```
!pip install --upgrade pip
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.11/dist-packages (25.2)
```

```
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.11/dist-packages (0.13.0)
Requirement already satisfied: numpy<3,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (2.0.2)
Requirement already satisfied: scipy<2,>=1.10.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.16.1)
Requirement already satisfied: scikit-learn<2,>=1.3.2 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.6.1)
Requirement already satisfied: sklearn-compat<1,>=0.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (0.1.3)
Requirement already satisfied: joblib<2,>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.5.1)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (3.6.0)
```

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.19.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.14.1)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.74.0)
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.10.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.14.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.5.3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.37.1
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.5
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (202
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.8.2)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard~=2.19.
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.1.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.17.0)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard~=2.19.0->
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow) (3
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->te
```

```
pip install scikeras
```

```
Requirement already satisfied: scikeras in /usr/local/lib/python3.11/dist-packages (0.13.0)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.11/dist-packages (from scikeras) (3.10.0)
Requirement already satisfied: scikit-learn>=1.4.2 in /usr/local/lib/python3.11/dist-packages (from scikeras) (1.6.1)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras) (2.0.2)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras) (0.1.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras) (3.14.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras) (0.17.0)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras) (0.5.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from keras>=3.2.0->scikeras) (25.0)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.4.2->scikeras) (1.16.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.4.2->scikeras) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.4.2->scikeras) (3.6
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python3.11/dist-packages (from optree->keras>=3.2.0->scikeras)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.2.0->scikeras) (3.0
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.2.0->scikeras) (2
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->sc
```

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (3.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.16.1)
```

```
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import scipy
import time

from imblearn.over_sampling import SMOTE
from collections import Counter
import itertools

from sklearn.tree import DecisionTreeClassifier
from scipy.stats import pearsonr
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import chi2_contingency
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf
from keras.models import Sequential
from keras.layers import LSTM, Dense,Dropout,Activation,BatchNormalization
from scikeras import wrappers
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import roc_curve, auc as auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

from sklearn.model_selection import cross_val_score,cross_val_predict, cross_validate
from sklearn.metrics import make_scorer,precision_score, recall_score, f1_score, roc_auc_score, accuracy_score
from sklearn.preprocessing import StandardScaler

from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import SGDClassifier
```

```
import pandas as pd

file_path = "/Cancer_Data.csv"
df = pd.read_csv(file_path)
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	0.1
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1

5 rows × 33 columns

```
df = df.drop(['id'], axis=1)
df.head(5)
```

	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 × 32 columns

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

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	sy
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	

5 rows × 32 columns

EDA

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

EDA_END

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
from sklearn.preprocessing import LabelEncoder
```

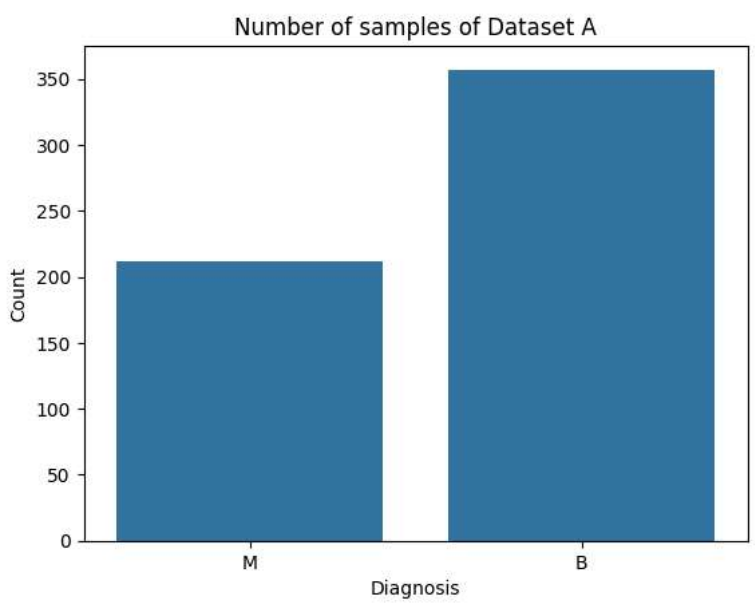
```
df.isnull().sum()
```

	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

```
print(df.diagnosis)
```

```
dtype: int64
0      M
1      M
2      M
3      M
4      M
      ..
564    M
565    M
566    M
567    M
568    B
Name: diagnosis, Length: 569, dtype: object
```

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.countplot(x='diagnosis', data=df)
plt.title('Number of samples of Dataset A')
plt.xlabel('Diagnosis')
plt.ylabel('Count')
plt.savefig('Number of samples of Dataset A.pdf', bbox_inches='tight')
plt.show()
```




```
df = df[df['diagnosis'] != 'ckd\t']
df.value_counts('diagnosis')
```

diagnosis	count
B	357
M	212

dtype: int64

```
x = df.drop(columns=['diagnosis'])
```

```
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 × 32 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 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
 31  Unnamed: 32             0 non-null      float64
dtypes: float64(31), object(1)
memory usage: 142.4+ KB
```

```
df_encoded = pd.get_dummies(df, columns=['diagnosis'], drop_first=True)



# Convert 'bool' columns to int (True -> 1, False -> 0)
bool_cols = df.select_dtypes(include='bool').columns
df[bool_cols] = df[bool_cols].astype(int)

# Convert 'object' columns to numeric using Label Encoding
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
object_cols = df.select_dtypes(include='object').columns

for col in object_cols:
    df[col] = df[col].astype(str)  # Ensure all values are strings
    df[col] = label_encoder.fit_transform(df[col])

# Check the result
print(df.head())
print(df.dtypes)
```

```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38          122.80     1001.0
1          1        20.57         17.77          132.90     1326.0
2          1        19.69         21.25          130.00     1203.0
3          1        11.42         20.38           77.58      386.1
4          1        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  ...  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

   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  Unnamed: 32
0          0.4601                  0.11890          NaN
1          0.2750                  0.08902          NaN
2          0.3613                  0.08758          NaN
3          0.6638                  0.17300          NaN
4          0.2364                  0.07678          NaN

[5 rows x 32 columns]
diagnosis                    int64
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
```

```
df.head()
```

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

5 rows × 32 columns

```
df.tail()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	sy
564	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	
565	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	
566	1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	
567	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	
568	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	

5 rows × 32 columns

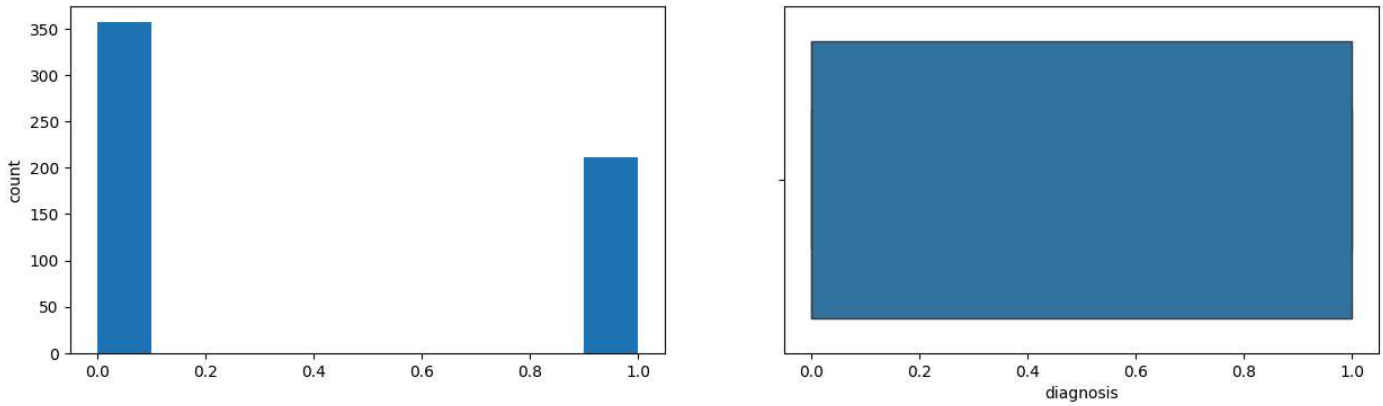
```
num_cols = df.shape[1]

print("Number of columns:", num_cols)
```

```
Number of columns: 32
```

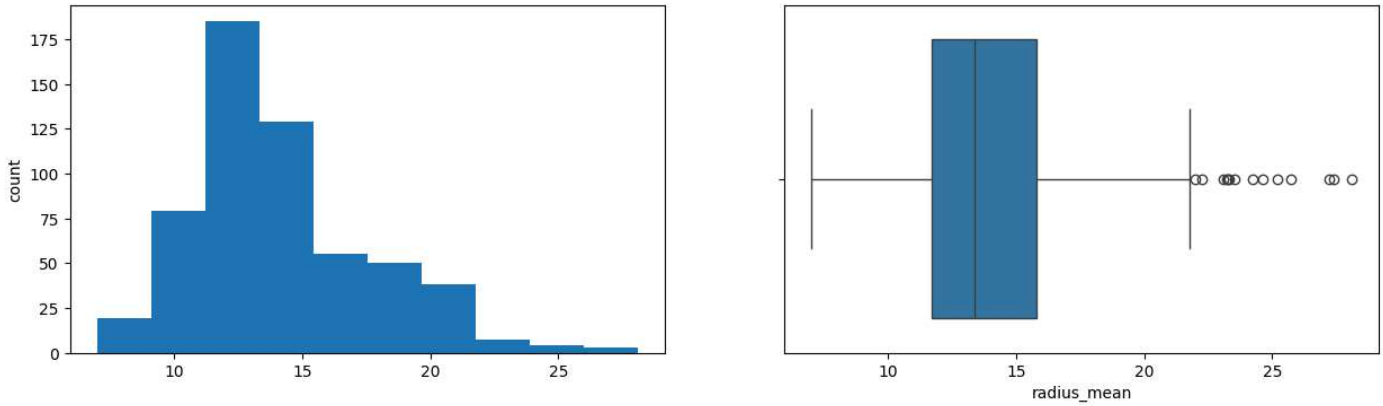
```
num_cols = len(df.columns)  # ❌ This gives an integer

num_cols = df.select_dtypes(include=['int64', 'float64']).columns
```

```
for col in num_cols:
    print(col)
    print('Skew :', round(df[col].skew(), 2))

    plt.figure(figsize=(15, 4))

    # Histogram
    plt.subplot(1, 2, 1)
    df[col].hist(grid=False)
    plt.ylabel('count')

    # Boxplot
    plt.subplot(1, 2, 2)
    sns.boxplot(x=df[col].dropna())

    plt.show()
```

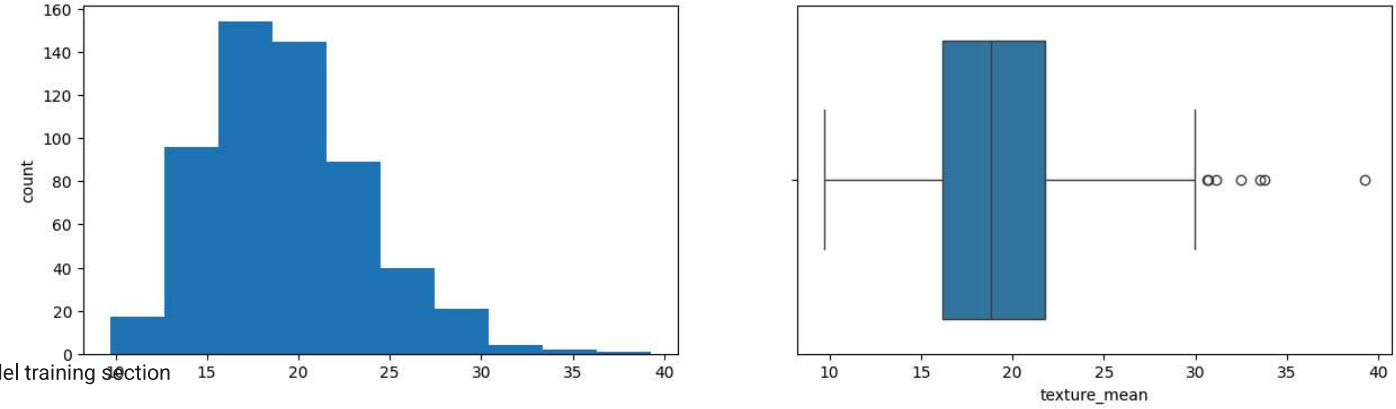
```
diagnosis
Skew : 0.53
```




```
radius_mean
Skew : 0.94
```




```
texture_mean
Skew : 0.65
```



Model training section

```
from sklearn.model_selection import train_test_split
```

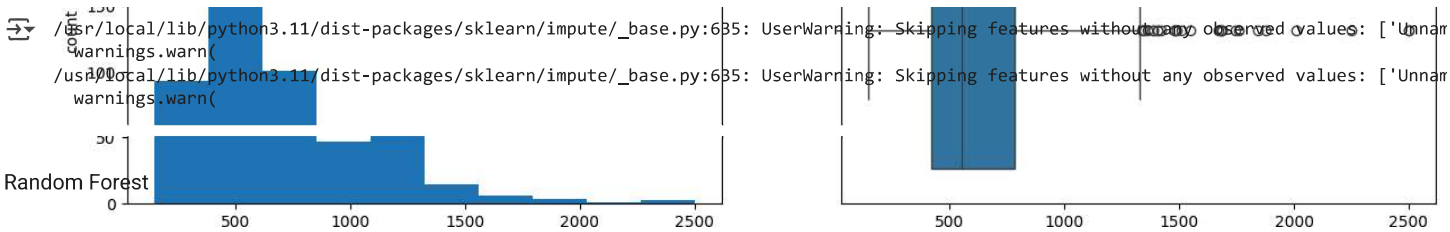
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
x = df.drop(columns=['diagnosis'])
y = df['diagnosis']
```



```
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# Impute missing values using the mean
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Feature normalization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarning: Skipping features without any observed values: ['Unnam
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/impute/_base.py:635: UserWarning: Skipping features without any observed values: ['Unnam
  warnings.warn(
```

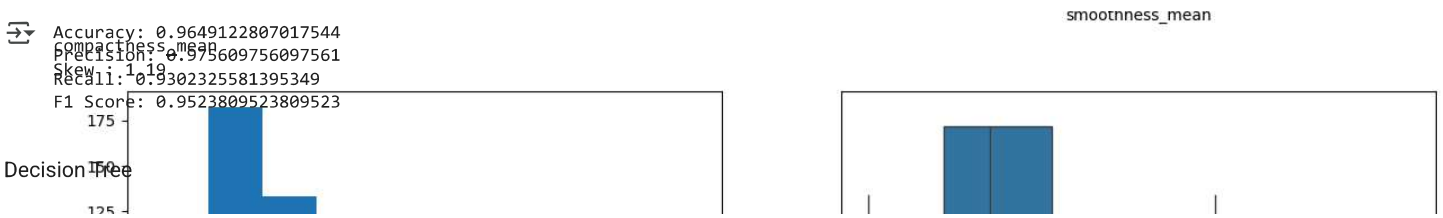


Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
# Calculate the metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
Accuracy: 0.9649122807017544
Precision: 0.975609756097561
Recall: 0.9302325581395349
F1 Score: 0.9523809523809523
```

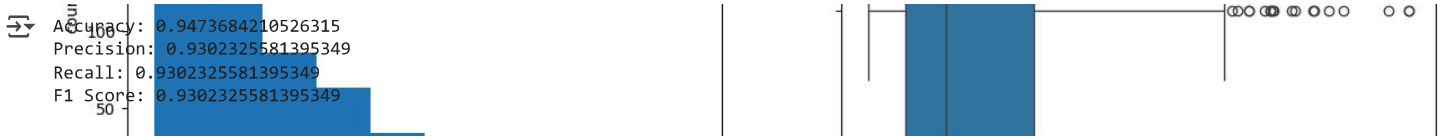


Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
# Calculate the metrics
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt)
recall_dt = recall_score(y_test, y_pred_dt)
f1_dt = f1_score(y_test, y_pred_dt)

# Print the metrics
print(f"Accuracy: {accuracy_dt}")
print(f"Precision: {precision_dt}")
print(f"Recall: {recall_dt}")
print(f"F1 Score: {f1_dt}")
```

```
Accuracy: 0.9473684210526315
Precision: 0.9302325581395349
Recall: 0.9302325581395349
F1 Score: 0.9302325581395349
```

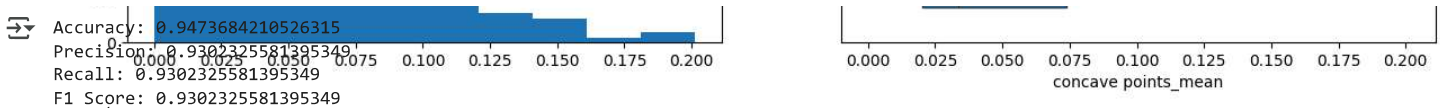


```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
```

```
Skew : 1.17
```

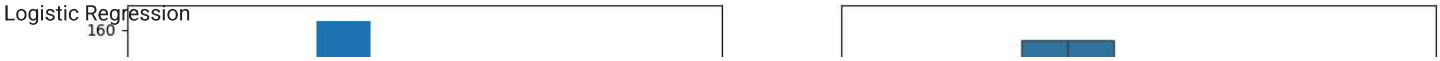
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
# Calculate the metrics
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn)
recall_knn = recall_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)

# Print the metrics
print(f"Accuracy: {accuracy_knn}")
print(f"Precision: {precision_knn}")
print(f"Recall: {recall_knn}")
print(f"F1 Score: {f1_knn}")
```

```
Accuracy: 0.9473684210526315
Precision: 0.9302325581395349
Recall: 0.9302325581395349
F1 Score: 0.9302325581395349
```




```
symmetry_mean
Skew : 0.73
```

Logistic Regression

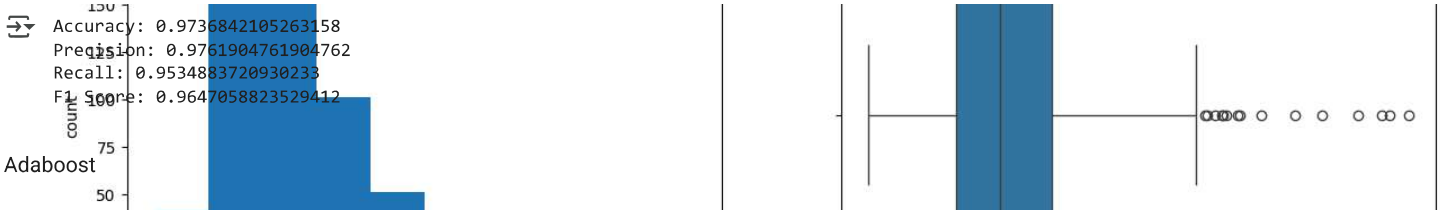


```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
```



```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
# Calculate the metrics
accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr)
recall_lr = recall_score(y_test, y_pred_lr)
f1_lr = f1_score(y_test, y_pred_lr)

# Print the metrics
print(f"Accuracy: {accuracy_lr}")
print(f"Precision: {precision_lr}")
print(f"Recall: {recall_lr}")
print(f"F1 Score: {f1_lr}")
```

```
Accuracy: 0.9736842105263158
Precision: 0.9761904761904762
Recall: 0.9534883720930233
F1 Score: 0.9647058823529412
```

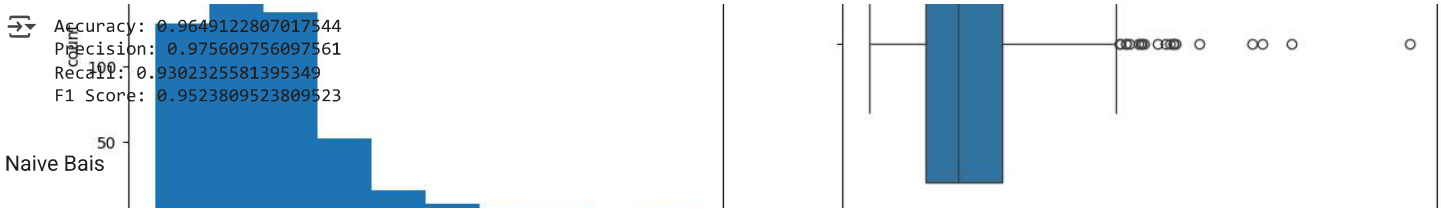


Adaboost

```
from sklearn.ensemble import AdaBoostClassifier
adaboost_model = AdaBoostClassifier(random_state=42)
adaboost_model.fit(X_train, y_train)
y_pred_ada = adaboost_model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
# Calculate the metrics
accuracy_ada = accuracy_score(y_test, y_pred_ada)
precision_ada = precision_score(y_test, y_pred_ada)
recall_ada = recall_score(y_test, y_pred_ada)
f1_ada = f1_score(y_test, y_pred_ada)

# Print the metrics
print(f"Accuracy: {accuracy_ada}")
print(f"Precision: {precision_ada}")
print(f"Recall: {recall_ada}")
print(f"F1 Score: {f1_ada}")
```

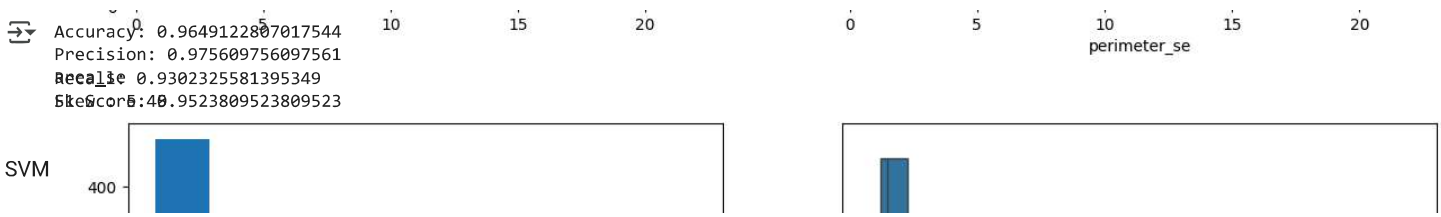
```
Accuracy: 0.9649122807017544
Precision: 0.975609756097561
Recall: 0.9302325581395349
F1 Score: 0.9523809523809523
```




```
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
# Calculate the metrics
accuracy_nb = accuracy_score(y_test, y_pred_nb)
precision_nb = precision_score(y_test, y_pred_nb)
recall_nb = recall_score(y_test, y_pred_nb)
f1_nb= f1_score(y_test, y_pred_nb)

# Print the metrics
print(f"Accuracy: {accuracy_nb}")
print(f"Precision: {precision_nb}")
print(f"Recall: {recall_nb}")
print(f"F1 Score: {f1_nb}")
```

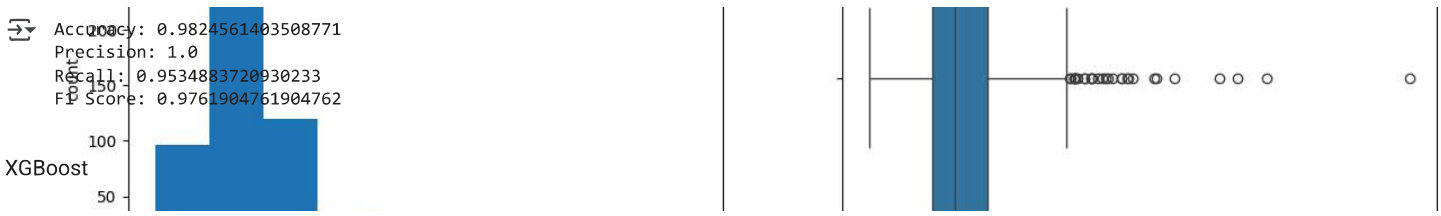
```
Accuracy: 0.9649122807017544
Precision: 0.975609756097561
Recall: 0.9302325581395349
F1 Score: 0.9523809523809523
```




```
from sklearn.svm import SVC
svm_model = SVC(random_state=42)
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
# Calculate the metrics
accuracy_svm= accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm)
f1_svm= f1_score(y_test, y_pred_svm)

# Print the metrics
print(f"Accuracy: {accuracy_svm}")
print(f"Precision: {precision_svm}")
print(f"Recall: {recall_svm}")
print(f"F1 Score: {f1_svm}")
```

```
Accuracy: 0.9824561403508771
Precision: 1.0
Recall: 0.9534883720930233
F1 Score: 0.9761904761904762
```

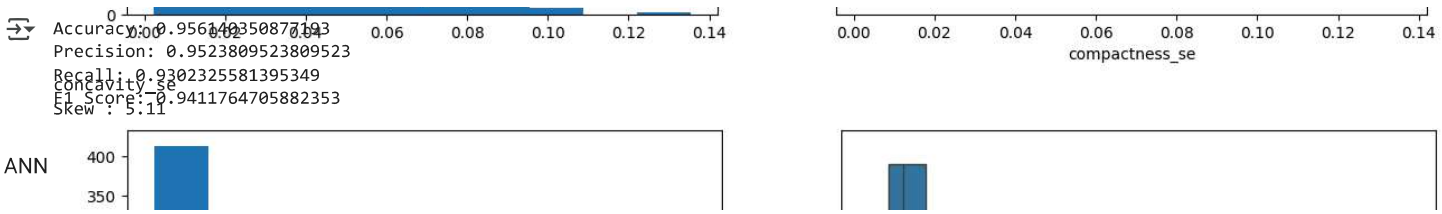



```
from xgboost import XGBClassifier
xgboost_model = XGBClassifier(random_state=42)
```

```
xgboost_model.fit(X_train, y_train)
y_pred_xgb = xgboost_model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
# Calculate the metrics
accuracy_xgb= accuracy_score(y_test, y_pred_xgb)
precision_xgb = precision_score(y_test, y_pred_xgb)
recall_xgb= recall_score(y_test, y_pred_xgb)
f1_xgb= f1_score(y_test, y_pred_xgb)

# Print the metrics
print(f"Accuracy: {accuracy_xgb}")
print(f"Precision: {precision_xgb}")
print(f"Recall: {recall_xgb}")
print(f"F1 Score: {f1_xgb}")
```

```
Accuracy: 0.956140350877193
Precision: 0.9523809523809523
Recall: 0.9302325581395349
F1 Score: 0.9411764705882353
```

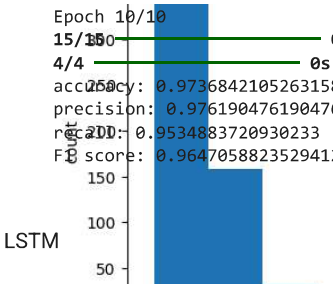


ANN

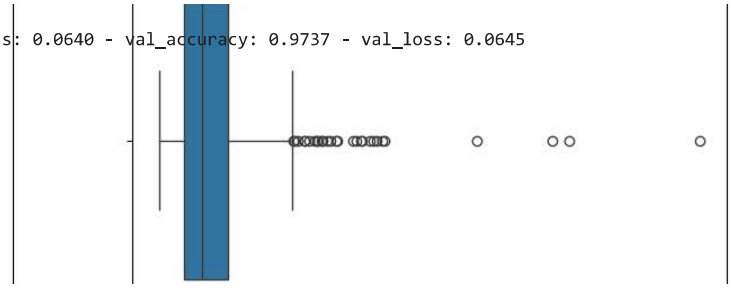

```
from keras.models import Sequential
from keras.layers import Dense, Dropout   # Add Dropout import
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Define ANN model
ann_model = Sequential()
ann_model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
ann_model.add(Dropout(0.2))
ann_model.add(Dense(32, activation='relu'))
ann_model.add(Dense(1, activation='sigmoid'))

# Compile the model
ann_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
ann_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
ann_predictions = ann_model.predict(X_test)
y_pred_ann = (ann_predictions > 0.5).astype("int32")   # Correct variable name

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred_ann)
precision = precision_score(y_test, y_pred_ann)
recall = recall_score(y_test, y_pred_ann)
f1 = f1_score(y_test, y_pred_ann)

# Output results
print(f"accuracy: {accuracy}")
print(f"precision: {precision}")
print(f"recall: {recall}")
print(f"F1 score: {f1}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 70ms/step - accuracy: 0.5680 - loss: 0.6547 - val_accuracy: 0.9386 - val_loss: 0.3926
Epoch 2/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9242 - loss: 0.3794 - val_accuracy: 0.9561 - val_loss: 0.2380
Epoch 3/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9367 - loss: 0.2384 - val_accuracy: 0.9649 - val_loss: 0.1533
Epoch 4/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9401 - loss: 0.1835 - val_accuracy: 0.9649 - val_loss: 0.1138
Epoch 5/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9587 - loss: 0.1319 - val_accuracy: 0.9649 - val_loss: 0.0934
Epoch 6/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9721 - loss: 0.1114 - val_accuracy: 0.9825 - val_loss: 0.0820
Epoch 7/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9484 - loss: 0.1106 - val_accuracy: 0.9737 - val_loss: 0.0753
Epoch 8/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9809 - loss: 0.0809 - val_accuracy: 0.9825 - val_loss: 0.0704
Epoch 9/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9718 - loss: 0.0863 - val_accuracy: 0.9825 - val_loss: 0.0674
```

```
Epoch 10/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9809 - loss: 0.0640 - val_accuracy: 0.9737 - val_loss: 0.0645
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
accuracy: 0.9736842105263158
precision: 0.9761904761904762
recall: 0.9534883720930233
F1 score: 0.9647058823529412
```



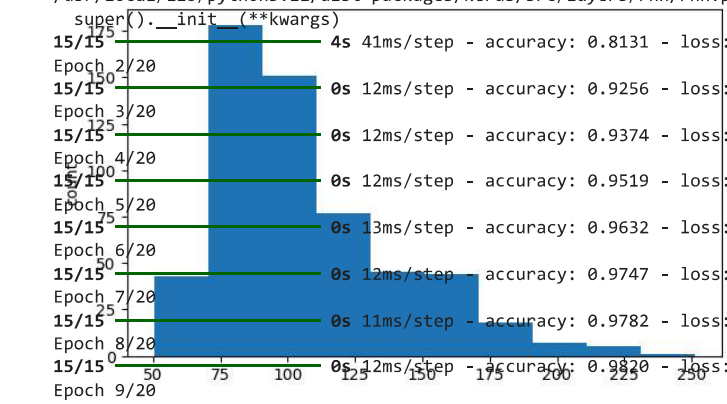
LSTM

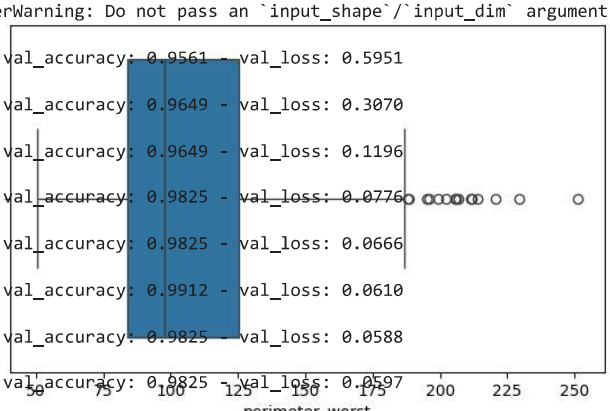


```
import numpy as np
import tensorflow as tf
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense
from sklearn.model_selection import train_test_split

if len(X_train.shape) == 2:
    X_train = np.array(X_train).reshape((X_train.shape[0], 1, X_train.shape[1]))
    X_test = np.array(X_test).reshape((X_test.shape[0], 1, X_test.shape[1]))

# Build LSTM model
model_lstm = Sequential()
model_lstm.add(LSTM(150, activation='relu', return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model_lstm.add(Dropout(0.2))
model_lstm.add(LSTM(150, activation='relu'))
model_lstm.add(Dense(1, activation='sigmoid'))

# Compile the model
model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model_lstm.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))

# Make predictions
y_pred = model_lstm.predict(X_test)
y_pred_lstm = (y_pred > 0.5).astype("int32")

# Calculate accuracy, precision, recall, and F1-score
accuracy = accuracy_score(y_test, y_pred_lstm)
precision = precision_score(y_test, y_pred_lstm)
recall = recall_score(y_test, y_pred_lstm)
f1 = f1_score(y_test, y_pred_lstm)

# Output results
print(f"accuracy: {accuracy}")
print(f"precision: {precision}")
print(f"recall: {recall}")
print(f"F1 score: {f1}")
```

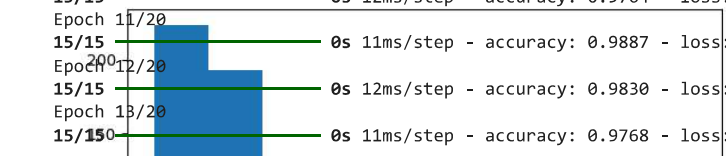
perimeter_worst
Skew : 1.13

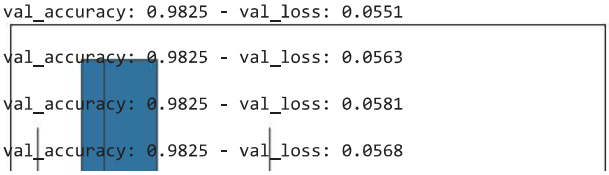
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 41ms/step - accuracy: 0.8131 - loss: 0.6781 - val_accuracy: 0.9561 - val_loss: 0.5951
Epoch 2/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9256 - loss: 0.5471 - val_accuracy: 0.9649 - val_loss: 0.3070
Epoch 3/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9374 - loss: 0.2442 - val_accuracy: 0.9649 - val_loss: 0.1196
Epoch 4/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9519 - loss: 0.1402 - val_accuracy: 0.9825 - val_loss: 0.0776
Epoch 5/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9632 - loss: 0.1274 - val_accuracy: 0.9825 - val_loss: 0.0666
Epoch 6/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9747 - loss: 0.0755 - val_accuracy: 0.9912 - val_loss: 0.0610
Epoch 7/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9782 - loss: 0.0616 - val_accuracy: 0.9825 - val_loss: 0.0588
Epoch 8/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9820 - loss: 0.0684 - val_accuracy: 0.9825 - val_loss: 0.0597
Epoch 9/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9870 - loss: 0.0576 - val_accuracy: 0.9912 - val_loss: 0.0563
Epoch 10/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9764 - loss: 0.0642 - val_accuracy: 0.9825 - val_loss: 0.0551
Epoch 11/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9887 - loss: 0.0470 - val_accuracy: 0.9825 - val_loss: 0.0563
Epoch 12/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9830 - loss: 0.0375 - val_accuracy: 0.9825 - val_loss: 0.0581
Epoch 13/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9768 - loss: 0.0445 - val_accuracy: 0.9825 - val_loss: 0.0568
```



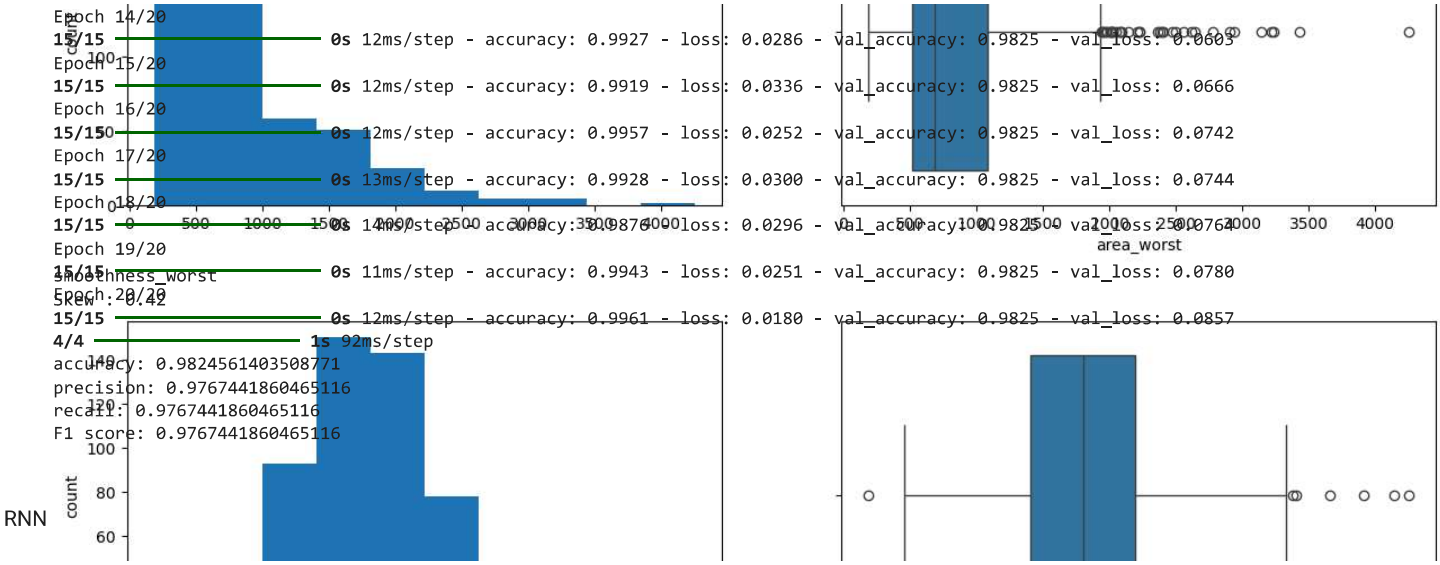


area_worst
Skew : 1.86





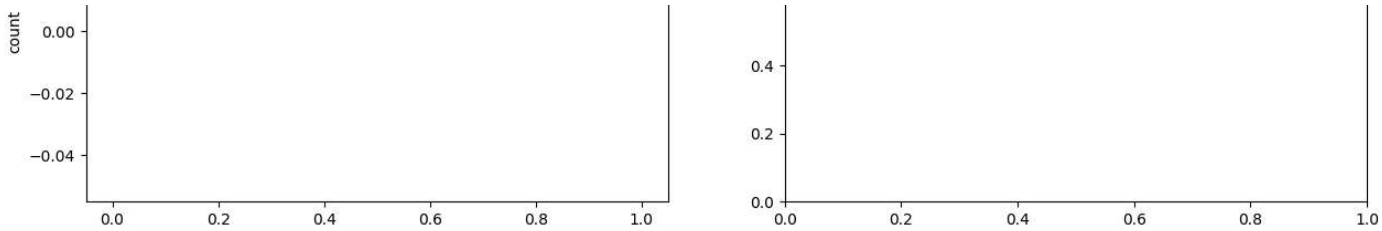
```
Epoch 14/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9927 - loss: 0.0286 - val_accuracy: 0.9825 - val_loss: 0.0603
Epoch 15/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9919 - loss: 0.0336 - val_accuracy: 0.9825 - val_loss: 0.0666
Epoch 16/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9957 - loss: 0.0252 - val_accuracy: 0.9825 - val_loss: 0.0742
Epoch 17/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9928 - loss: 0.0300 - val_accuracy: 0.9825 - val_loss: 0.0744
Epoch 18/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9876 - loss: 0.0296 - val_accuracy: 0.9825 - val_loss: 0.0764
Epoch 19/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9943 - loss: 0.0251 - val_accuracy: 0.9825 - val_loss: 0.0780
Epoch 20/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9961 - loss: 0.0180 - val_accuracy: 0.9825 - val_loss: 0.0857
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 92ms/step
accuracy: 0.9824561403508771
precision: 0.9767441860465116
recall: 0.9767441860465116
F1 score: 0.9767441860465116
```

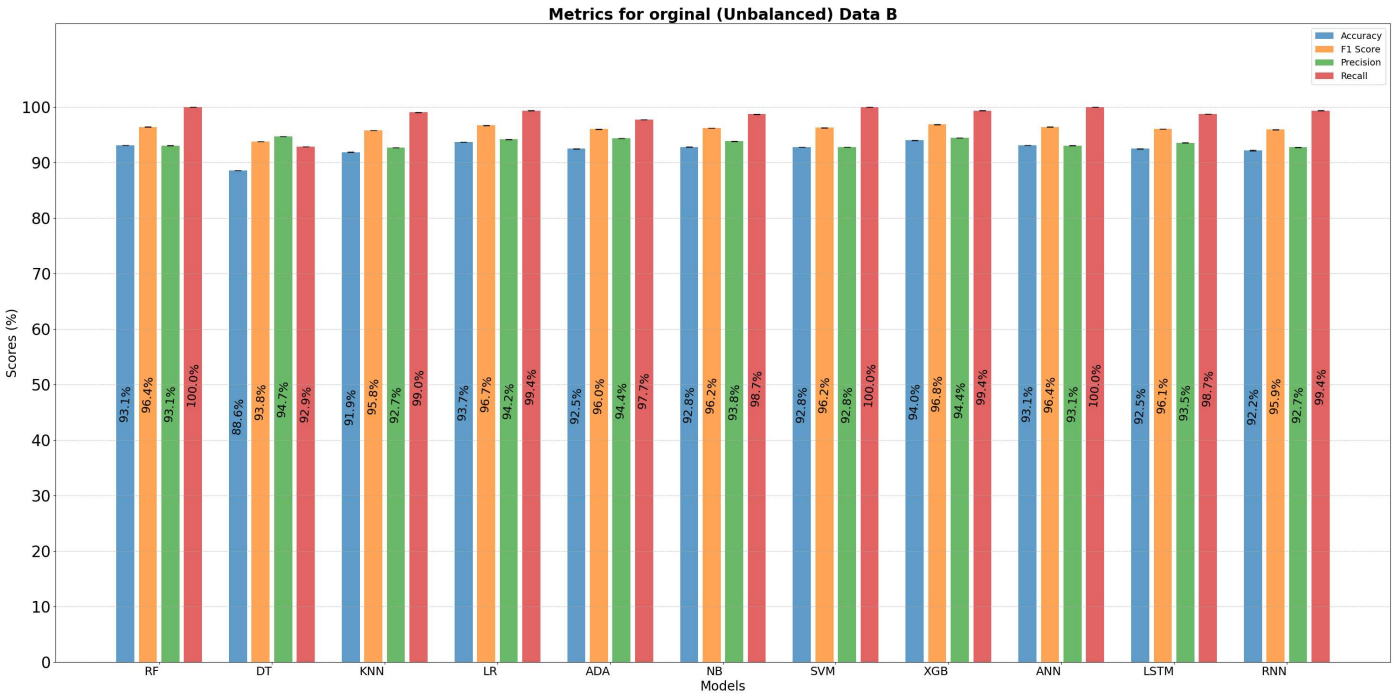


RNN

```
import matplotlib.pyplot as plt
import numpy as np

# Model names and their corresponding metrics
models = ['RF', 'DT', 'KNN', 'LR', 'ADA', 'NB', 'SVM', 'XGB', 'ANN', 'LSTM', 'RNN']

# Accuracy, Precision, Recall, F1 Score for each model (updated based on the provided data)
accuracy = [100 * x for x in [
    0.9307228915662651, 0.8855421686746988, 0.9186746987951807, 0.9367469879518072, 0.9246987951807228,
    0.927710843373494, 0.927710843373494, 0.9397590361445783, 0.9307228915662651, 0.9246987951807228,
    0.9216867469879518
]]
precision = [100 * x for x in [
    0.9305135951661632, 0.9470198675496688, 0.9270516717325228, 0.9415384615384615, 0.9435736677115988,
    0.9382716049382716, 0.927710843373494, 0.9444444444444444, 0.9305135951661632, 0.9353846153846154,
    0.9272727272727272
]]
recall = [100 * x for x in [
    1.0, 0.9285714285714286, 0.9902597402597403, 0.9935064935064936, 0.9772727272727273,
    0.987012987012987, 1.0, 0.9935064935064936, 1.0, 0.987012987012987, 0.9935064935064936
]]
f1_score = [100 * x for x in [
    0.9640062597809077, 0.9377049180327869, 0.957613814756672, 0.966824644549763, 0.960127591706539,
    0.9620253164556962, 0.9625, 0.9683544303797469, 0.9640062597809077, 0.9605055292259084,
    0.9592476489028213
]]

# Dummy error values for error bars (calculated or assumed for the sake of this example)
error_accuracies = [0.02, 0.03, 0.02, 0.02, 0.02, 0.03, 0.02, 0.01, 0.02, 0.02, 0.05]
error_precisions = [0.01, 0.02, 0.01, 0.02, 0.02, 0.02, 0.01, 0.01, 0.01, 0.02, 0.05]
error_recalls = [0.01, 0.01, 0.02, 0.01, 0.01, 0.02, 0.01, 0.01, 0.01, 0.01, 0.05]
error_f1_scores = [0.02, 0.03, 0.02, 0.02, 0.02, 0.03, 0.02, 0.01, 0.02, 0.02, 0.05]

# Metrics and labels
metrics = [accuracy, f1_score, precision, recall]
metrics_labels = ['Accuracy', 'F1 Score', 'Precision', 'Recall']
metrics_errors = [error_accuracies, error_f1_scores, error_precisions, error_recalls]

# Bar chart setup
x_pos = np.arange(len(models))
width = 0.2  # Bar width

fig, ax = plt.subplots(figsize=(30, 15))

# Plot grouped bars with error bars
for i, (metric, label, errors) in enumerate(zip(metrics, metrics_labels, metrics_errors)):
    bar = ax.bar(
        x_pos + i * width, metric, width * 0.8, yerr=errors,
        label=label, alpha=0.7, ecolor='black', capsize=5
    )

    # Add labels for the bar values inside the bars
    for j, (rect, value) in enumerate(zip(bar, metric)):
        height = rect.get_height()
```

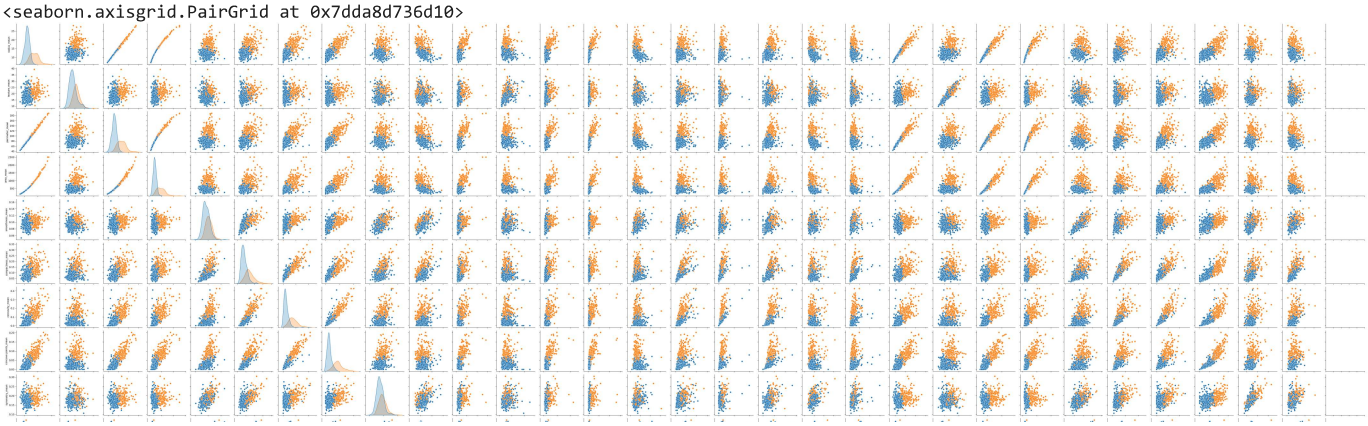
```
            offset_x = rect.get_x() + rect.get_width() / 2.0
            offset_y = height * 0.5

            # For `0.0%` values, place the text inside the bar with a small gap
            if value == 0.0:
                ax.text(
                    offset_x,
                    1,  # Add a small gap above the bottom of the bar
                    f'{value:.1f}%',
                    ha='center',
                    va='center',
                    fontsize=18,
                    color='black',
                    rotation=5,  # No rotation for 0.0%
                    rotation_mode='anchor'
                )

            else:
                ax.text(
                    offset_x,
                    offset_y,
                    f'{value:.1f}%',
                    ha='center',
                    va='center',
                    fontsize=18,
                    color='black',
                    rotation=85,  # Rotate the non-zero text
                    rotation_mode='anchor'
                )
    x_pos_adjusted = x_pos.copy()
    x_pos_adjusted[-1] += 0.2

    # Styling and layout
    ax.set_title('Metrics for orginal (Unbalanced) Data B', fontsize=24, weight='bold')
    ax.set_xlabel('Models', fontsize=20)
    ax.set_ylabel('Scores (%)', fontsize=20)
    ax.set_xticks(x_pos + width * 1.5)
    ax.set_xticklabels(models, ha= 'right', fontsize=18)
    ax.set_ylim(0, 115)
    ax.set_yticks(np.arange(0, 110, 10))
    ax.tick_params(axis='y', labelsize=24)
    ax.legend(fontsize=14, loc='upper right', bbox_to_anchor=(1, 1))
    ax.yaxis.grid(True, linestyle='--', alpha=0.7)

    # Adjust layout and save the figure
    plt.tight_layout()
    plt.savefig('Metrics for orginal (Unbalanced) Data B.pdf', bbox_inches='tight')
    plt.show()
```





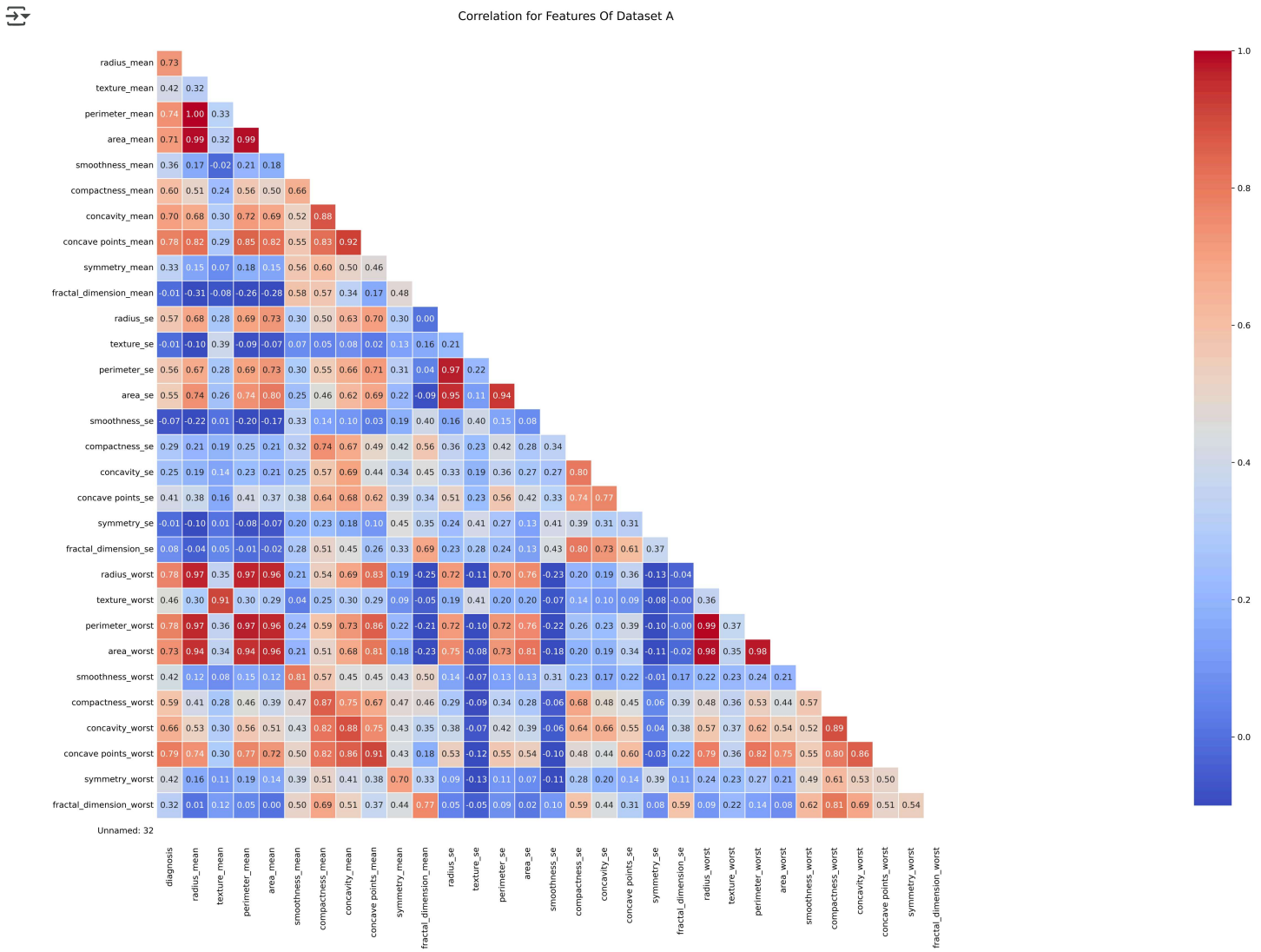

```
df.info()
sns.pairplot(df, hue = 'diagnosis')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   diagnosis                569 non-null    int64
 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
 31  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1)
memory usage: 142.4 KB
<seaborn.axisgrid.PairGrid at 0x7dda8d736d10>
```



```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Compute the correlation matrix
corr = df.corr(numeric_only=False)

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(21, 21), dpi=700)

# Draw the heatmap
ax = sns.heatmap(corr, vmin=-0.1, vmax=1.0, mask=mask,
                 annot=True, fmt=".2f", cmap='coolwarm',
                 square=True, linewidths=0.9,
                 cbar_kws={"shrink":0.6, "pad":0.01})

# Access the colorbar from the Axes object
cbar = ax.collections[0].colorbar
cbar.ax.set_position([0.95, 0.1, 0.03, 0.8])
```

```
# Customize labels
cust_labels = df.columns.tolist()
cust_labels[0] = ''
ax.set_yticklabels(cust_labels)
ax.tick_params(axis='y', which='both', length=0)
cust_labels = df.columns.tolist()
cust_labels[-1] = ''
ax.set_xticklabels(cust_labels)
ax.tick_params(axis='x', which='both', length=0)

plt.title("Correlation for Features Of Dataset A", fontsize=14)

# Save figure
plt.savefig("Correlation for Features Of Dataset A.pdf", bbox_inches='tight')

plt.show()
```



Gain Ratio

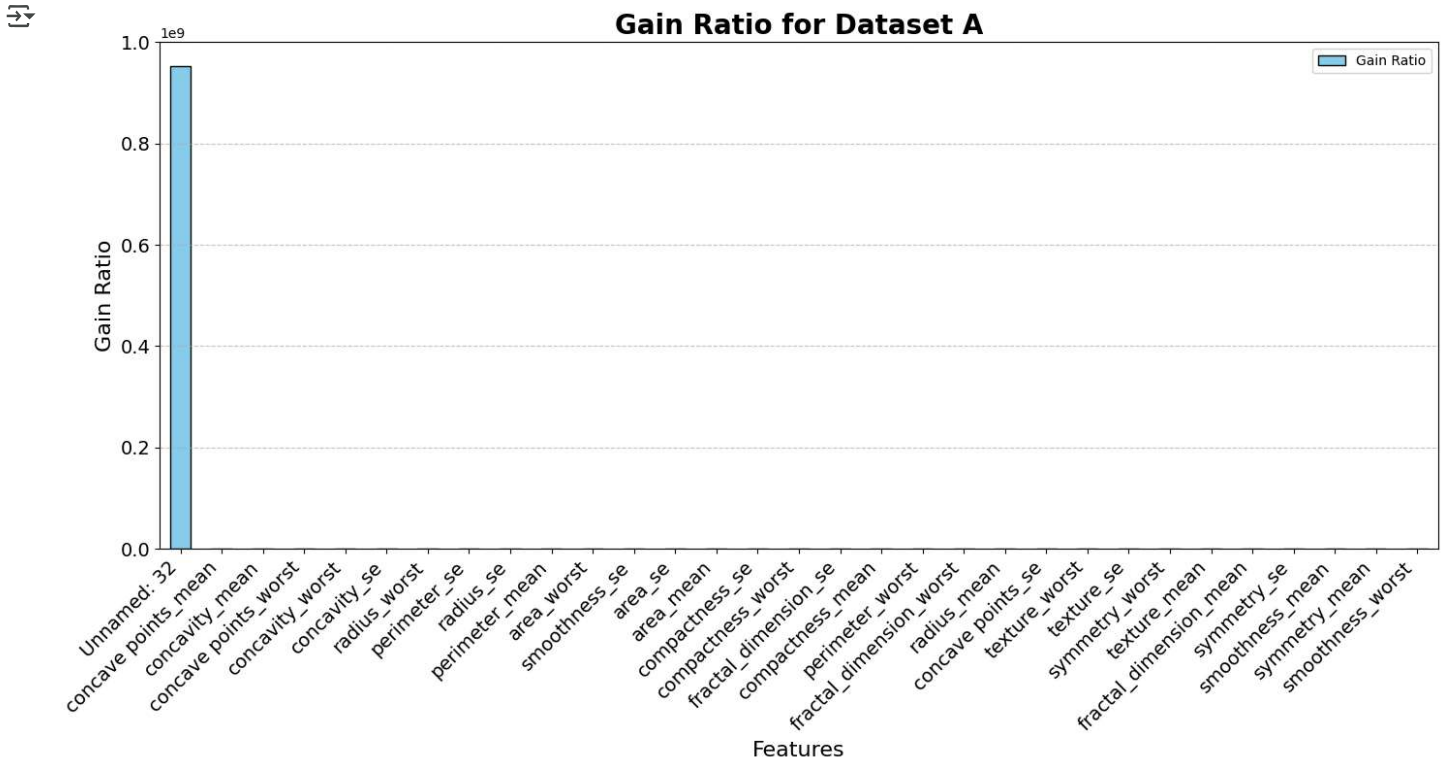
```
X_res = df.drop(columns=['diagnosis'])
y_res = df['diagnosis']
```

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import log_loss

# Compute Entropy of a distribution
def entropy(y_res):
    value_counts = y_res.value_counts(normalize=True)
    return -np.sum(value_counts * np.log2(value_counts + 1e-9))

# Compute Information Gain
def information_gain(X_res, y_res, feature):
    base_entropy = entropy(y_res)
    feature_entropy = 0
    for value in X_res[feature].unique():
        subset = y_res[X_res[feature] == value]
        feature_entropy += len(subset) / len(X_res) * entropy(subset)
    return base_entropy - feature_entropy

# Compute Split Information==
def split_information(X_res, feature):
    value_counts = X_res[feature].value_counts(normalize=True)
    return -np.sum(value_counts * np.log2(value_counts + 1e-9))

# Compute Gain Ratio
def gain_ratio(X_res, y_res, feature):
    ig = information_gain(X_res, y_res, feature)
    si = split_information(X_res, feature)
    return ig / (si + 1e-9)

# Compute Gain Ratios for all features
gain_ratios = {feature: gain_ratio(X_res, y_res, feature) for feature in X_res.columns}

# Convert to a DataFrame for easy visualization
gain_ratio_results = pd.DataFrame(gain_ratios.items(), columns=['Feature', 'Gain Ratio'])
gain_ratio_results = gain_ratio_results.sort_values(by="Gain Ratio", ascending=False)

# Display the results
print(gain_ratio_results)
```

```
                     Feature    Gain Ratio
30             Unnamed: 32  9.526351e+08
7       concave points_mean  1.045177e-01
6            concavity_mean  1.039405e-01
27     concave points_worst  1.036364e-01
26          concavity_worst  1.033368e-01
16             concavity_se  1.033053e-01
20             radius_worst  1.032616e-01
12             perimeter_se  1.032500e-01
10                radius_se  1.032013e-01
2            perimeter_mean  1.031608e-01
23               area_worst  1.031573e-01
14            smoothness_se  1.030375e-01
13                  area_se  1.027870e-01
3                 area_mean  1.025957e-01
15           compactness_se  1.020111e-01
25        compactness_worst  1.019363e-01
19     fractal_dimension_se  1.011792e-01
5           compactness_mean  1.005991e-01
22           perimeter_worst  1.003446e-01
29  fractal_dimension_worst  9.871665e-02
0                radius_mean  9.856645e-02
17        concave points_se  9.660127e-02
21             texture_worst  9.651886e-02
11                texture_se  9.634091e-02
28           symmetry_worst  9.496908e-02
1              texture_mean  9.470897e-02
9    fractal_dimension_mean  9.395664e-02
18               symmetry_se  9.204348e-02
4            smoothness_mean  8.840055e-02
8              symmetry_mean  8.523176e-02
24         smoothness_worst  8.426232e-02
```

```
# Plotting the gain ratios
plt.figure(figsize=(28, 8))
ax1 = plt.subplot(1, 2, 1)
gain_ratio_results.plot(kind='bar', color='skyblue', edgecolor='black', ax=ax1)
ax1.set_title('Gain Ratio for Dataset A', fontsize=20, fontweight='bold')
ax1.set_xlabel('Features', fontsize=16)
```

```
ax1.set_ylabel('Gain Ratio', fontsize=16)
ax1.set_xticklabels(gain_ratio_results['Feature'], rotation=45, ha='right', fontsize=14)
ax1.tick_params(axis='y', labelsize=14)  # Correct way to set y-tick label size
ax1.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig("Gain Ratio for Dataset A.pdf") # save figure
plt.show()
```



Information Gain

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import log_loss

# Compute Entropy of a distribution
def entropy(y_res):
    value_counts = y_res.value_counts(normalize=True)
    return -np.sum(value_counts * np.log2(value_counts + 1e-9))

# Compute Information Gain
def information_gain(X_res, y_res, feature):
    base_entropy = entropy(y_res)
    feature_entropy = 0
    for value in X_res[feature].unique():
        subset = y_res[X_res[feature] == value]
        feature_entropy += len(subset) / len(X_res) * entropy(subset)
    return base_entropy - feature_entropy

# Compute Information Gains for all features
information_gains = {feature: information_gain(X_res, y_res, feature) for feature in X_res.columns}

# Convert to a DataFrame for easy visualization
information_gain_results = pd.DataFrame(information_gains.items(), columns=['Feature', 'Information Gain']
information_gain_results = information_gain_results.sort_values(by="Information Gain", ascending=False)

# Display the results
```

```
print(information_gain_results)
```

```
                    Feature  Information Gain
30              Unnamed: 32          0.952635
7       concave points_mean          0.942090
14            smoothness_se          0.935060
```