Group Name: " "

Member 1: Rodel Yzer C. De Boda

SR Code: 22-04506

Member 2: Adhem V. Hernandez

SR Code: 22-07415

Laboratory Activity 2: Applied Classification Analysis

In this laboratory we will create an classification algorithm for our gathered classification datasets.

Dataset Link: "https://www.kaggle.com/datasets/erdemtaha/cancer-data/data"

Description of the dataset

Insert Here: "The dataset contains the characteristics of patients diagnosed with cancer. The dataset contains a unique ID for each patient, the type of cancer (diagnosis), the visual characteristics of the cancer and the average values of these characteristics."

Import the Required Packages

For this exercise we will require the Pandas package for loading the data, the matplotlib package for plotting as well as scitkit-learn for creating the Classification model. Import all of the required packages and relevant modules for these tasks.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
import missingno as msno


import numpy as np
from array import array
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.metrics import accuracy_score, f1_score, precision_score
```

Read the Data

```
df = pd.read_csv("/content/Cancer_Data.csv")
```

Read the data and find summary statistics

Get the info of your dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
```

```
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Get the first five and last five of your dataset

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10

5 rows × 33 columns

```
df.tail(5)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00

5 rows × 33 columns

Get the summary statistics of your dataset to show the total cont, mean, standard deviation, min and max value, and percentiles of each column of your dataset.

```
df.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	conca points_me
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.0000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.0489
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.0388
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.0000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.0203
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.0335
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.0740
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.2012

8 rows × 32 columns

Data Cleaning

Visualize the missing values/data in your dataset to see how many data and percentage of data are missing in each column of your dataset.

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10

5 rows × 33 columns

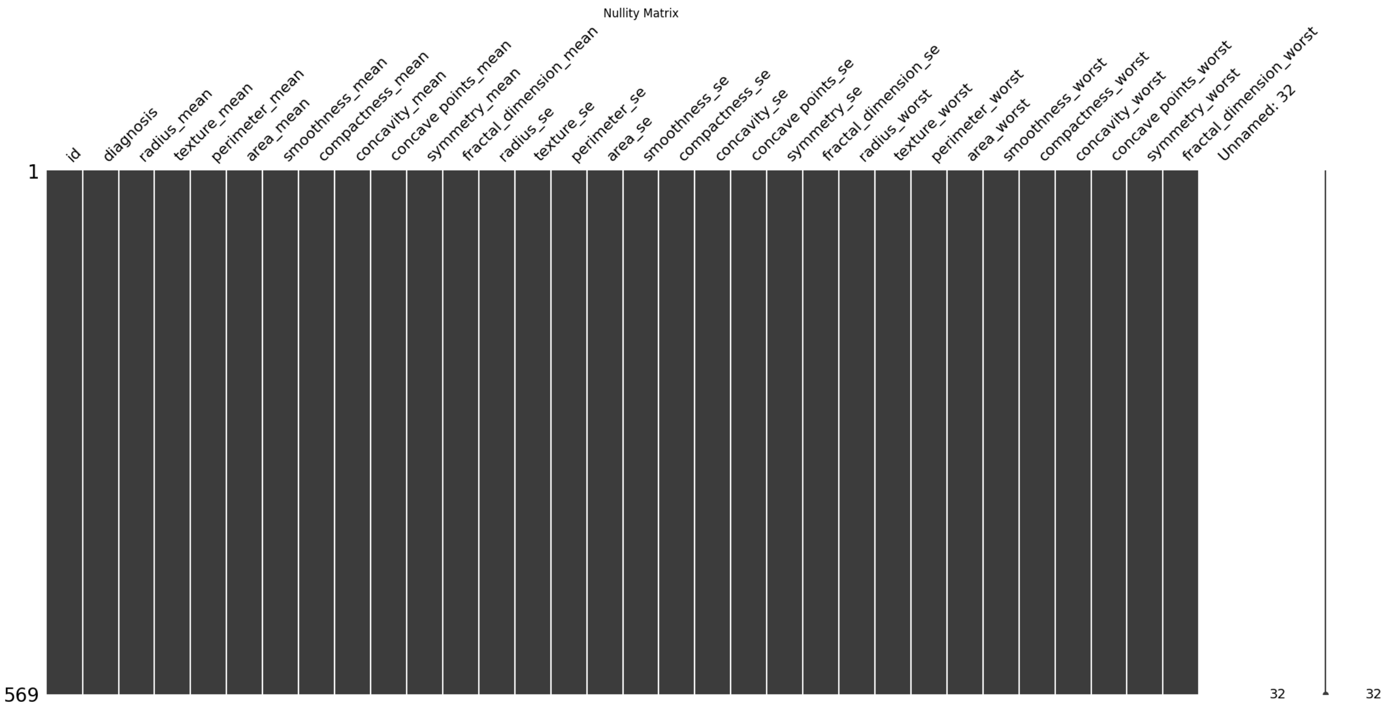
```
df.isnull().sum()
```

	0
id	0
diagnosis	0
radius_mean	0
texture_mean	0
perimeter_mean	0
area_mean	0
smoothness_mean	0
compactness_mean	0
concavity_mean	0
concave points_mean	0
symmetry_mean	0
fractal_dimension_mean	0
radius_se	0
texture_se	0
perimeter_se	0
area_se	0
smoothness_se	0
compactness_se	0
concavity_se	0
concave points_se	0
symmetry_se	0
fractal_dimension_se	0
radius_worst	0
texture_worst	0
perimeter_worst	0
area_worst	0
smoothness_worst	0
compactness_worst	0
concavity_worst	0
concave points_worst	0
symmetry_worst	0
fractal_dimension_worst	0
Unnamed: 32	569

dtype: int64

Use a nullity matrix for your dataset for easy visualization of missing data

```
msno.matrix(df)
plt.title('Nullity Matrix')
plt.show()
```



Imputation of data to missing data

```
#drop data with 80% missing percentage value
columns_to_drop = df.columns[df.isnull().sum() / len(df) * 100 > 80]
df.drop(columns_to_drop, axis=1, inplace=True)
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	conc points_
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10

5 rows × 32 columns

```
#dropping or deleting columns that is unnecessary
df.drop(['id'], axis=1, inplace=True)
df.head()
```

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

5 rows × 31 columns

After deleting the columns with 80% missing data. Impute data/value in the missing data of your dataset.

```
features = df.columns.drop(['diagnosis'])
for column in features:
  df[column].fillna(df[column].mean(), inplace=True)
```

```
<ipython-input-155-ea25aba2fed0>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[column].fillna(df[column].mean(), inplace=True)
```

After imputation of data, Check the completeness of your dataset by getting its info.

```
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
```

Visualize the Data

Load the dataset using Pandas and plot the different the target category

Plot a number of different features vs the allocated species classifications e.g. Sepal Length vs Petal length and Species. Visually inspect the plots and look for any patterns that could indicate separation between each of the species.

```
df['diagnosis'].unique()
```

```
array(['M', 'B'], dtype=object)
```

Preprocessing

Feature Engineering

You need to select the most appropriate features that will provide the most powerful classification model.

Before we can construct the model we must first convert the species values into labels that can be used within the model. Replace:

Example:

1. The species string Iris-setosa with the value 0
2. The species string Iris-versicolor with the value 1
3. The species string Iris-virginica with the value 2

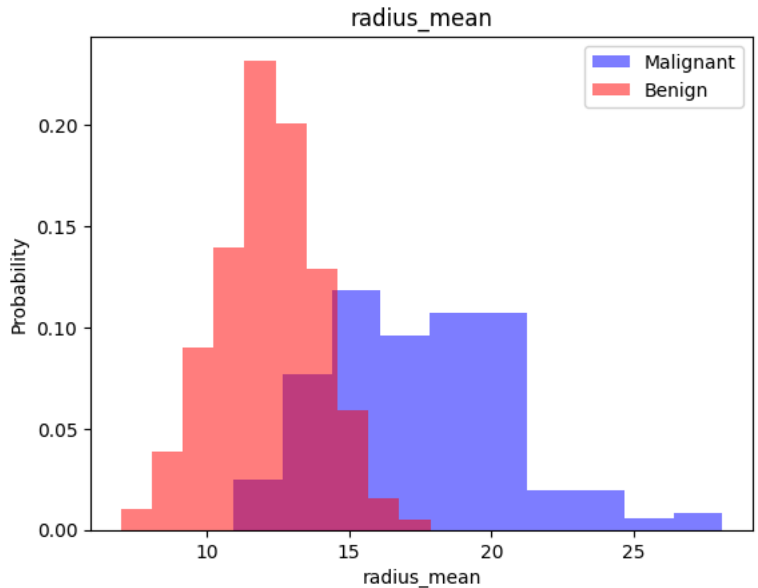
You can use get_dummies function / OnehotEncoder / LabelEncoder Library / StandardScaler with this.

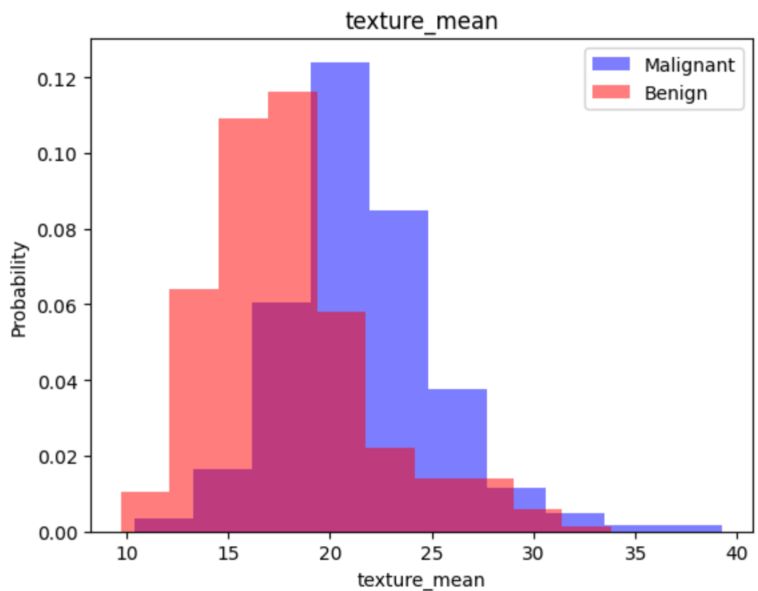
```
features
```

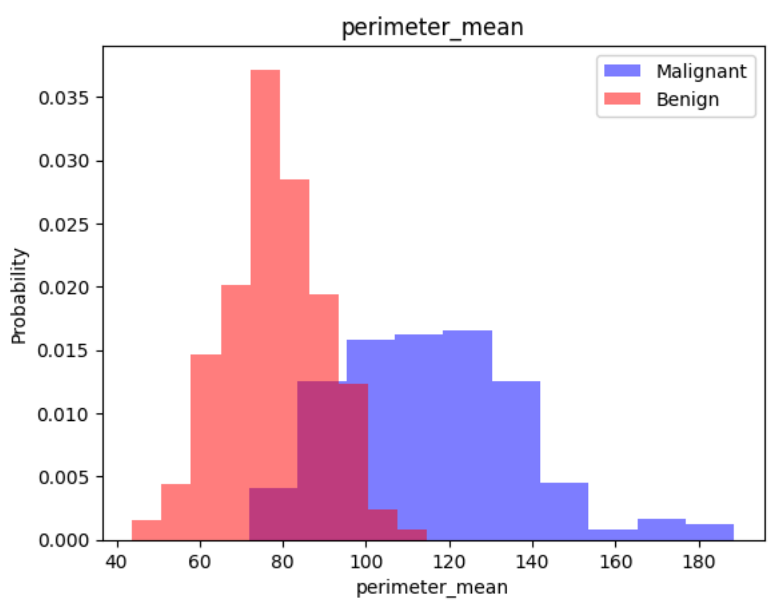
```
Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

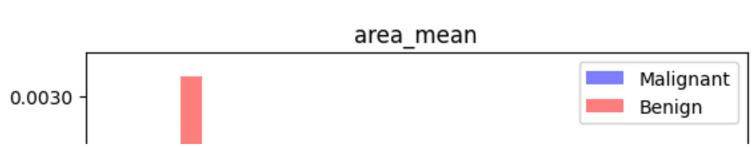
```
df['diagnosis'] = (df['diagnosis'] == 'M').astype(int)
```

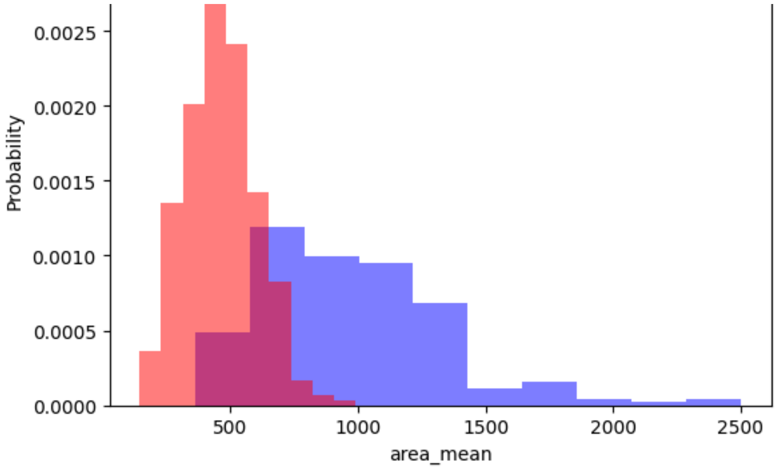
```
for feature in features:
    plt.hist(df[df['diagnosis']==1][feature], color='blue', label='Malignant', density=True, alpha=0.5)
    plt.hist(df[df['diagnosis']==0][feature], color='red', label='Benign', density=True, alpha=0.5)
    plt.ylabel('Probability')
    plt.xlabel(feature)
    plt.legend()
    plt.title(feature)
    plt.show()
```

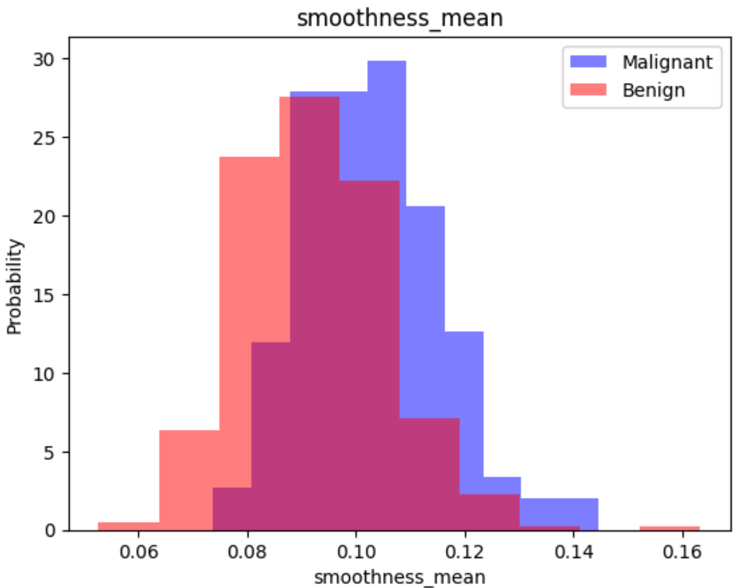

radius_mean
Malignant
Benign
Probability
0.00
0.05
0.10
0.15
0.20
10
15
20
25
radius_mean

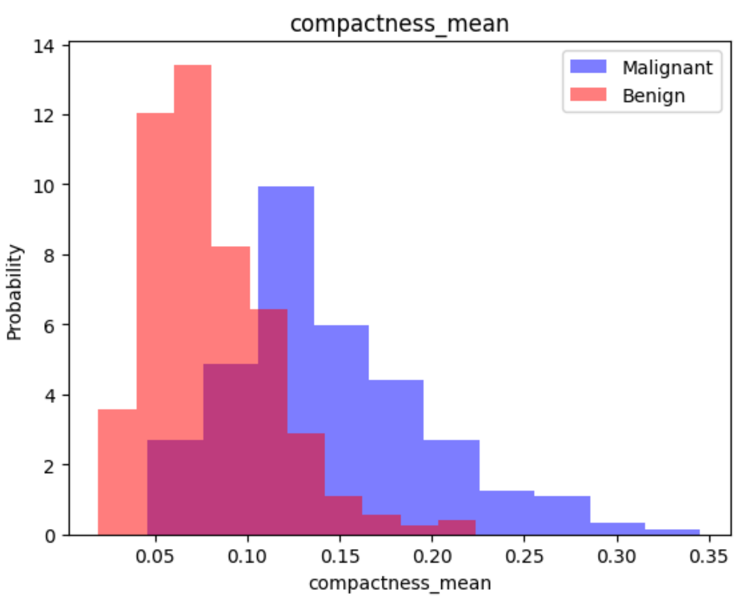

texture_mean
Malignant
Benign
Probability
0.00
0.02
0.04
0.06
0.08
0.10
0.12
10
15
20
25
30
35
40
texture_mean

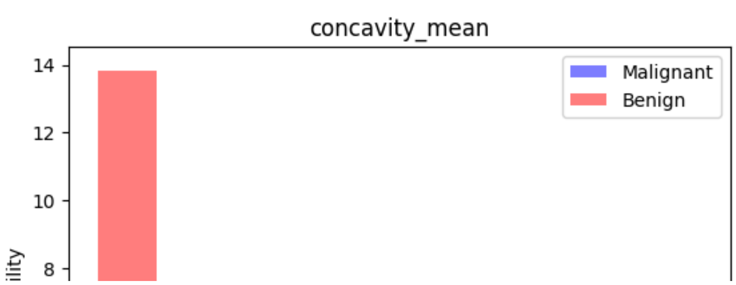

perimeter_mean
Malignant
Benign
Probability
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
40
60
80
100
120
140
160
180
perimeter_mean

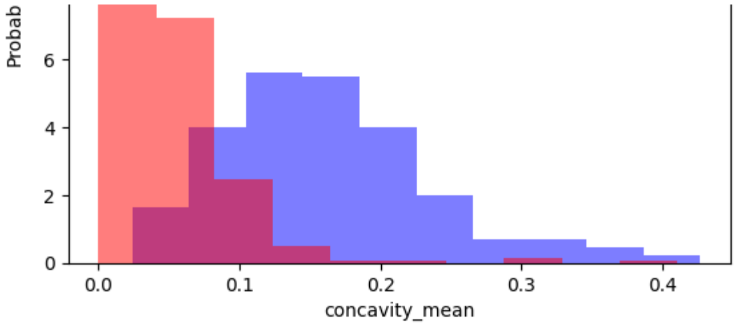

area_mean
Malignant
Benign
0.0030


Probability
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
500
1000
1500
2000
2500
area_mean

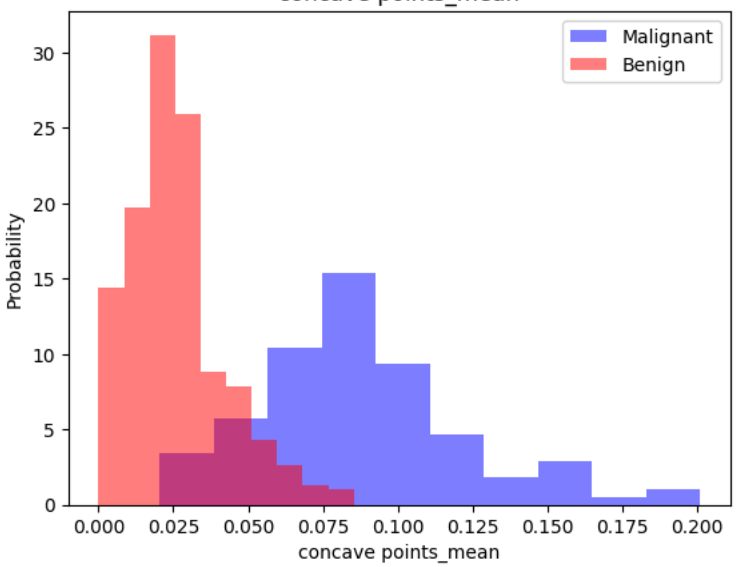

smoothness_mean
Malignant
Benign
Probability
30
25
20
15
10
5
0
0.06
0.08
0.10
0.12
0.14
0.16
smoothness_mean


compactness_mean
Malignant
Benign
Probability
14
12
10
8
6
4
2
0
0.05
0.10
0.15
0.20
0.25
0.30
0.35
compactness_mean

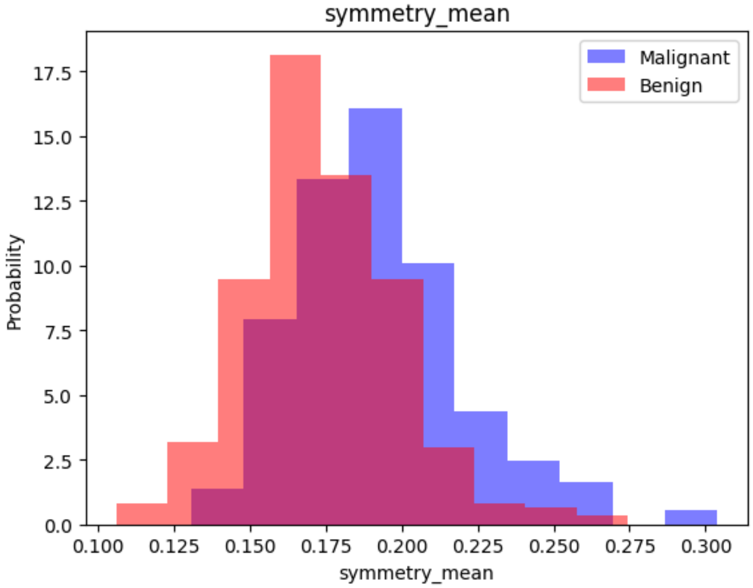

concavity_mean
Malignant
Benign
14
12
10
8



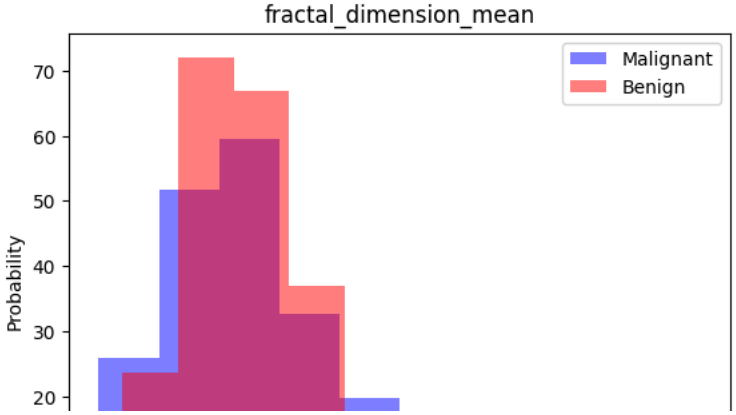
concave points_mean

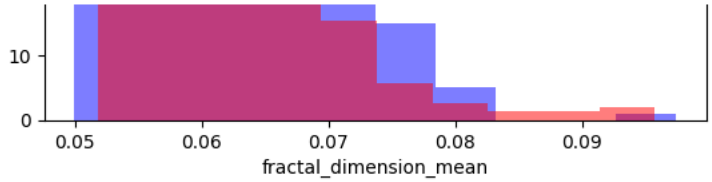


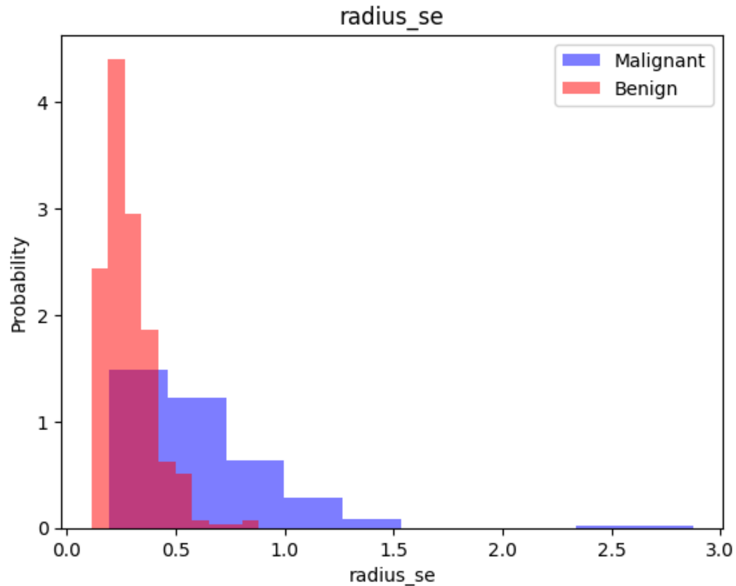
symmetry_mean

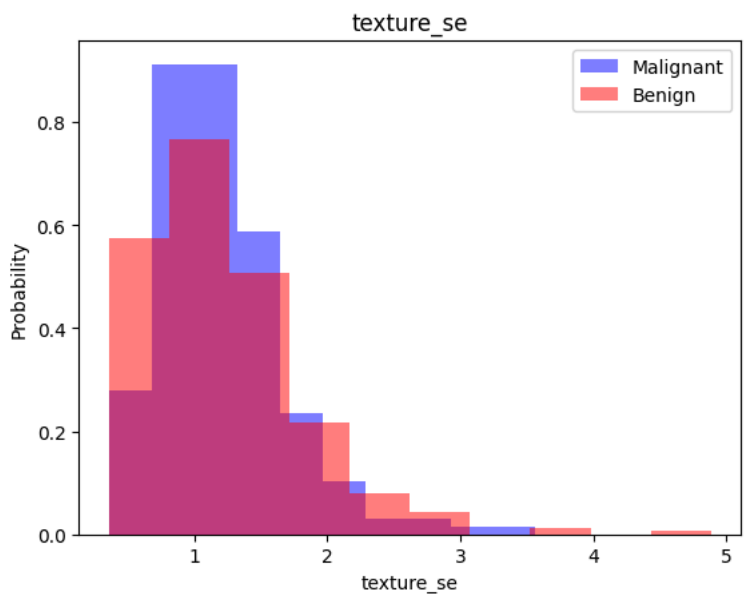


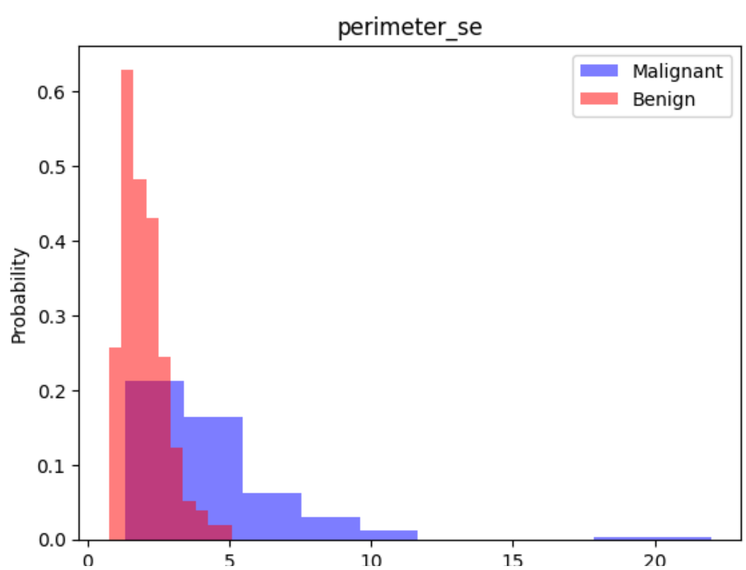
fractal_dimension_mean



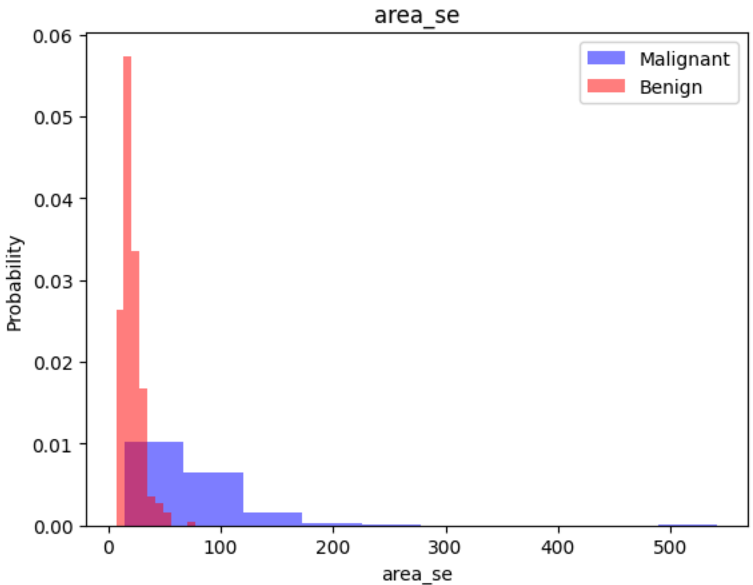

10
0
0.05
0.06
0.07
0.08
0.09
fractal_dimension_mean

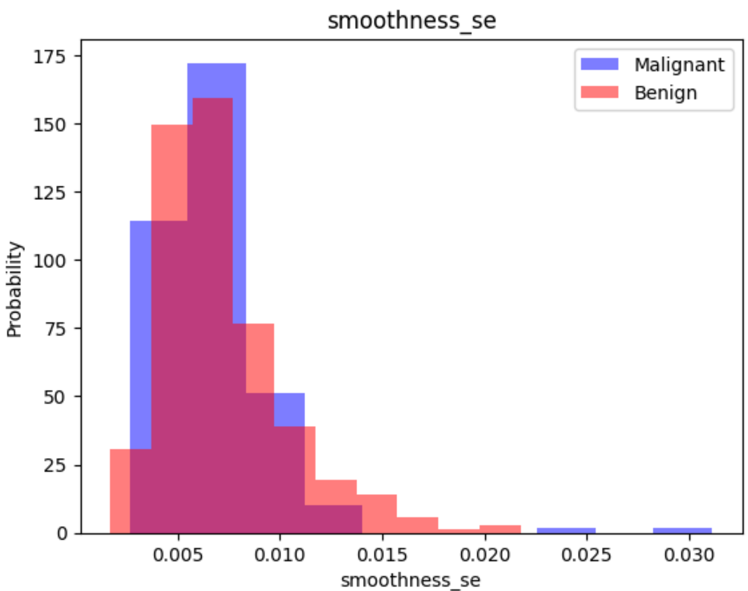

radius_se
Malignant
Benign
Probability
radius_se

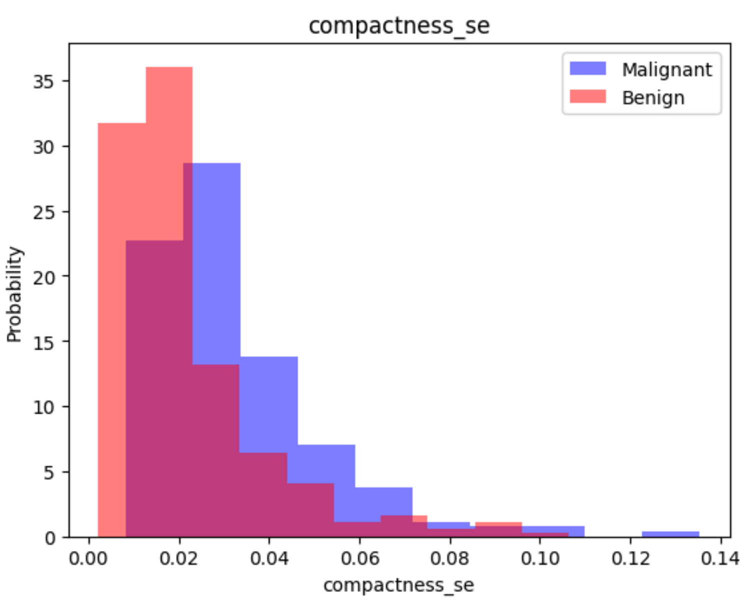

texture_se
Malignant
Benign
Probability
texture_se

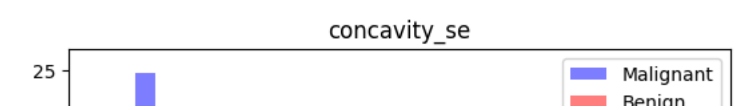

perimeter_se
Malignant
Benign
Probability

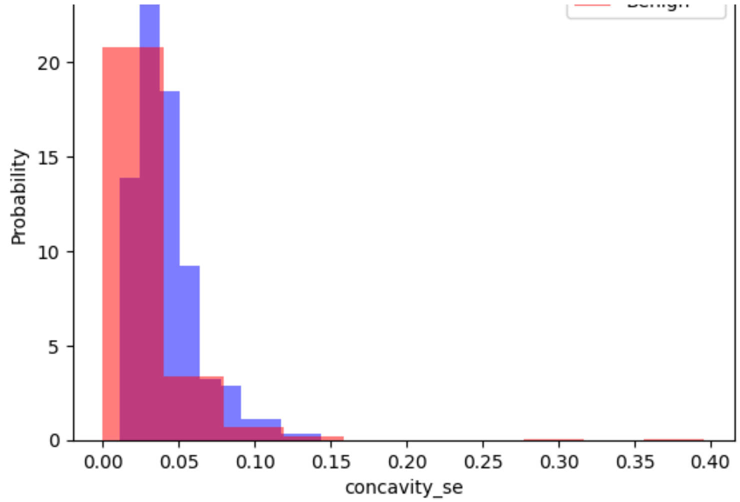

perimeter_se

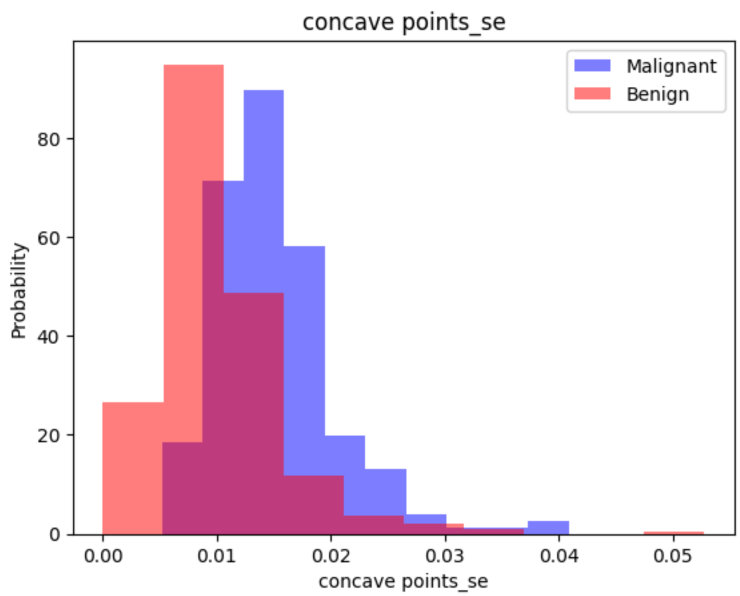

area_se
Malignant
Benign
Probability
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
100
200
300
400
500
area_se

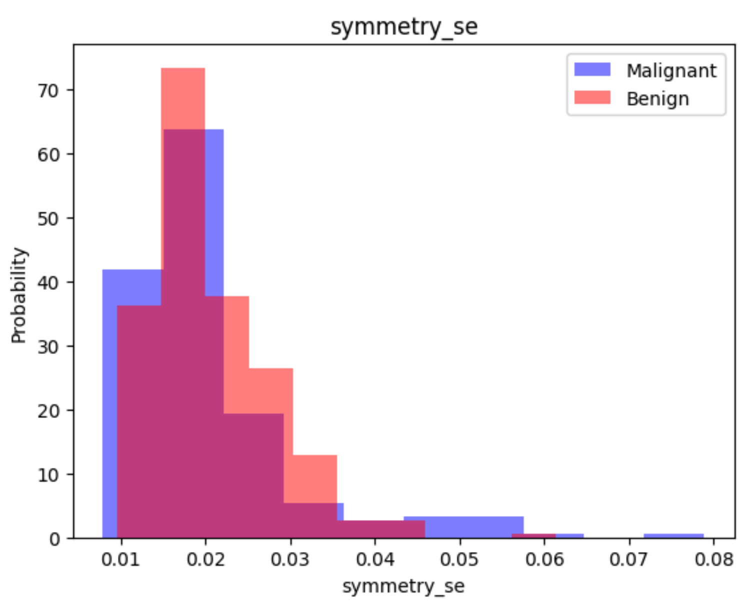

smoothness_se
Malignant
Benign
Probability
175
150
125
100
75
50
25
0
0.005
0.010
0.015
0.020
0.025
0.030
smoothness_se

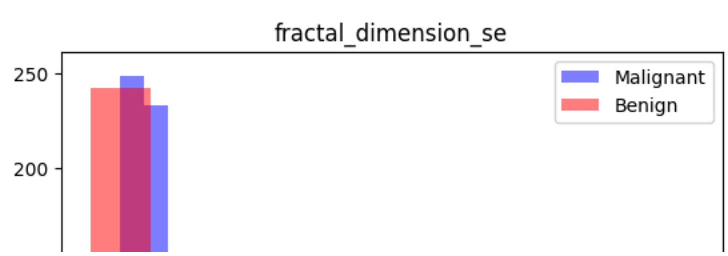

compactness_se
Malignant
Benign
Probability
35
30
25
20
15
10
5
0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
compactness_se

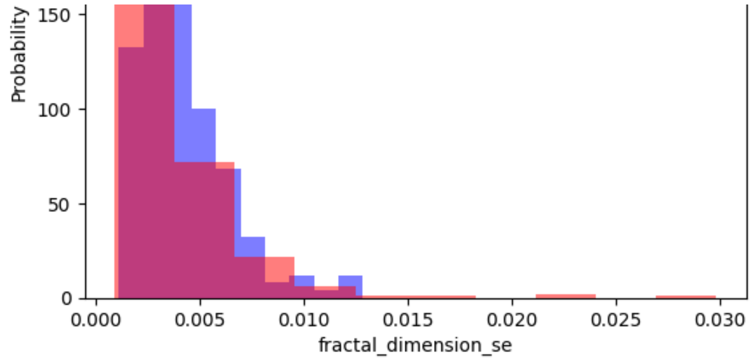

concavity_se
25
Malignant

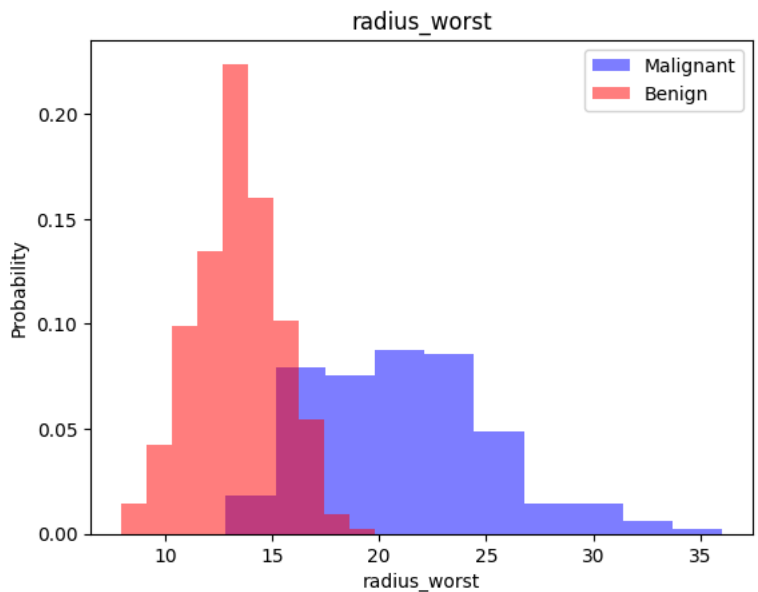

Benign
Probability
0
5
10
15
20
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
concavity_se

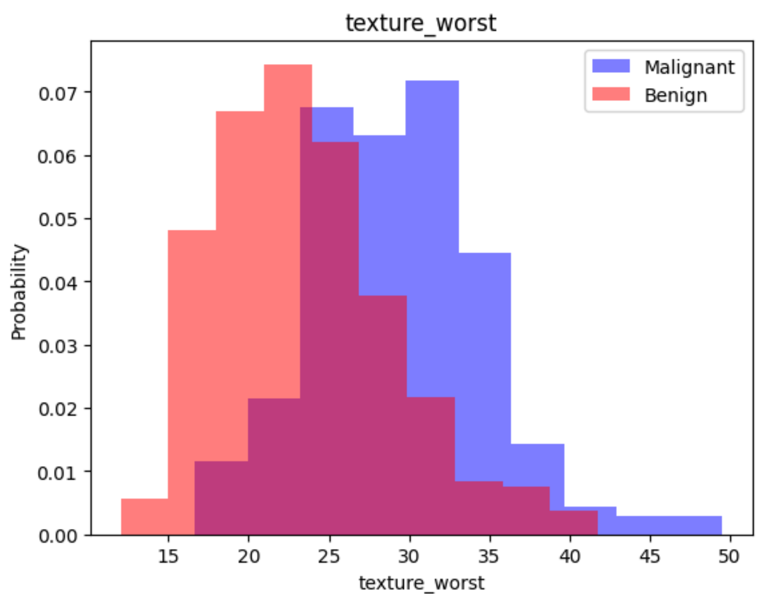

concave points_se
Malignant
Benign
Probability
0
20
40
60
80
0.00
0.01
0.02
0.03
0.04
0.05
concave points_se

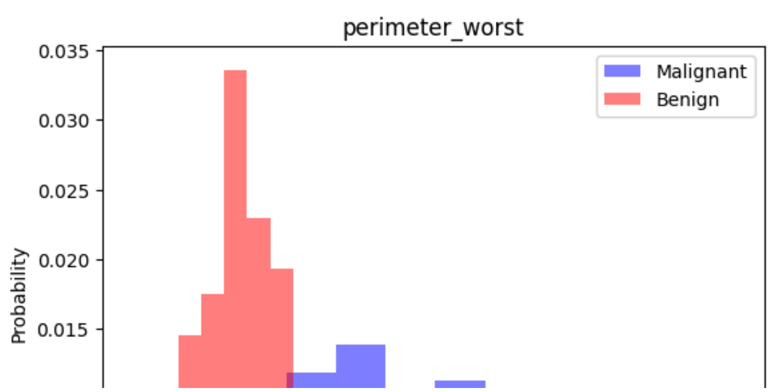

symmetry_se
Malignant
Benign
Probability
0
10
20
30
40
50
60
70
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
symmetry_se

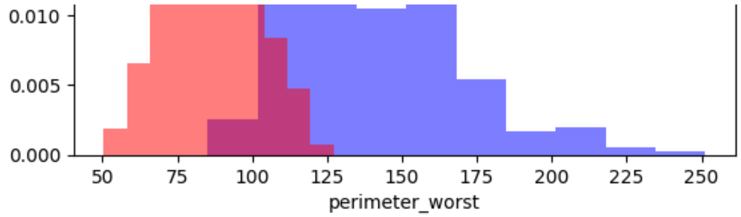

fractal_dimension_se
Malignant
Benign
200
250

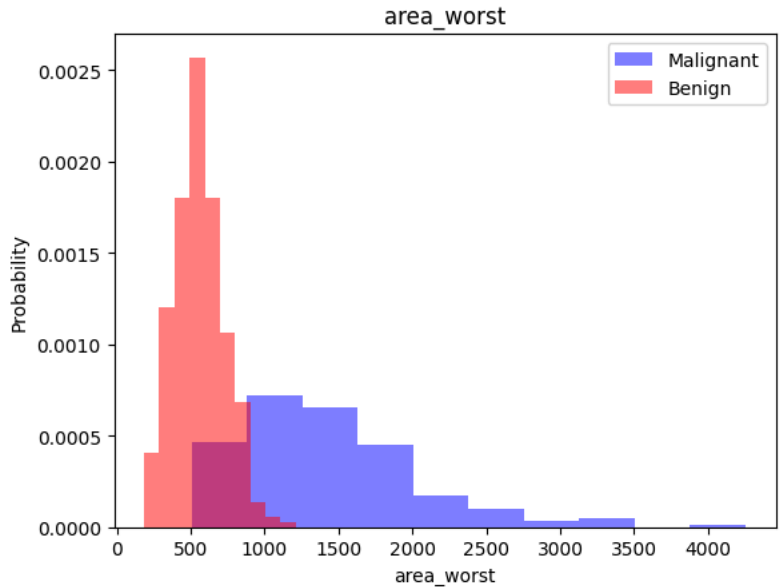

Probability
0 50 100 150
0.000 0.005 0.010 0.015 0.020 0.025 0.030
fractal_dimension_se

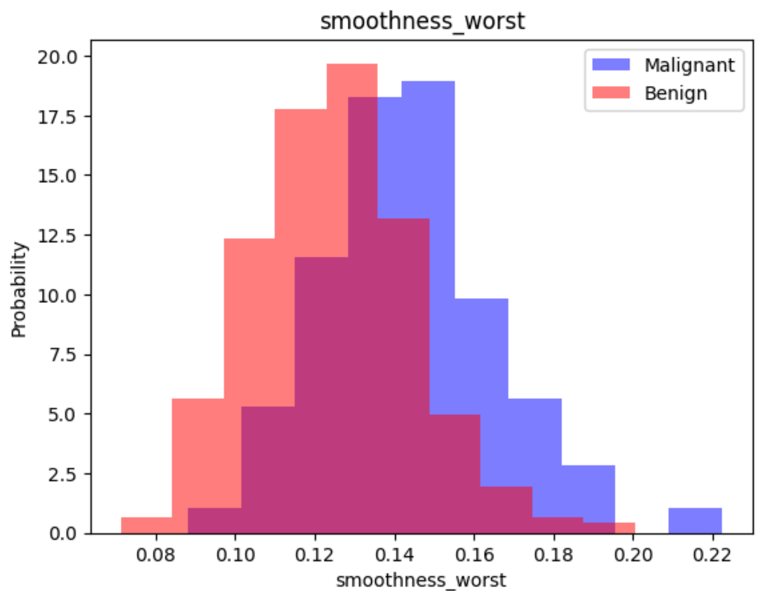

radius_worst
Malignant
Benign
Probability
0.00 0.05 0.10 0.15 0.20
10 15 20 25 30 35
radius_worst

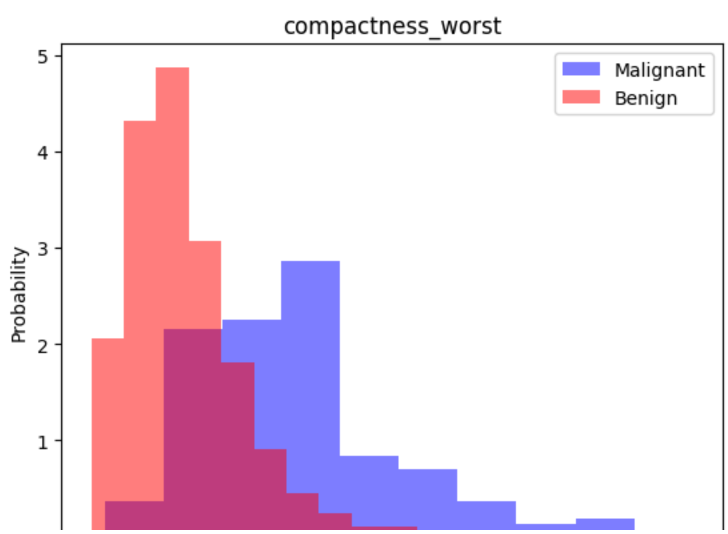

texture_worst
Malignant
Benign
Probability
0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07
15 20 25 30 35 40 45 50
texture_worst

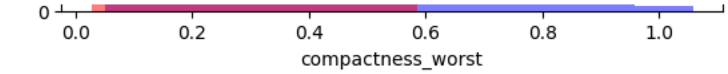

perimeter_worst
Malignant
Benign
Probability
0.015 0.020 0.025 0.030 0.035

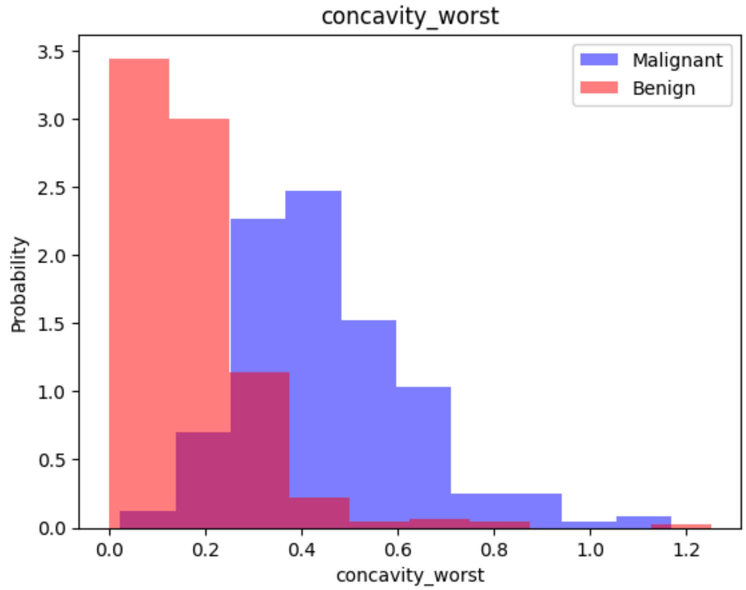

0.010
0.005
0.000
50
75
100
125
150
175
200
225
250
perimeter_worst

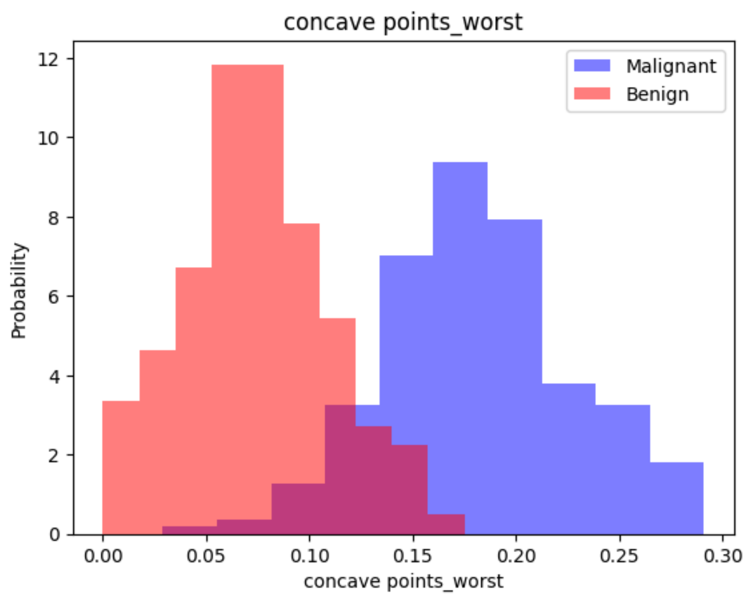

area_worst
Malignant
Benign
Probability
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
500
1000
1500
2000
2500
3000
3500
4000
area_worst

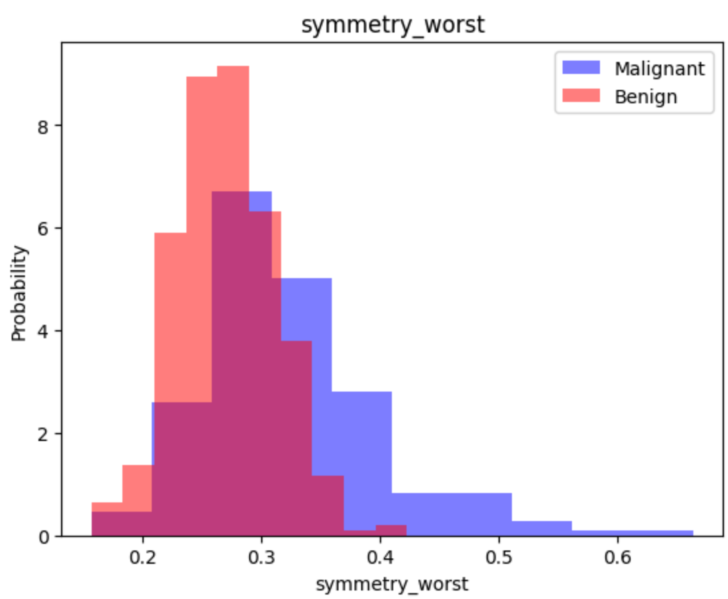

smoothness_worst
Malignant
Benign
Probability
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.08
0.10
0.12
0.14
0.16
0.18
0.20
0.22
smoothness_worst

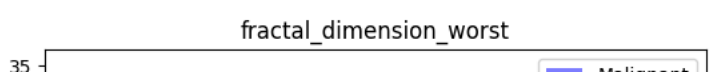

compactness_worst
Malignant
Benign
Probability
5
4
3
2
1

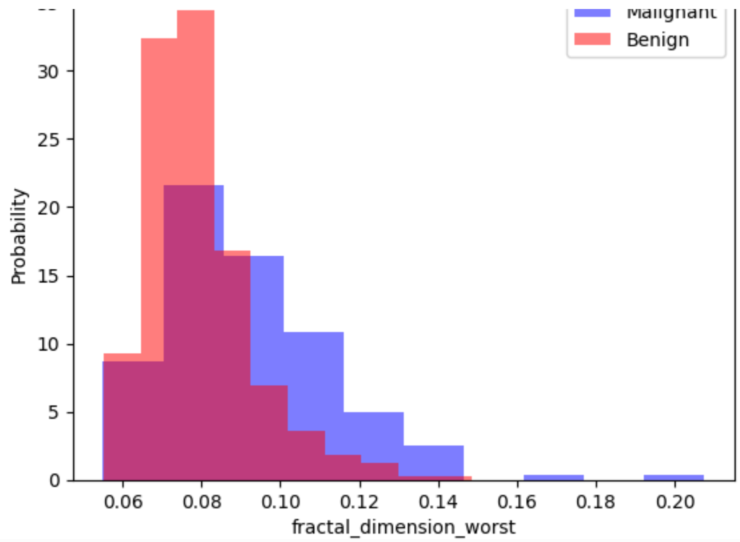

compactness_worst


concavity_worst
Malignant
Benign
Probability
concavity_worst


concave points_worst
Malignant
Benign
Probability
concave points_worst


symmetry_worst
Malignant
Benign
Probability
symmetry_worst


fractal_dimension_worst


Malignant
Benign
Probability
30
25
20
15
10
5
0
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
fractal_dimension_worst

```
# Get a list of all columns
cols = df.columns.tolist()

# Move the first column to the end
cols = cols[1:] + cols[:1]

# Reorder the DataFrame
df = df[cols]
```

```
train, valid, test = np.split(df.sample(frac=1), [int(0.6 * len(df)), int(0.8 * len(df))])
```

```
/usr/local/lib/python3.11/dist-packages/numpy/_core/fromnumeric.py:57: FutureWarning: 'DataFrame.swapaxes' is deprecated and will be rem
  return bound(*args, **kwds)
```

```
train
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
266	10.600	18.95	69.28	346.4	0.09688	0.11470	0.06387	0.02642	0.1922
321	20.160	19.66	131.10	1274.0	0.08020	0.08564	0.11550	0.07726	0.1928
511	14.810	14.70	94.66	680.7	0.08472	0.05016	0.03416	0.02541	0.1659
491	17.850	13.23	114.60	992.1	0.07838	0.06217	0.04445	0.04178	0.1220
318	9.042	18.90	60.07	244.5	0.09968	0.19720	0.19750	0.04908	0.2330
...	...	...	...	...	...	...	...	...	...
422	11.610	16.02	75.46	408.2	0.10880	0.11680	0.07097	0.04497	0.1886
31	11.840	18.70	77.93	440.6	0.11090	0.15160	0.12180	0.05182	0.2301
26	14.580	21.53	97.41	644.8	0.10540	0.18680	0.14250	0.08783	0.2252
217	10.200	17.48	65.05	321.2	0.08054	0.05907	0.05774	0.01071	0.1964
478	11.490	14.59	73.99	404.9	0.10460	0.08228	0.05308	0.01969	0.1779

341 rows × 31 columns

```
test
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
230	17.050	19.08	113.40	895.0	0.11410	0.15720	0.19100	0.109000	0.2131
529	12.070	13.44	77.83	445.2	0.11000	0.09009	0.03781	0.027980	0.1657
10	16.020	23.24	102.70	797.8	0.08206	0.06669	0.03299	0.033230	0.1528
252	19.730	19.82	130.70	1206.0	0.10620	0.18490	0.24170	0.097400	0.1733
46	8.196	16.84	51.71	201.9	0.08600	0.05943	0.01588	0.005917	0.1769
...	...	...	...	...	...	...	...	...	...
133	15.710	13.93	102.00	761.7	0.09462	0.09462	0.07135	0.059330	0.1816
509	15.460	23.95	103.80	731.3	0.11830	0.18700	0.20300	0.085200	0.1807
108	22.270	19.67	152.80	1509.0	0.13260	0.27680	0.42640	0.182300	0.2556
99	14.420	19.77	94.48	642.5	0.09752	0.11410	0.09388	0.058390	0.1879
145	11.900	14.65	78.11	432.8	0.11520	0.12960	0.03710	0.030030	0.1995

114 rows × 31 columns

```
valid
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
393	21.61	22.28	144.40	1407.0	0.11670	0.20870	0.28100	0.15620	0.2162
192	9.72	18.22	60.73	288.1	0.06950	0.02344	0.00000	0.00000	0.1653
397	12.80	17.46	83.05	508.3	0.08044	0.08895	0.07390	0.04083	0.1574
188	11.81	17.39	75.27	428.9	0.10070	0.05562	0.02353	0.01553	0.1718
437	14.04	15.98	89.78	611.2	0.08458	0.05895	0.03534	0.02944	0.1714
...	...	...	...	...	...	...	...	...	...
463	11.60	18.36	73.88	412.7	0.08508	0.05855	0.03367	0.01777	0.1516
545	13.62	23.23	87.19	573.2	0.09246	0.06747	0.02974	0.02443	0.1664
160	11.75	20.18	76.10	419.8	0.10890	0.11410	0.06843	0.03738	0.1993
507	11.06	17.12	71.25	366.5	0.11940	0.10710	0.04063	0.04268	0.1954
471	12.04	28.14	76.85	449.9	0.08752	0.06000	0.02367	0.02377	0.1854

114 rows × 31 columns

Select the features by writing the column names in the list below:

```
from imblearn.over_sampling import RandomOverSampler


def scale_dataset(dataframe, oversample=False):
    X = dataframe[dataframe.columns[:-1]].values
    y = dataframe[dataframe.columns[-1]].values

    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    if oversample:
      ros = RandomOverSampler()
      X, y = ros.fit_resample(X, y)

    data = np.hstack((X, np.reshape(y, (-1, 1))))

    return data, X, y


train, X_train, y_train = scale_dataset(train, oversample=True)
valid, X_valid, y_valid = scale_dataset(valid, oversample=False)
test, X_test, y_test = scale_dataset(test, oversample=False)
```

Constructing Logistic Regression Model

```
##Logistic Regression
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
y_train_pred_logistic = logistic_model.predict(X_train)
```

Display the prediction for the Logistic model against the training data (First Model)

Double-click (or enter) to edit

```
results = pd.DataFrame({'Actual': y_train, 'Predicted': y_train_pred_logistic})
results
```

	Actual	Predicted
0	0	0
1	1	1
2	0	0
3	0	0
4	0	0
...	...	...
421	1	1
422	1	1
423	1	1
424	1	1
425	1	1

426 rows × 2 columns

Next steps: Generate code with `results` View recommended plots New interactive sheet

Compute the accuracy of the model against the training set

```
accuracy_model1 = accuracy_score(y_train, y_train_pred_logistic)
f1_model1 = f1_score(y_train, y_train_pred_logistic)
precision_model1 = precision_score(y_train, y_train_pred_logistic)

print(f"F1 Score: {f1_model1}")
print(f"Precision: {precision_model1}")
print(f"Accuracy on training set: {accuracy_model1*100:.2f}% or {accuracy_model1}")
```

```
F1 Score: 0.9905660377358491
Precision: 0.995260663507109
Accuracy on training set: 99.06% or 0.9906103286384976
```

Construct the K-Nearest Neighbors Model

```
##Import Required Libraries of KNN here
from sklearn.neighbors import KNeighborsClassifier
```

Display the prediction for the KNN model against the training data (Second Model)

```
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model = knn_model.fit(X_train, y_train)
y_train_pred_knn = knn_model.predict(X_train)
```

Compute the accuracy of the model against the training set

```
accuracy_model2 = accuracy_score(y_train, y_train_pred_knn)
f1_model2 = f1_score(y_train, y_train_pred_knn)
precision_model2 = precision_score(y_train, y_train_pred_knn)

print(f"F1 Score: {f1_model2}")
print(f"Precision: {precision_model2}")
print(f"Accuracy on training set: {accuracy_model2*100:.2f}% or {accuracy_model2}")
```

```
F1 Score: 0.9813084112149533
Precision: 0.9767441860465116
Accuracy on training set: 98.12% or 0.9812206572769953
```

Constructing Decision Tree Model

```
##Decision Tree
from sklearn.tree import DecisionTreeClassifier


tree_model = DecisionTreeClassifier()
tree_model.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Display the prediction for the Decision Tree model against the training data (Third Model)

```
y_train_pred_tree = tree_model.predict(X_train)
```

Compute the accuracy of the model against the training set

```
accuracy_model3 = accuracy_score(y_train, y_train_pred_tree)
f1_model3 = f1_score(y_train, y_train_pred_tree)
precision_model3 = precision_score(y_train, y_train_pred_tree)

print(f"F1 Score: {f1_model3}")
print(f"Precision: {precision_model3}")
print(f"Accuracy on training set: {accuracy_model3*100:.2f}% or {accuracy_model3}")
```

```
F1 Score: 1.0
Precision: 1.0
Accuracy on training set: 100.00% or 1.0
```

Results

Get the highest model for your dataset using if else condition on every f1 score / accuracy / precision of your model.

```
if f1_model1 > f1_model2 and f1_model1 > f1_model3:
   f1_best_model = "Model 1"
elif f1_model2 > f1_model1 and f1_model2 > f1_model3:
   f1_best_model = "Model 2"
else:
   f1_best_model = "Model 3"

if precision_model1 > precision_model2 and precision_model1 > precision_model3:
   precision_best_model = "Model 1"
elif precision_model2 > precision_model1 and precision_model2 > precision_model3:
   precision_best_model = "Model 2"
else:
   precision_best_model = "Model 3"

if accuracy_model1 > accuracy_model2 and accuracy_model1 > accuracy_model3:
   accuracy_best_model = "Model 1"
elif accuracy_model2 > accuracy_model1 and accuracy_model2 > accuracy_model3:
   accuracy_best_model = "Model 2"
else:
   accuracy_best_model = "Model 3"
```

CONCLUSION

Create conclusion and result based on the output of every model.

```
print(f'Best F1 Score: {f1_best_model}')
print(f'Best Precision: {precision_best_model}')
print(f'Best Accuracy: {accuracy_best_model}')
```

```
Best F1 Score: Model 3
Best Precision: Model 3
Best Accuracy: Model 3
```

CONCULSION

he Decision Tree model (Model 3) was found to be the best-performing classification model overall after three models were evaluated on the provided dataset: Logistic Regression (Model 1), K-Nearest Neighbors (Model 2), and Decision Tree (Model 3).

The fact that Model 3 obtained the highest results on all three evaluation metrics—F1 Score, Precision, and Accuracy—supports this conclusion. This shows that the Decision Tree model demonstrated both high recall (reducing false negatives) and good precision (reducing false positives), correctly classifying occurrences in the dataset.

The Decision Tree model finally outperformed K-Nearest Neighbors and Logistic Regression, despite their encouraging performances. To enhance these models' performance on this specific dataset, more research and adjustment may be required.

Because the Decision Tree model performs better overall in terms of F1 Score, Precision, and Accuracy, it is advised for this classification assignment. It is crucial to remember that these findings are unique to the evaluation measures and dataset in question. Conclusions may vary depending on the evaluation criteria and datasets used.
Double-click (or enter) to edit