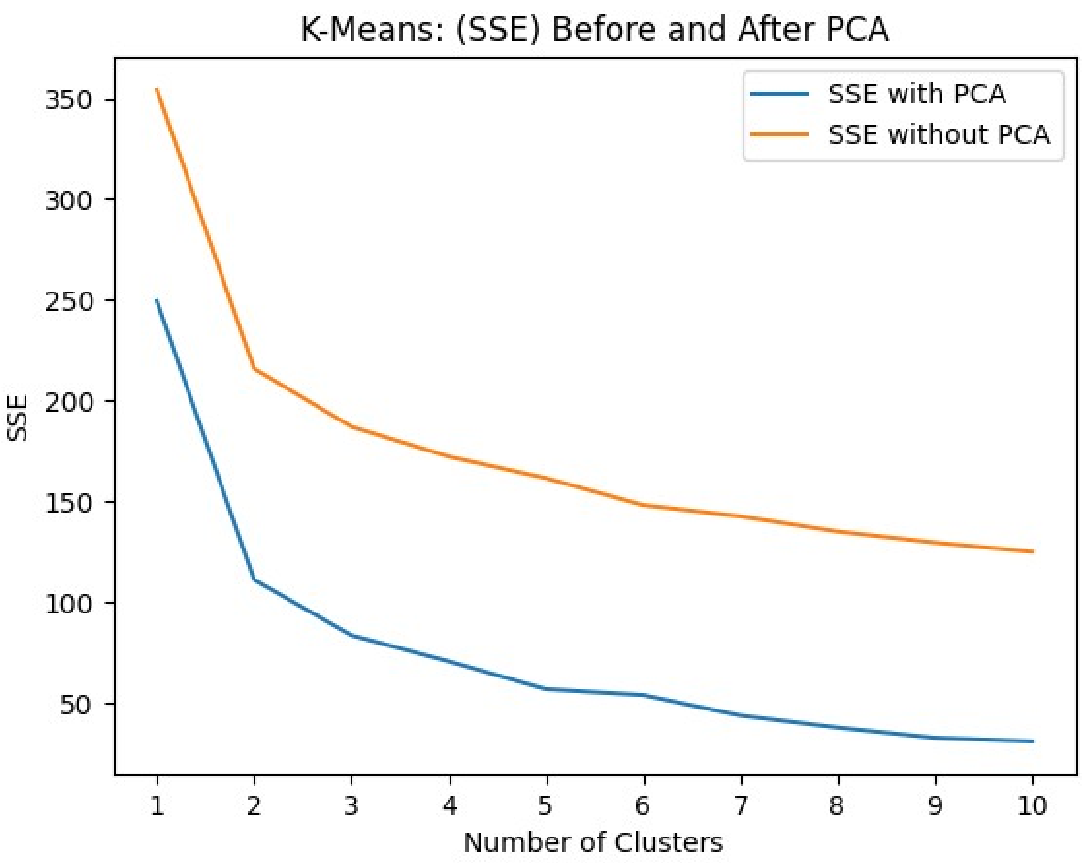
```python
# Apply custom_kmeans on the PCA-reduced data
k = k_optimal_PCA # Specify the number of clusters
labels_PCA, centroids_PCA, sse_PCA = custom_kmeans(X_projected_kmeans,
k)

# Visualization of the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X_projected_kmeans[:, 0], X_projected_kmeans[:, 1],
c=labels_PCA, cmap='viridis', alpha=0.6)
plt.scatter(centroids_PCA[:, 0], centroids_PCA[:, 1], c='red',
marker='x', s=100)  # Plotting centroids

plt.title(f'Custom K-means Clustering on PCA-Reduced Data with k={k}')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(label='Cluster Label')
plt.show()
print("SSE =", sse_PCA)
```



```
SSE = 83.37197137033976
```

#5.) Comparison

```
# Visualize results
plt.plot(range(1, 11), sse_custom_PCA, label='SSE with PCA')
plt.plot(range(1, 11), sse_custom, label='SSE without PCA')

plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("K-Means: (SSE) Before and After PCA")

# Display the legend
plt.legend()

plt.show()
```




```
sse_pca_df = pd.DataFrame(sse_custom_PCA, columns=['SSE (K-Means)'])
sse_df = pd.DataFrame(sse_custom, columns=['SSE (K-Means with PCA)'])
comparison_df = pd.concat([sse_df, sse_pca_df], axis=1)
comparison_df.head(10)
```

	SSE (K-Means with PCA)	SSE (K-Means)
0	354.436613	249.456667
1	215.838320	110.971509
2	187.030253	83.371971
3	172.172287	70.435200
4	161.445437	56.645198
5	148.049740	53.659692
6	142.450486	43.533109
7	134.925103	37.564503
8	129.431043	32.438924
9	124.852063	30.690678