Breast cancer

Assaingment

dec 6 2024

0.1 BreastCancerDetectionModel

0.1.1 Objective The objective is to evaluate and compare the performance of five supervised learning algorithms—
Logistic Regression, Decision Tree, Random Forest, SVM, and k-NN—on the breast cancer dataset from sklearn. This involves data preprocessing, model implementation, and performance analysis to identify the most effective algorithm for this classification problem.

0.1.2 DataDescription

Source: The dataset is sourced from the sklearn library

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

0.2 1. Loading and Preprocessing (2 marks)

Load the breast cancer dataset from sklearn.

Preprocess the data to handle any missing values and perform necessary feature scaling.

Explain the preprocessing steps you performed and justify why they are necessary for this dataset.

0.2.1 DataCollection:

[2]:
```
# Load the dataset
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
```

[3]:
```
# converting to dataframe
df = pd.DataFrame(data.data, columns = data.feature_names)
```

[5]:
```
df['target'] = data.target
```

[9]:
```
df.head()
```

[9]:

	meanradius	meantexture	meanperimeter	meanarea	meansmoothness	\
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	

	meancompactness	meanconcavity	meanconcavepoints	meansymmetry	\
0	0.27760	0.3001	0.14710	0.2419	
1	0.07864	0.0869	0.07017	0.1812	
2	0.15990	0.1974	0.12790	0.2069	
3	0.28390	0.2414	0.10520	0.2597	
4	0.13280	0.1980	0.10430	0.1809	

	mean fractal dimension	...	worst texture	worst perimeter	worst area	\
0	0.07871	...	17.33	184.60	2019.0	
1	0.05667	...	23.41	158.80	1956.0	
2	0.05999	...	25.53	152.50	1709.0	
3	0.09744	...	26.50	98.87	567.7	
4	0.05883	...	16.67	152.20	1575.0	

	worstsmoothness	worst compactness	worstconcavity	worstconcavepoints	\
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	

	worstsymmetry	worst fractaldimension	target
0	0.4601	0.11890	0
1	0.2750	0.08902	0
2	0.3613	0.08758	0
3	0.6638	0.17300	0
4	0.2364	0.07678	0

[5 rows x 31 columns]

[15]:
```
df.info()
```

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

```
3    meanarea                 569non-null   float64
4    meansmoothness           569non-null   float64
5    meancompactness          569non-null   float64
6    meanconcavity            569non-null   float64
7    meanconcavepoints        569non-null   float64
8    meansymmetry             569non-null   float64
9    meanfractal dimension    569 non-null  float64
10   radius error             569non-null   float64
11   textureerror             569non-null   float64
12   perimetererror           569non-null   float64
13   areaerror                569non-null   float64
14   smoothnesserror          569non-null   float64
15   compactnesserror         569non-null   float64
16   concavity error          569non-null   float64
17 concavepoints    error     569 non-null  float64
18   symmetryerror            569non-null   float64
19 fractal dimension error    569  non-null float64
2    worst radius             569non-null   float64
0    worst texture            569non-null   float64
21   worst perimeter          569non-null   float64
2    worst area               569non-null   float64
2    worst smoothness         569non-null   float64
2    worst compactness        569non-null   float64
3    worst concavity          569non-null   float64
227 worst concavepoints       569 non-null  float64
248  worst symmetry           569non-null   float64
29 worst fractal dimension    569 non-null  float64
30   target                   569non-null   int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

[17]:
```
df.isnull()
```

[17]:
```
     mean radius  mean texture  mean perimeter  meanarea  meansmoothness  \
0          False         False           False     False           False
1          False         False           False     False           False
2          False         False           False     False           False
3          False         False           False     False           False
4          False         False           False     False           False
..           ...           ...         ...  ...       ...    ...     ...
564        False         False           False     False           False
565        False         False           False     False           False
566        False         False           False     False           False
567        False         False           False     False           False
568        False         False

                                                          mean symmetry

     mean compactness mean concavity mean concave points              \
```

0 1 False False False False
2 3 False False False False
4 .. False False False False
56 False False False False
4 False False False False
56 ... ... False ... False
5 False False False False
56 False False False False
6 False False False False
56 False False False False
7 False False
56
8 mean fractal dimension ... worst texture worst perimeter worst area \
0 1 False ... False False False
2 3 False ... False False False
4 .. False ... False False False
56 False ... False False False
4 False ... False False False
56 ... ... False ... False
5 False ... False False False
56 False ... False False False
6 False ... False False False
56 False ... False False False
7 False False
56
8 worst smoothness worst compactness worst concavity \
0 1 False False False
2 3 False False False
4 .. False False False
56 False False False
4 False False False
56 ... ... False
5 False False False
56 False False False
6 False False False
56 False False False
7 False False
56
8 worst concave points worst symmetry worst fractal dimension target
0 1 False False False False
2 3 False False False False
4 .. False False False False
56 False False False False
4 False False False False
56 ... ... False
5 False False False False
False False False

```
566             False        False              False  False
567             False        False              False  False
568             False        False              False  False

[569 rows x 31 columns]
```

[19]:
```
df.isnull().sum()
```

[19]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

[21]:
```
df.duplicated().sum()
```

[21]: 0

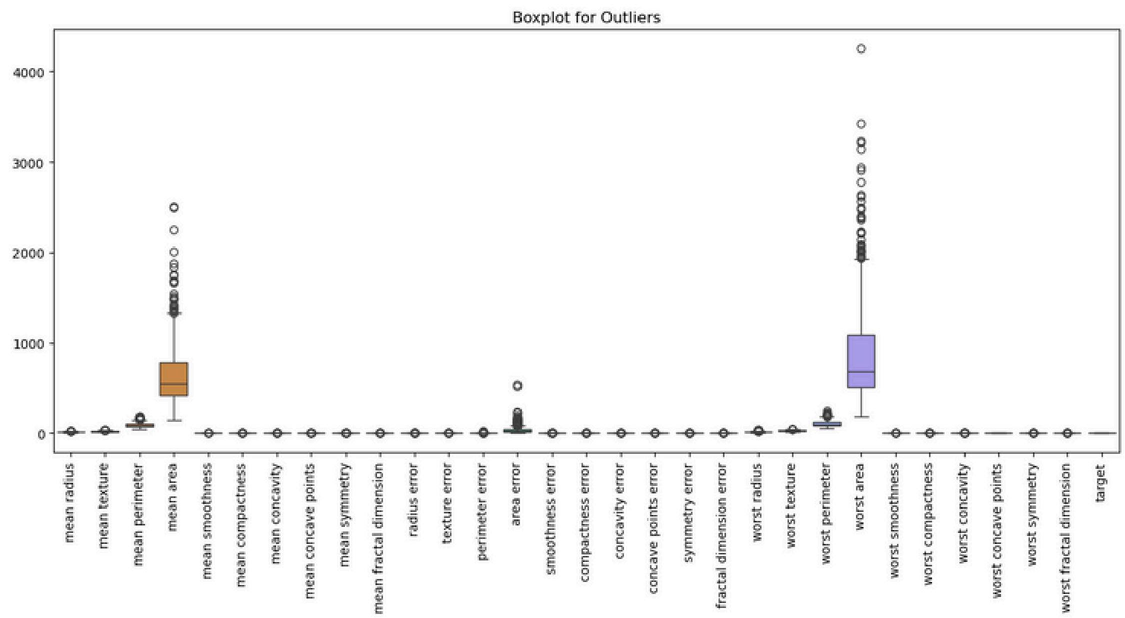
[23]:
```
df.shape
```

[23]: (569, 31)

0.2.2 CheckingforOutliers

Using Boxplots: Boxplots visually indicate outliers as points beyond the whiskers of the plot.

[25]:
```
plt.figure(figsize=(15, 6))
sns.boxplot(data=df)
plt.xticks(rotation=90)
plt.title("Boxplot for Outliers")
plt.show()
```



Using the IQR Method: The Interquartile Range (IQR) method identifies outliers based on statistical thresholds: Outliers are values: Below Q1−1.5×IQR Above Q3+1.5×IQR

[27]:
```
import pandas as pd

# Assuming `df` is your DataFrame containing all features (excluding target)
# Exclude the 'target' column if present
features_df = df.drop(columns=['target'], errors='ignore')
# Define a function to detect outliers based on IQR
def detect_outliers_iqr(dataframe):

    outliers = {}
    for column in dataframe.columns:
        Q1 = dataframe[column].quantile(0.25) #  First  quartile
        Q3 = dataframe[column].quantile(0.75) # Third quartile
        IQR = Q3 - Q1    # Interquartile range
        lower_bound = Q1 - 1.5 * IQR
```

```
        upper_bound = Q3 + 1.5 * IQR
        outliers[column] = dataframe[(dataframe[column] < lower_bound) |
(dataframe[column] > upper_bound)].index.tolist()

    return outliers

# Call the function and get outliers
outliers = detect_outliers_iqr(features_df)
print("Outliers detected:", outliers)
```

```
Outliers detected: {'mean radius': [82, 108, 122, 164, 180, 202, 212, 236, 339,
352, 369, 461, 503, 521], 'mean texture': [219, 232, 239, 259, 265, 455, 562],
'mean perimeter': [82, 108, 122, 164, 180, 202, 212, 236, 339, 352, 461, 503,
521], 'mean area': [23, 82, 108, 122, 164, 180, 202, 212, 236, 250, 265, 272,
339, 352, 368, 369, 372, 373, 393, 449, 461, 503, 521, 563, 564], 'mean
smoothness': [3, 105, 122, 504, 520, 568], 'mean compactness': [0, 3, 9, 12, 14,
78, 82, 108, 122, 181, 190, 258, 351, 352, 400, 567], 'mean concavity': [0, 68,
78, 82, 108, 112, 122, 152, 180, 202, 212, 258, 351, 352, 400, 461, 563, 567],
'mean concave points': [78, 82, 108, 122, 180, 202, 212, 352, 393, 461], 'mean
symmetry': [3, 22, 25, 60, 78, 108, 122, 146, 150, 152, 258, 288, 323, 424,
561], 'mean fractal dimension': [3, 9, 68, 71, 78, 151, 152, 176, 258, 318, 376,
379, 504, 505, 507], 'radius error': [0, 12, 25, 27, 38, 42, 77, 78, 82, 108,
122, 138, 161, 168, 210, 212, 218, 236, 250, 258, 265, 272, 290, 300, 302, 339,
352, 366, 368, 369, 417, 460, 461, 468, 503, 521, 563, 564], 'texture error':
[12, 83, 122, 136, 152, 192, 245, 258, 314, 345, 389, 416, 443, 471, 473, 528,
557, 559, 561, 565], 'perimeter error': [0, 12, 25, 38, 42, 77, 78, 82, 108,
122, 138, 161, 168, 210, 212, 218, 236, 250, 256, 258, 262, 265, 272, 300, 302,
335, 339, 352, 366, 368, 369, 417, 460, 461, 503, 521, 563, 564], 'area error':
[0, 2, 4, 12, 18, 23, 24, 25, 27, 30, 38, 42, 53, 56, 70, 77, 78, 82, 95, 108,
121, 122, 138, 156, 161, 162, 164, 168, 180, 210, 212, 218, 219, 236, 250, 252,
256, 258, 262, 265, 272, 300, 302, 335, 337, 339, 352, 366, 368, 369, 417, 433,
460, 461, 468, 492, 498, 503, 521, 533, 535, 563, 564, 565, 567], 'smoothness
error': [71, 76, 110, 111, 116, 122, 173, 176, 185, 196, 212, 213, 245, 273,
275, 288, 314, 332, 345, 391, 416, 424, 469, 505, 507, 520, 537, 538, 539, 556],
'compactness error': [3, 9, 12, 42, 62, 68, 71, 78, 108, 112, 116, 122, 152,
176, 190, 213, 288, 290, 318, 351, 376, 388, 430, 465, 468, 485, 539, 567],
'concavity error': [12, 42, 68, 78, 108, 112, 116, 122, 152, 176, 190, 202, 213,
242, 250, 290, 318, 351, 376, 388, 485, 539], 'concave points error': [12, 42,
68, 78, 138, 152, 161, 210, 213, 258, 288, 290, 366, 376, 389, 461, 485, 528,
563], 'symmetry error': [3, 12, 22, 42, 60, 63, 68, 78, 119, 122, 138, 146, 176,
190, 192, 212, 214, 290, 314, 329, 332, 343, 345, 351, 366, 520, 553], 'fractal
dimension error': [3, 9, 12, 14, 68, 71, 83, 112, 122, 145, 147, 151, 152, 176,
190, 213, 242, 257, 290, 376, 388, 450, 465, 468, 485, 504, 505, 507], 'worst
radius': [23, 82, 108, 164, 180, 212, 219, 236, 265, 272, 339, 352, 368, 369,
461, 503, 521], 'worst texture': [219, 239, 259, 265, 562], 'worst perimeter':
[23, 82, 108, 180, 212, 236, 265, 272, 339, 352, 368, 369, 461, 503, 521],
'worst area': [0, 1, 18, 23, 24, 56, 82, 108, 122, 162, 164, 180, 181, 202, 212,
218, 219, 236, 250, 254, 265, 272, 300, 323, 339, 352, 368, 369, 373, 393, 449,
```

461, 503, 521, 564], 'worst smoothness': [3, 41, 192, 203, 379, 504, 505], 'worst compactness': [0, 3, 9, 14, 15, 26, 33, 42, 72, 108, 181, 190, 379, 430, 562, 567], 'worst concavity': [9, 68, 108, 152, 190, 202, 252, 379, 400, 430, 562, 567], 'worst concave points': [], 'worst symmetry': [0, 3, 8, 9, 15, 22, 26, 31, 34, 35, 42, 68, 78, 119, 146, 190, 199, 203, 214, 323, 351, 370, 489], 'worst fractal dimension': [3, 5, 9, 14, 15, 26, 31, 34, 72, 105, 118, 151, 152, 181, 190, 229, 242, 252, 379, 465, 504, 505, 562, 567]}

[29]:
```python
for column in features_df.columns:
    Q1 = features_df[column].quantile(0.25)
    Q3 = features_df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    features_df[column] = features_df[column].clip(lower=lower_bound, 
↪upper=upper_bound)

df.head()
```

[29]:

	meanradius	mean texture	meanperimeter	meanarea	meansmoothness \
0	17.99	10.38	122.80	1001.0	0.11840
1	20.57	17.77	132.90	1326.0	0.08474
2	19.69	21.25	130.00	1203.0	0.10960
3	11.42	20.38	77.58	386.1	0.14250
4	20.29	14.34	135.10	1297.0	0.10030

	meancompactness	mean concavity	mean concavepoints	meansymmetry \
0	0.27760	0.3001	0.14710	0.2419
1	0.07864	0.0869	0.07017	0.1812
2	0.15990	0.1974	0.12790	0.2069
3	0.28390	0.2414	0.10520	0.2597
4	0.13280	0.1980	0.10430	0.1809

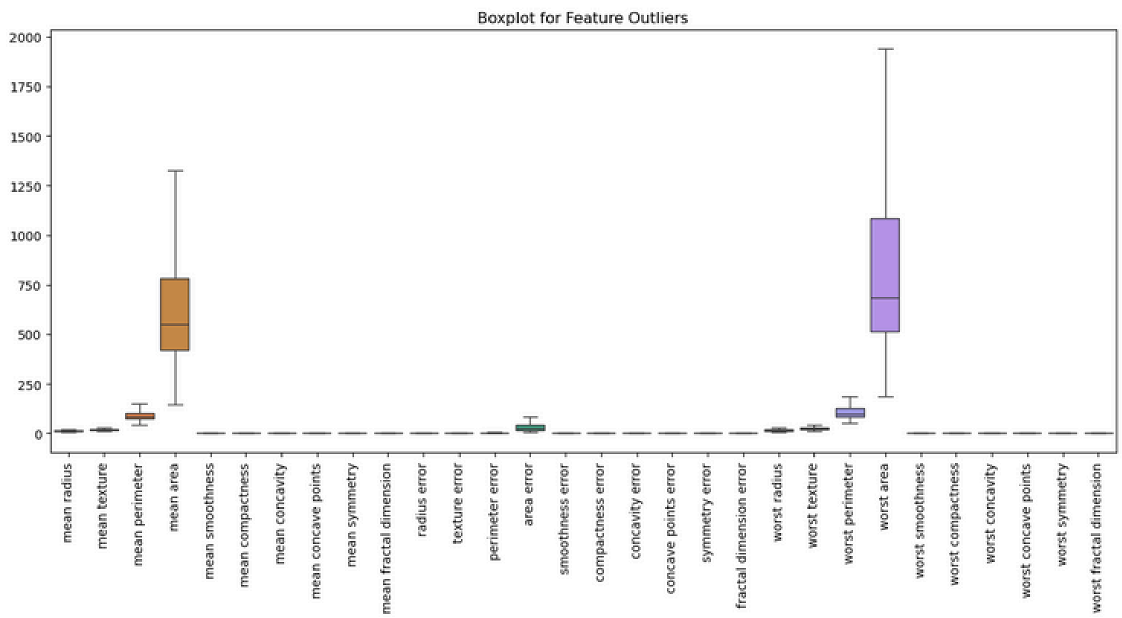
	mean fractal dimension	...	worst texture	worst perimeter	worst area \
0	0.07871	...	17.33	184.60	2019.0
1	0.05667	...	23.41	158.80	1956.0
2	0.05999	...	25.53	152.50	1709.0
3	0.09744	...	26.50	98.87	567.7
4	0.05883	...	16.67	152.20	1575.0

	worst smoothness	worst compactness	worst concavity	worst concave points \
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

worstsymmetry worst fractaldimension target

0	0.4601	0.11890	0
1	0.2750	0.0890	0
2	0.3613	2	0
3	0.6638	0.0875	0
4	0.2364	8	0
		0.1730	
[5 rows x 31 columns]		0	
		0.0767	
		8	

[31]:
```
# Plot boxplots for all features
plt.figure(figsize=(15, 6))
sns.boxplot(data=features_df)
plt.xticks(rotation=90)
plt.title("Boxplot for Feature Outliers")
plt.show()
```



[33]:
```
# checking skewness
df.skew()
```

[33]: mean radius 0.94238
mean texture 0
mean perimeter 0.65045
mean area 0
mean smoothness 0.99065
mean compactness 0
mean concavity 1.645732
mean concave points 0.45632
mean symmetry 4
1.190123
1.401180
1.171180 9
0.72560
9

```
mean fractal dimension     1.304489
radius error               3.088612
texture error              1.646444
perimeter error            3.443615
area error                 5.447186
smoothness error           2.314450
compactness error          1.902221
concavity error            5.110463
concave points error       1.444678
symmetry error             2.195133
fractal dimension error    3.923969
worst radius               1.103115
worst texture              0.498321
worst perimeter            1.128164
worst area                 1.859373
worst smoothness           0.415426
worst compactness          1.473555
worst concavity            1.150237
worst concave points       0.492616
worst symmetry             1.433928
worst fractal dimension    1.662579
target                    -0.528461
dtype: float64
```

[35]:
```
df_final = df.copy()
df_final.head()
```

[35]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness \
0	17.99	10.38	122.80	1001.0	0.11840
1	20.57	17.77	132.90	1326.0	0.08474
2	19.69	21.25	130.00	1203.0	0.10960
3	11.42	20.38	77.58	386.1	0.14250
4	20.29	14.34	135.10	1297.0	0.10030

	mean compactness	mean concavity	mean concave points	mean symmetry \
0	0.27760	0.3001	0.14710	0.2419
1	0.07864	0.0869	0.07017	0.1812
2	0.15990	0.1974	0.12790	0.2069
3	0.28390	0.2414	0.10520	0.2597
4	0.13280	0.1980	0.10430	0.1809

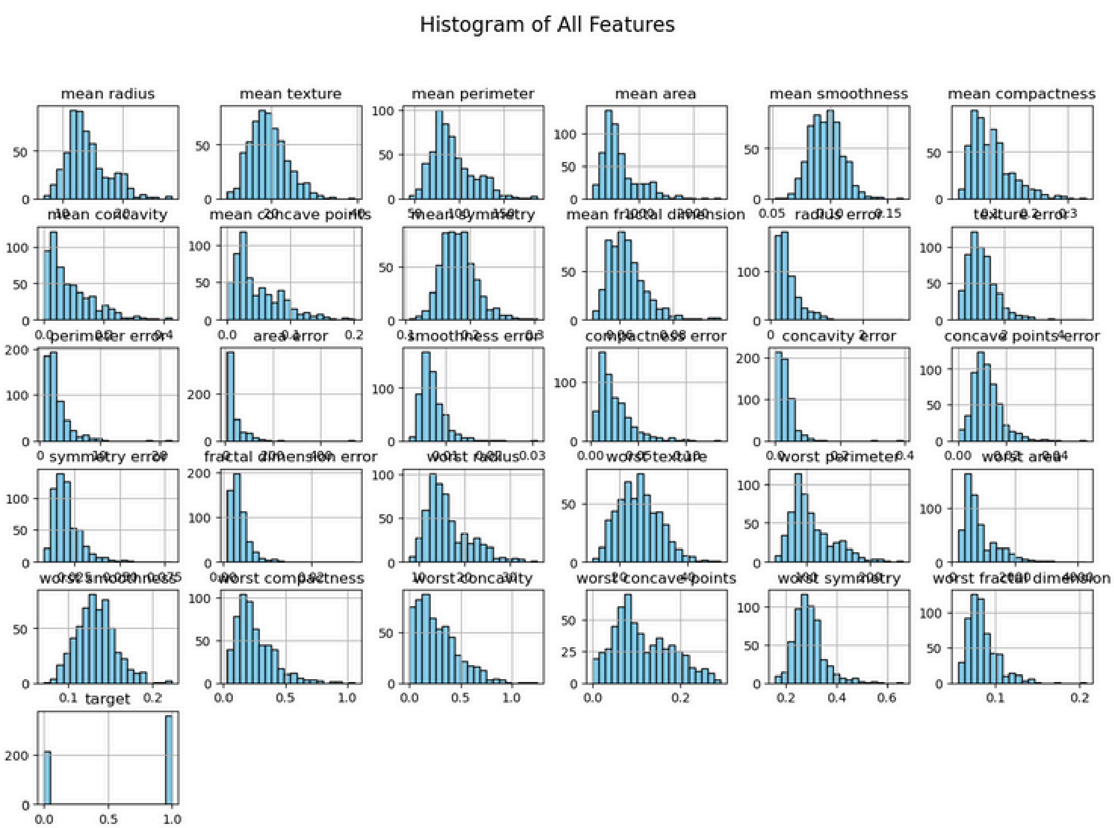
	mean fractal dimension	...	worst texture	worst perimeter	worst area \
0	0.07871	...	17.33	184.60	2019.0
1	0.05667	...	23.41	158.80	1956.0
2	0.05999	...	25.53	152.50	1709.0
3	0.09744	...	26.50	98.87	567.7
4	0.05883	...	16.67	152.20	1575.0

	worst smoothness	worst compactness	worst concavity	worst concave points \
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

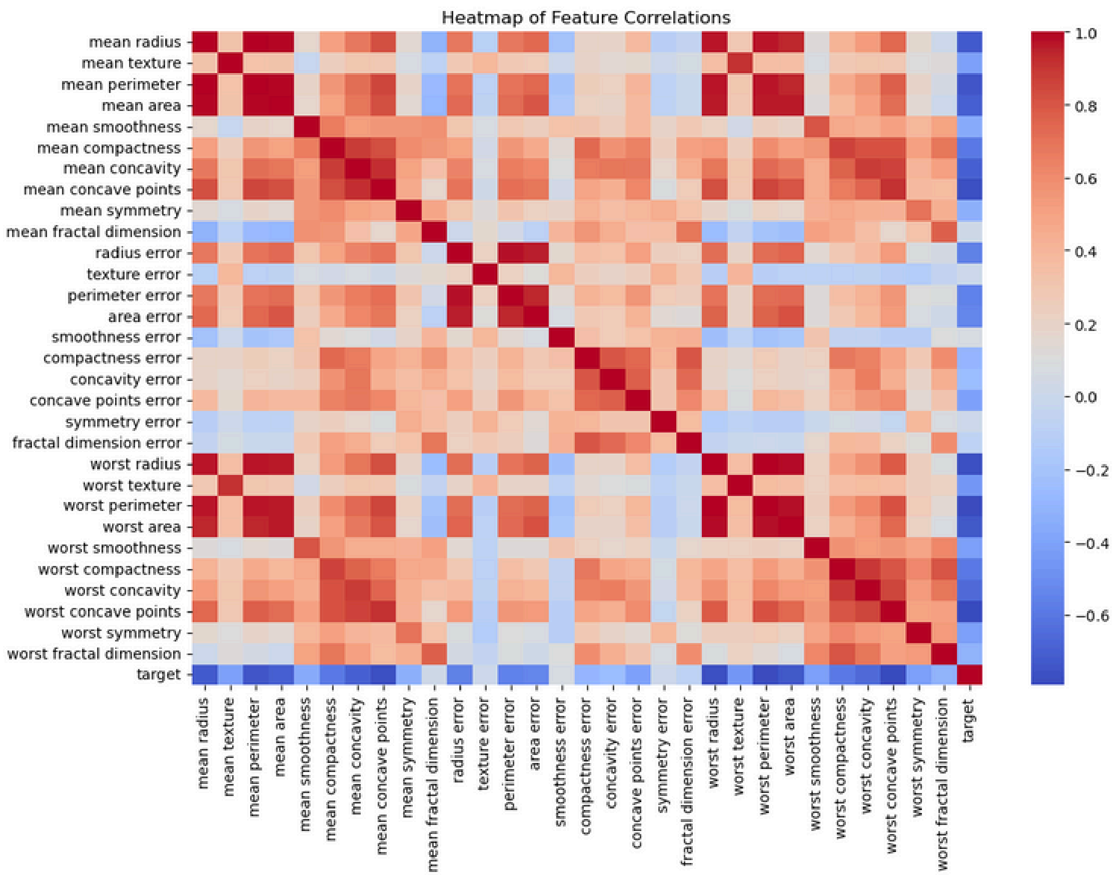
	worst symmetry	worst fractal dimension	target
0	0.4601	0.11890	0
1	0.2750	0.08902	0
2	0.3613	0.08758	0
3	0.6638	0.17300	0
4	0.2364	0.07678	0

[5 rows x 31 columns]

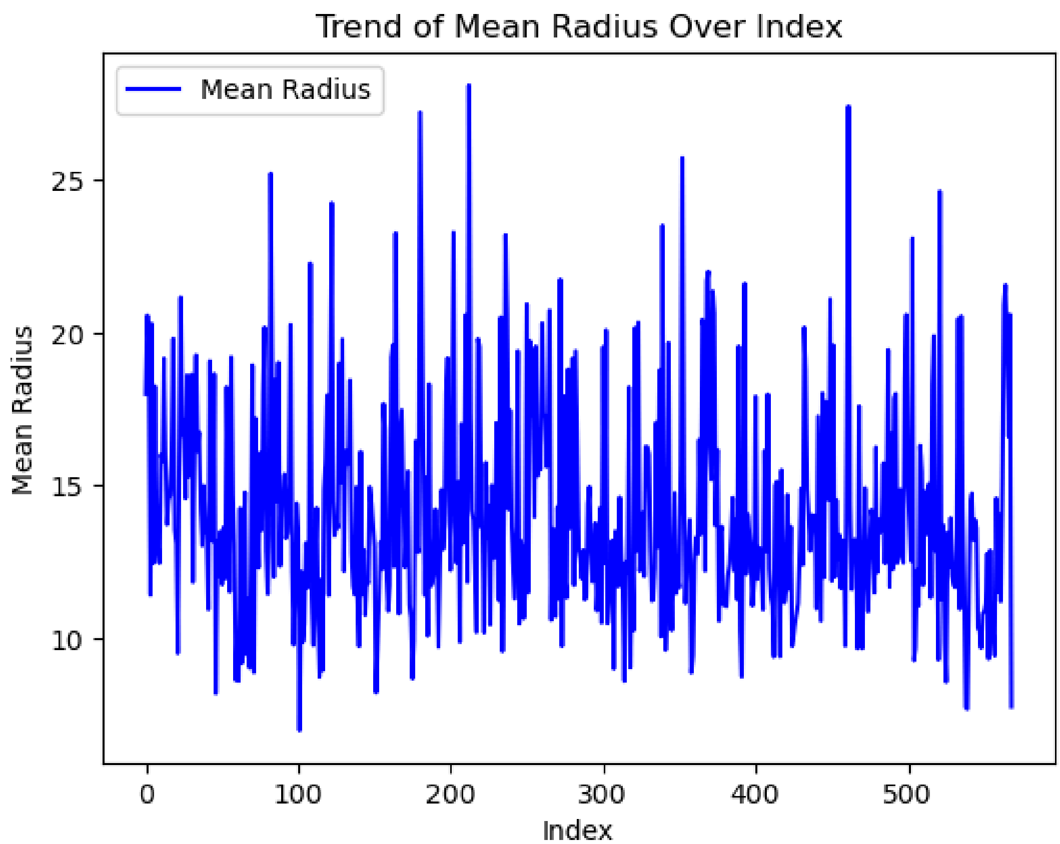
[37]:
```
#Histogram

# Plot histograms for all features
df.hist(figsize=(15, 10), bins=20, color='skyblue', edgecolor='black')
plt.suptitle("Histogram of All Features", fontsize=16)
plt.show()
```



[38]:
```
# Compute correlation matrix
corr_matrix = df.corr()
# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm', cbar=True)
plt.title("Heatmap of Feature Correlations")
plt.show()
```



[39]:
```
# Example line plot for feature trends
plt.plot(df.index, df['mean radius'], label='Mean Radius', color='blue')
plt.title("Trend of Mean Radius Over Index")
plt.xlabel("Index")
plt.ylabel("Mean Radius")
plt.legend()
plt.show()
```



0.3 Featureselection

[43]:
```
# Using correlation matrix

# Compute correlation matrix
corr_matrix = df_final.corr().abs()
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
# Find features with correlation > 0.9
to_drop = [column for column in upper.columns if any(upper[column] > 0.9)]
print(f"Features to drop due to high correlation: {to_drop}")

# Drop features
df_reduced = df_final.drop(columns=to_drop)
```

Features to drop due to high correlation: ['mean perimeter', 'mean area', 'mean concave points', 'perimeter error', 'area error', 'worst radius', 'worst texture', 'worst perimeter', 'worst area', 'worst concave points']

[45]:
```python
df_reduced.head()
```

[45]:
```
   mean radius  mean texture  mean smoothness  mean compactness  \
0        17.99         10.38          0.11840           0.27760
1        20.57         17.77          0.08474           0.07864
2        19.69         21.25          0.10960           0.15990
3        11.42         20.38          0.14250           0.28390
4        20.29         14.34          0.10030           0.13280

   mean concavity  mean symmetry  mean fractal dimension  radius error  \
0          0.3001         0.2419                 0.07871        1.0950
1          0.0869         0.1812                 0.05667        0.5435
2          0.1974         0.2069                 0.05999        0.7456
3          0.2414         0.2597                 0.09744        0.4956
4          0.1980         0.1809                 0.05883        0.7572

   texture error  smoothness error  ...  concavity error  \
0         0.9053          0.006399  ...          0.05373
1         0.7339          0.005225  ...          0.01860
2         0.7869          0.006150  ...          0.03832
3         1.1560          0.009110  ...          0.05661
4         0.7813          0.011490  ...          0.05688

   concave points error  symmetry error  fractal dimension error  \
0               0.01587         0.03003                 0.006193
1               0.01340         0.01389                 0.003532
2               0.02058         0.02250                 0.004571
3               0.01867         0.05963                 0.009208
4               0.01885         0.01756                 0.005115

   worst smoothness  worst compactness  worst concavity  worst symmetry  \
0            0.1622             0.6656           0.7119          0.4601
1            0.1238             0.1866           0.2416          0.2750
2            0.1444             0.4245           0.4504          0.3613
3            0.2098             0.8663           0.6869          0.6638
4            0.1374             0.2050           0.4000          0.2364

   worst fractal dimension  target
0                  0.11890       0
1                  0.08902       0
2                  0.08758       0
3                  0.17300       0
4                  0.07678       0

[5 rows x 21 columns]
```

0.4 XandY

[47]:
```python
x = df_reduced.drop('target', axis=1)
x
```

[47]:
```
     mean radius  mean texture  mean smoothness  mean compactness  \
0          17.99         10.38          0.11840           0.27760
1          20.57         17.77          0.08474           0.07864
2          19.69         21.25          0.10960           0.15990
3          11.42         20.38          0.14250           0.28390
4          20.29         14.34          0.10030           0.13280
..           ...           ...              ...               ...
564        21.56         22.39          0.11100           0.11590
565        20.13         28.25          0.09780           0.10340
566        16.60         28.08          0.08455           0.10230
567        20.60         29.33          0.11780           0.27700
568         7.76         24.54          0.05263           0.04362

     mean concavity  mean symmetry  mean fractal dimension  radius error  \
0           0.30010         0.2419                 0.07871        1.0950
1           0.08690         0.1812                 0.05667        0.5435
2           0.19740         0.2069                 0.05999        0.7456
3           0.24140         0.2597                 0.09744        0.4956
4           0.19800         0.1809                 0.05883        0.7572
..              ...            ...                     ...           ...
564         0.24390         0.1726                 0.05623        1.1760
565         0.14400         0.1752                 0.05533        0.7655
566         0.09251         0.1590                 0.05648        0.4564
567         0.35140         0.2397                 0.07016        0.7260
568         0.00000         0.1587                 0.05884        0.3857

     textureerror  smoothness error  compactnesserror  concavityerror  \
0          0.9053          0.006399           0.04904         0.05373
1          0.7339          0.005225           0.01308         0.01860
2          0.7869          0.006150           0.04006         0.03832
3          1.1560          0.009110           0.07458         0.05661
4          0.7813          0.011490           0.02461         0.05688
..            ...               ...               ...             ...
564        1.2560          0.010300           0.02891         0.05198
565        2.4630          0.005769           0.02423         0.03950
566        1.0750          0.005903           0.03731         0.04730
567        1.5950          0.006522           0.06158         0.07117
568        1.4280          0.007189           0.00466         0.00000

     concave points error  symmetry error  fractal dimension error
0                 0.01587         0.03003                 0.006193
1                 0.01340         0.01389                 0.003532
```

```
2   3          0.02058        0.02250              0.00457
4   ..         0.01867        0.05963              1
564            0.01885        0.01756              0.00920
565               ...            ...               8...
566            0.02454        0.01114              0.00425
567            0.01678        0.01898              9
568            0.01557        0.01318              0.00249
               0.01664        0.02324              8
               0.0000         0.02676              0.00389
               0                                   2
     worst smoothness worst compactness worst concavity 0.00618 worst symmetry \
0          0.16220          0.66560          0.7119 5         0.4601
1          0.12380          0.18660          0.2416 0.00278   0.2750
2          0.14440          0.42450          0.4504 3         0.3613
3          0.20980          0.86630          0.6869           0.6638
4          0.13740          0.20500          0.4000           0.2364
..           ...              ...              ...      ...   0.2060
564        0.14100          0.21130          0.4107           0.2572
565        0.11660          0.19220          0.3215           0.2218
566        0.11390          0.30940          0.3403           0.4087
567        0.16500          0.86810          0.9387           0.2871
568        0.08996          0.06444          0.0000

     worst fractal dimension
0                  0.11890
1                  0.08902
2                  0.08758
3                  0.17300
4                  0.07678
..                   ...
564                0.07115
565                0.06637
566                0.07820
567                0.12400
568                0.07039

[569 rows x 20 columns]
```

[49]:
```
y = df_reduced['target']
y
```

[49]:
```
0
1      0
2     0
3     0
4     0
      0
       ..
```

5640 5650 5660 5670 5681 Name: target,
Length: 569, dtype: int32

0.4.1 Scaling

[51]:
```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler
standard_scaler = StandardScaler()
minmax_scaler = MinMaxScaler()
```

[53]:
```python
#Applying scaling
x_normalized = minmax_scaler.fit_transform(x)
# converting into dataframe
x_normalized = pd.DataFrame(x_normalized)
x_normalized.head()
```

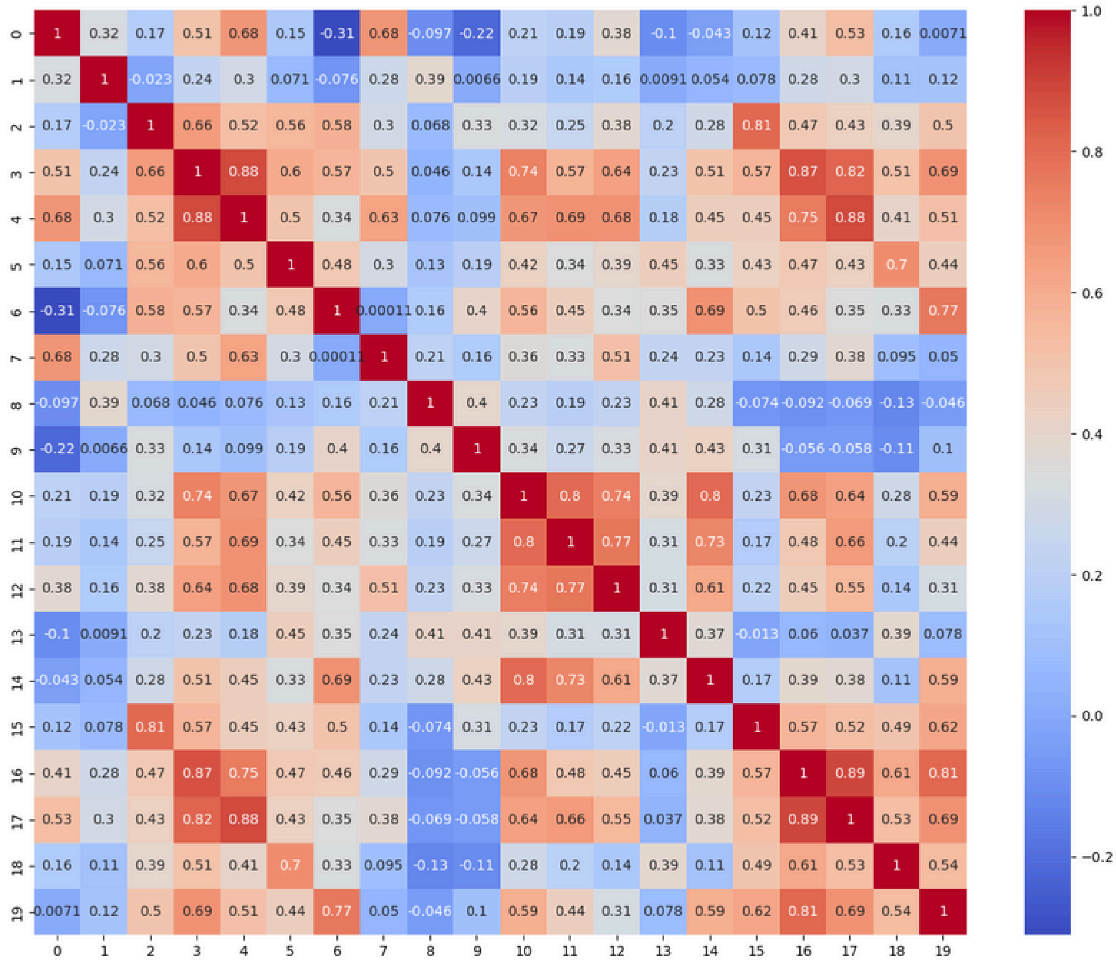
[53]:

	0	1	2	3	4	5	6 \
0	0.521037	0.022658	0.593753	0.792037	0.703140	0.686364	0.605518
1	0.643144	0.272574	0.289880	0.181768	0.203608	0.379798	0.141323
2	0.601496	0.390260	0.514309	0.431017	0.462512	0.509596	0.211247
3	0.210090	0.360839	0.811321	0.811361	0.565604	0.776263	1.000000
4	0.629893	0.156578	0.430351	0.347893	0.463918	0.378283	0.186816

	7	8	9	10	11	12	13 \
0	0.356147	0.120469	0.159296	0.351398	0.135682	0.300625	0.311645
1	0.156437	0.082589	0.119387	0.081323	0.046970	0.253836	0.084539
2	0.229622	0.094303	0.150831	0.283955	0.096768	0.389847	0.205690
3	0.139091	0.175875	0.251453	0.543215	0.142955	0.353665	0.728148
4	0.233822	0.093065	0.332359	0.167918	0.143636	0.357075	0.136179

	14	15	16	17	18	19
0	0.183042	0.601136	0.619292	0.568610	0.598462	0.418864
1	0.091110	0.347553	0.154563	0.192971	0.233590	0.222878
2	0.127006	0.483590	0.385375	0.359744	0.403706	0.213433
3	0.287205	0.915472	0.814012	0.548642	1.000000	0.773711
4	0.145800	0.437364	0.172415	0.319489	0.157500	0.142595

[55]:
```python
correlation = x_normalized.corr()
plt.figure(figsize=(15, 12))
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.show()
```



0.5 Split the dataset into training and testing sets

[57]:
```python
from sklearn.model_selection import train_test_split
```

[59]:
```python
x_train, x_test, y_train, y_test = train_test_split(x_normalized, y,␣
↪test_size=0.2, random_state=42)
```

0.6 ClassificationAlgorithmImplementation

Classification Algorithms: Brief Descriptions and Suitability #### 1. Logistic Regression How it works: Logistic Regression is a linear model that uses the logistic (sigmoid) function to predict probabilities for binary classification. The decision boundary is linear, separating the classes by maximizing the likelihood of the observed data.

Why suitable for this dataset: The dataset is well-structured, and logistic regression performs well for linearly separable data. It's a good baseline due to its simplicity and interpretability. #### 2. Decision Tree Classifier How it works: Decision Trees split the dataset recursively based on feature values that maximize information gain (or minimize impurity, like Gini Index). The process continues until leaf nodes classify the data. why suitable for this dataset: Decision trees handle both numerical and categorical data well and can model complex interactions between features. They are interpretable and work effectively with the dataset's mix of features. #### 3. Random Forest Classifier How it works: Random Forest combines multiple decision trees (a forest) where each tree is trained on a random subset of the data and features. The final prediction is the majority vote (classification) or average (regression) of individual trees.

Why suitable for this dataset: It's robust to overfitting due to its ensemble nature and provides high accuracy on structured datasets like this. It can handle feature importance well, which is crucial for understanding significant predictors of breast cancer. #### 4. Support Vector Machine (SVM) How it works: SVM finds the hyperplane that maximizes the margin between two classes. For non-linearly separable data, it uses kernels (e.g., radial basis function or polynomial) to map features into higher dimensions for separation.

Why suitable for this dataset: SVM is effective for small-to-medium-sized datasets with a clear margin of separation. The breast cancer dataset has a limited number of samples and may benefit from kernel methods for separating classes. #### 5. k-Nearest Neighbors (k-NN) How it works: k-NN classifies a sample based on the majority class of its k closest neighbors (in terms of a distance metric like Euclidean). It's a non-parametric and instance-based learning algorithm.

Why suitable for this dataset: k-NN works well on datasets where class distributions are distinct in feature space. While computationally expensive for large datasets, it performs well on this dataset's manageable size and dimensionality.

Train the Models Train each of the five algorithms on the training data.

[61]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
# Initialize models
models = {

    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42), "Random Forest":
    RandomForestClassifier(random_state=42),   "SVM":   SVC(),   "k-NN":
    KNeighborsClassifier(),

}
# Train models
for name, model in models.items():

    model.fit(x_train, y_train)
```

Evaluate Model Performance Evaluate each model's performance using accuracy, precision, recall, F1-score, and confusion matrix.

[63]:
```python
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

results = []

for name, model in models.items():
    # Predict on test data
    y_pred = model.predict(x_test)

    # Evaluate performance
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    f1_score = report["weighted avg"]["f1-score"]
    # Store results
    results.append({"Model": name, "Accuracy": accuracy, "F1-Score": f1_score})
    # Print detailed classification report
    print(f"Model: {name}")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("-" * 50)



# Convert results to a DataFrame
import pandas as pd
results_df = pd.DataFrame(results)
```

```
Model: Logistic Regression
              precision    recall  f1-score   support

           0       0.97      0.88      0.93        43
           1       0.93      0.99      0.96        71

    accuracy                           0.95       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.95      0.95      0.95       114

Confusion Matrix:
 [[38  5]
 [ 1 70]]
--------------------------------------------------
Model: Decision Tree
              precision    recall  f1-score   support

           0       0.87      0.91      0.89        43
           1       0.94      0.92      0.93        71
```

```
         accuracy                          0.91        114
        macroavg   0.90          0.91      0.91        114
     weightedavg   0.91          0.91      0.91        114
Confusion Matrix:

 [[39  4]  [
 6 65]]
-------------------------------------------------
Model: Random Forest
                  precision      recall  f1-score     support
                       0.95        0.95  0.95              43
                0    0.97        0.97  0.97              71
                1                      0.96             114
         accuracy                      0.96             114
        macroavg                       0.96             114
     weightedavg   0.96          0.96
Confusion Matrix:  0.96          0.96


 [[41  2]  [
 2 69]]
-------------------------------------------------
Model: SVM
                  precision      recall  f1-score     support
                       0.93        0.91  0.92              43
                0    0.94        0.96  0.95              71
                1                      0.94             114
         accuracy                      0.93             114
        macroavg                       0.94             114
     weightedavg   0.94          0.93
Confusion Matrix:  0.94          0.94


 [[39  4]  [
 3 68]]
-------------------------------------------------
Model: k-NN
                  precision      recall  f1-score     support
                       0.93        0.93  0.93              43
                0    0.96        0.96  0.96              71
                1                      0.95             114
         accuracy                      0.94             114
        macroavg                       0.95             114
     weightedavg   0.94          0.94
Confusion Matrix:  0.95          0.95


 [[40 3]
```

```
 [ 3 68]]
--------------------------------------------------
```

Compare Models Compare the performance metrics in a summary table.

```
[65]: # Sort models by accuracy or F1-score
results_df = results_df.sort_values(by="Accuracy", ascending= False)
print(results_df)
```

```
                 Model  Accuracy  F1-Score
2        Random Forest  0.964912  0.964912
0  Logistic Regression  0.947368  0.946806
4                 k-NN  0.947368  0.947368
3                  SVM  0.938596  0.938450
1        Decision Tree  0.912281  0.912651
```

Identify Best and Worst Models Best Model: Random Forest with highest accuracy 0.97 and F1-score 0.97. Worst Model: Decision Tree with lowest accuracy 0.91 and F1-score 0.91.

0.7 HyperparameterTuningwithRandomizedSearchCV

Hyperparameter Tuning for the Best Model After comparing the performance of various models (Logistic Regression, Decision Tree, Random Forest, SVM, k-NN), assume the best-performing model is selected (e.g., Random Forest Classifier). The following demonstrates how to optimize the selected model's performance through hyperparameter tuning.

```
[71]: from sklearn.model_selection import RandomizedSearchCV

# Define the model
rf = RandomForestClassifier(random_state=42)
# Define the parameter distribution
param_dist = {
    'n_estimators': [50, 100, 200, 300, 400],
    'max_depth': [10, 20, 30, 40, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4, 6],
    'max_features': ['sqrt', 'log2', None]
}
```

```
[80]: # Randomized Search with 5-fold cross-validation
random_search = RandomizedSearchCV(estimator=rf, scoring='accuracy',
↪param_distributions=param_dist, n_iter=50, cv=5,
↪verbose=2,random_state=42,n_jobs=-1)
random_search.fit(x,y)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

[80]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42),
                   n_iter=50, n_jobs=-1,
                   param_distributions={'max_depth': [10, 20, 30, 40, None],
                                        'max_features': ['sqrt', 'log2', None],
                                        'min_samples_leaf': [1, 2, 4, 6],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [50, 100, 200, 300,
                                                         400]},
                   random_state=42, scoring='accuracy', verbose=2)
```

[74]:
```python
# Best parameters and accuracy
print("Best Parameters:", random_search.best_params_)
print("Best Cross-Validation Accuracy:", random_search.best_score_)
```

```
Best Parameters: {'n_estimators': 50, 'min_samples_split': 2,
'min_samples_leaf': 1, 'max_features': None, 'max_depth': 10}
Best Cross-Validation Accuracy: 0.9543393882937432
```

0.7.1 Saving the model

[82]:
```python
import joblib

# Save the best model
joblib.dump(random_search.best_estimator_, 'rf_model.joblib')
print("Model saved to rf_model.joblib")
```

```
Model saved to rf_model.joblib
```

[]:
```python

```