Implementation & Results:-

In machine learning, many methods utilize binary classification. The most common are:

Support Vector Machines

Naive Bayes

Nearest Neighbor

Decision Trees

Logistic Regression

Neural Networks

###

We will use the breast cancer dataset from scikit-learn

Step 1: Import Packages and Display the dataset and perform shape, head, value counts Function

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
dataset = load_breast_cancer(as_frame=True)
dataset
```

```
..          ...          ...        ...
564     0.14100      0.21130     0.4107
565     0.11660      0.19220     0.3215
566     0.11390      0.30940     0.3403
567     0.16500      0.86810     0.9387
568     0.08996      0.06444     0.0000

     worst concave points  worst symmetry  worst fractal dimension  target
0                  0.2654          0.4601                  0.11890       0
1                  0.1860          0.2750                  0.08902       0
2                  0.2430          0.3613                  0.08758       0
3                  0.2575          0.6638                  0.17300       0
4                  0.1625          0.2364                  0.07678       0
..                    ...             ...                      ...     ...
564                0.2216          0.2060                  0.07115       0
565                0.1628          0.2572                  0.06637       0
566                0.1418          0.2218                  0.07820       0
567                0.2650          0.4087                  0.12400       0
568                0.0000          0.2871                  0.07039       1

[569 rows x 31 columns],
'target_names': array(['malignant', 'benign'], dtype='<U9'),
'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) dataset\n--------------------------------------------
n\n**Data Set Characteristics:**\n\n:Number of Instances: 569\n\n:Number of Attributes: 30 numeric, predictive attributes and the
lass\n\n:Attribute Information:\n    - radius (mean of distances from center to points on the perimeter)\n    - texture (standard
leviation of gray-scale values)\n    - perimeter\n    - area\n    - smoothness (local variation in radius lengths)\n    - compactness
perimeter^2 / area - 1.0)\n    - concavity (severity of concave portions of the contour)\n    - concave points (number of concave portions
f the contour)\n    - symmetry\n    - fractal dimension ("coastline approximation" - 1)\n\n    The mean, standard error, and "worst" or
argest (mean of the three\n    worst/largest values) of these features were computed for each image,\n    resulting in 30 features.  For
nstance, field 0 is Mean Radius, field\n    10 is Radius SE, field 20 is Worst Radius.\n\n    - class:\n            - WDBC-Malignant\n
 WDBC-Benign\n\n:Summary Statistics:\n\n===================================== ====== ======\n                                      Min
lax\n===================================== ====== ======\nradius (mean):                        6.981  28.11\ntexture (mean):
.71   39.28\nperimeter (mean):                    43.79  188.5\narea (mean):                        143.5  2501.0\nsmoothness (mean):
.053  0.163\ncompactness (mean):                  0.019  0.345\nconcavity (mean):                   0.0    0.427\nconcave points
mean):                0.0    0.201\nsymmetry (mean):                     0.106  0.304\nfractal dimension (mean):            0.05
.097\nradius (standard error):              0.112  2.873\ntexture (standard error):           0.36   4.885\nperimeter (standard error):
.757  21.98\narea (standard error):                6.802  542.2\nsmoothness (standard error):       0.002  0.031\ncompactness (standard
rror):          0.002  0.135\nconcavity (standard error):          0.0    0.396\nconcave points (standard error):      0.0
.053\nsymmetry (standard error):              0.008  0.079\nfractal dimension (standard error):  0.001  0.03\nradius (worst):
```

```
dataset['data'].head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0

5 rows × 30 columns

```
dataset['data'].shape
```

```
(569, 30)
```

```
dataset['target'].head()
```

	target
0	0
1	0
2	0
3	0
4	0

dtype: int64

```
dataset['target'].value_counts()
```

target	count
1	357
0	212

dtype: int64

Define explanatory and target variables:

Step 2:Split the dataset into training and testing sets

```
(1)	x = dataset['data']
y = dataset['target']
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)
```

——--

```
(2) from sklearn.preprocessing import StandardScaler
ss_train = StandardScaler()
x_train = ss_train.fit_transform(x_train)
ss_test = StandardScaler()
x_test = ss_train.transform(x_test)
```

Step 3: Normalize the data for numerical stability

```
from sklearn.preprocessing import StandardScaler
ss_train = StandardScaler()
x_train = ss_train.fit_transform(x_train)
ss_test = StandardScaler()
x_test = ss_train.transform(x_test)
```

Step 4: Fit a logistic regression model to the training data

```
from sklearn.linear_model import LogisticRegression
logistic_classifier = LogisticRegression()
logistic_classifier.fit(x_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

Step 5: Make predictions on the testing data

```
y_pred = logistic_classifier.predict(x_test)
print (y_pred[0:5])
print (y_test [0:5])
```

```
[1 0 0 1 1]
204    1
70     0
131    0
431    1
540    1
Name: target, dtype: int64
```

Step 6: Calculate the accuracy score by comparing the actual values and predicted values

We will calculate the confusion matrix to get the necessary parameters:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
TN, FP, FN, TP = confusion_matrix(y_test, y_pred).ravel()
print ('True Positive (TP) ', TP)
print ('True Negative (TN)',TN)
print ('False Positive(FP) ',FP)
print ('False Negative(FN) ',FN)
```

```
True Positive (TP)  87
True Negative (TN) 53
False Positive(FP)  1
False Negative(FN)  2
```

```
from sklearn import metrics
import matplotlib.pyplot as plt
print("Logistic Regression's Accuracy: ", metrics.accuracy_score(y_test, y_pred))
```

```
Logistic Regression's Accuracy:  0.9790209790209791
```

```
# another way to fing accuracy
accuracy=(TP + TN) / (TP + TN + FP + FN)
print("Logistic Regression's Accuracy: {:0.3f}".format(accuracy))
```

```
Logistic Regression's Accuracy: 0.979
```

Other binary classifier in the scikit_learn library

Logistic regression is just one of many classification algorithms defined in Scikit-learn

Initializing each binary classifier To quickly train each model in loop, we'll initialize each mode and store it by name in a dictionary:

```
models={}
#Logistic Regression
from sklearn.linear_model import LogisticRegression
models['Logistic Regression']=LogisticRegression()
# Support Vector Machines
from sklearn.svm import LinearSVC
models['Support Vector Machines'] =LinearSVC()
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
```

```
models['Decision Tree']=DecisionTreeClassifier()
#Random Forest
from sklearn.ensemble import RandomForestClassifier
models['Random Forest']=RandomForestClassifier()
# Navie Bayes
from sklearn.naive_bayes import GaussianNB
models['Naive Bayes']= GaussianNB()
#K-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier
models['K-Nearest Neighbor'] = KNeighborsClassifier()
models
```

```
{'Logistic Regression': LogisticRegression(),
 'Support Vector Machines': LinearSVC(),
 'Decision Tree': DecisionTreeClassifier(),
 'Random Forest': RandomForestClassifier(),
 'Naive Bayes': GaussianNB(),
 'K-Nearest Neighbor': KNeighborsClassifier()}
```

Performance evaluation of each binary classifier

```
from sklearn.metrics import accuracy_score, precision_score, recall_score
accuracy, precision, recall={},{},{}
for key in models.keys():
    #fit the classifier
    models[key].fit(x_train, y_train)
    #Make prediction
    predictions=models [key].predict(x_test)
    #Calculate Accuracy
    accuracy[key] = accuracy_score(predictions, y_test)
    precision [key] = precision_score(predictions, y_test)
    recall[key]=recall_score(predictions, y_test)
```

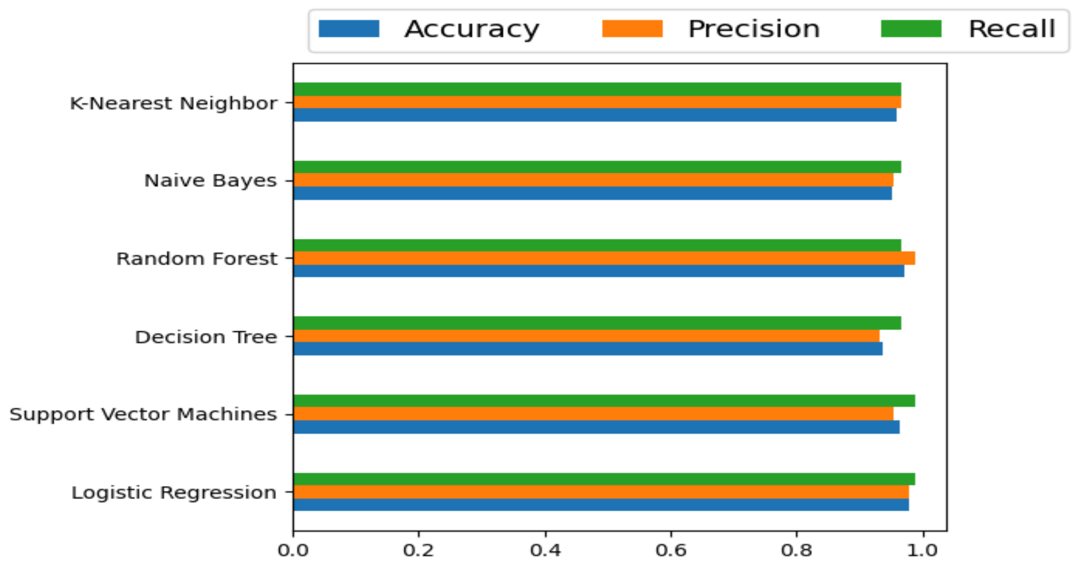
With all metrics stored, we can use pandas to view the data as a table:

```
import pandas as pd
df_model = pd.DataFrame(index=models.keys(), columns=['Accuracy', 'Precision', 'Recall'])
df_model['Accuracy']=accuracy.values()
df_model['Precision'] = precision.values()
df_model['Recall']=recall.values()
df_model
```

	Accuracy	Precision	Recall
Logistic Regression	0.979021	0.977528	0.988636
Support Vector Machines	0.965035	0.955056	0.988372
Decision Tree	0.937063	0.932584	0.965116
Random Forest	0.972028	0.988764	0.967033
Naive Bayes	0.951049	0.955056	0.965909
K-Nearest Neighbor	0.958042	0.966292	0.966292

Visualization:

```
ax = df_model.plot.barh()
ax.legend(
ncol=len(models.keys()),
bbox_to_anchor=(0, 1),
loc='lower left',
prop={'size': 14}
)
plt.tight_layout()
```



Colab Link :-
https://colab.research.google.com/drive/1IHscp16yByVJIsNSBsjLSGdT7XFvzaeV