**CHAPTER 6**

**EVALUATION DETAILS**

This project is implemented and evaluated using six distinct classification algorithms. They are Random Forest algorithm, Multi-layer perceptron algorithm.

**6.1 Sample Code**

#importing required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

from sklearn import metrics

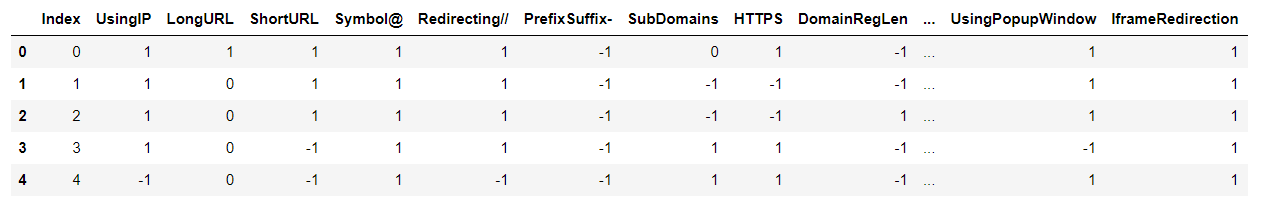
import warnings

warnings.filterwarnings('ignore')

#Loading data into dataframe

data = pd.read\_csv("phishing.csv")

data.head()



**Fig 6.1 Dataset**

#Shape of dataframe

data.shape

#Listing the features of the dataset

data.columns

#Information about the dataset

data.info()

# nunique value in columns

data.nunique()

#droping index column

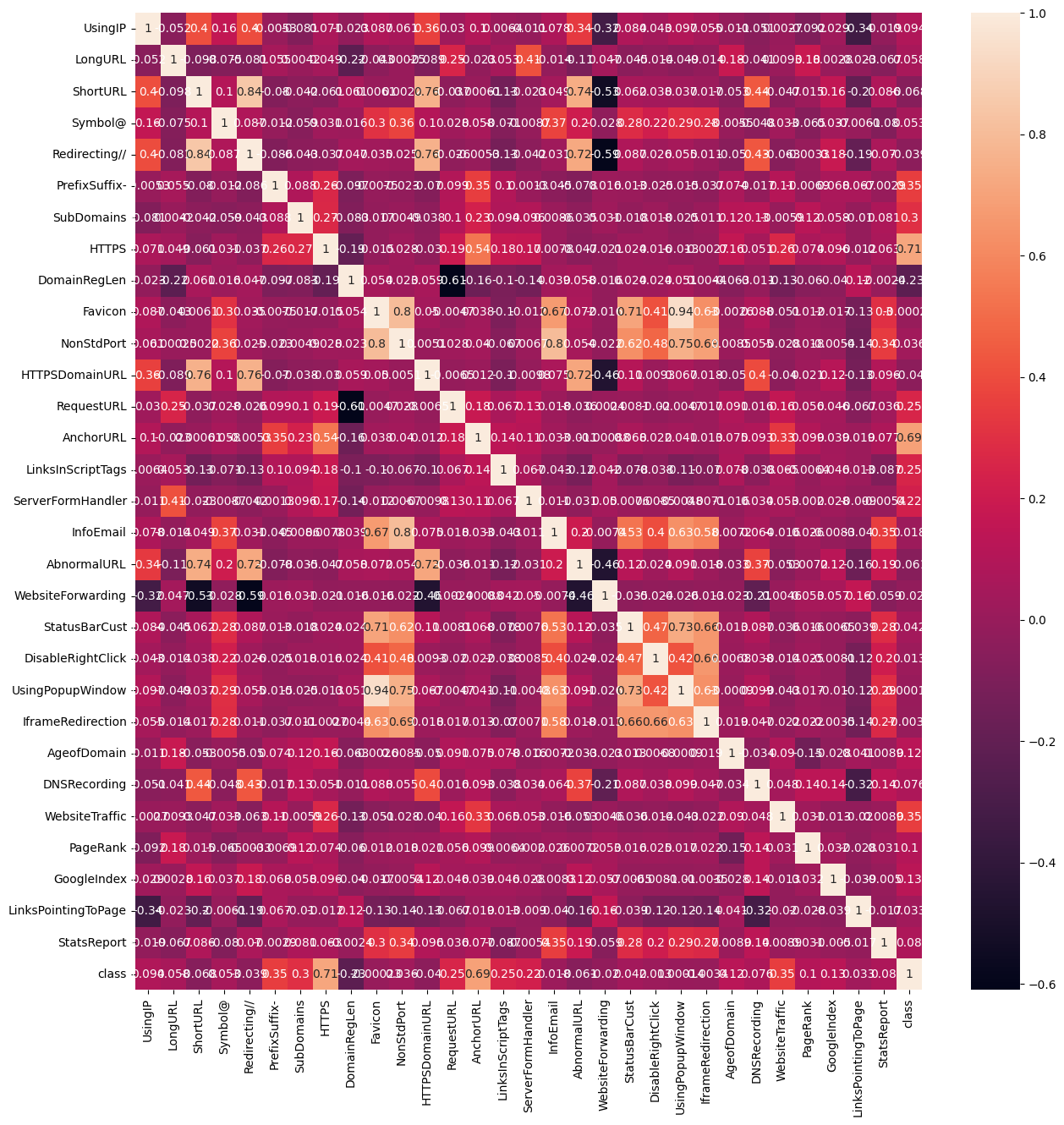
data = data.drop(['Index'],axis = 1)

#Correlation heatmap

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

sns.heatmap(data.corr(), annot=True)

plt.show()



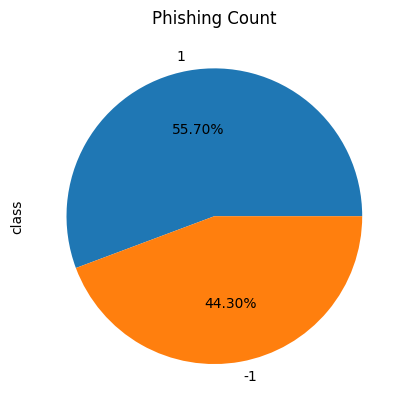
**Fig 6.2 Correlation Heatmap**

# Phishing Count in pie chart

data['class'].value\_counts().plot(kind='pie',autopct='%1.2f%%')

plt.title("Phishing Count")

plt.show()



**Fig 6.3 Data Pie Chart**

# Splitting the dataset into dependant and independant fetature

X = data.drop(["class"],axis =1)

y = data["class"]

# Splitting the dataset into train and test sets: 80-20 split

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\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape

# Creating holders to store the model performance results

ML\_Model = []

accuracy = []

f1\_score = []

recall = []

precision = []

#function to call for storing the results

def storeResults(model, a,b,c,d):

ML\_Model.append(model)

accuracy.append(round(a, 3))

f1\_score.append(round(b, 3))

recall.append(round(c, 3))

precision.append(round(d, 3))

**Random Forest**

# Random Forest Classifier Model

from sklearn.ensemble import RandomForestClassifier

# instantiate the model

forest = RandomForestClassifier(n\_estimators=100, random\_state=42)

# fit the model

forest.fit(X\_train,y\_train)

#computing the accuracy, f1\_score, Recall, precision of the model performance

acc\_train\_forest = metrics.accuracy\_score(y\_train,y\_train\_forest)

acc\_test\_forest = metrics.accuracy\_score(y\_test,y\_test\_forest)

print("Random Forest : Accuracy on training Data: {:.3f}".format(acc\_train\_forest))

print("Random Forest : Accuracy on test Data: {:.3f}".format(acc\_test\_forest))

print()

f1\_score\_train\_forest = metrics.f1\_score(y\_train,y\_train\_forest)

f1\_score\_test\_forest = metrics.f1\_score(y\_test,y\_test\_forest)

print("Random Forest : f1\_score on training Data: {:.3f}".format(f1\_score\_train\_forest))

print("Random Forest : f1\_score on test Data: {:.3f}".format(f1\_score\_test\_forest))

print()

recall\_score\_train\_forest = metrics.recall\_score(y\_train,y\_train\_forest)

recall\_score\_test\_forest = metrics.recall\_score(y\_test,y\_test\_forest)

print("Random Forest : Recall on training Data: {:.3f}".format(recall\_score\_train\_forest))

print("Random Forest : Recall on test Data: {:.3f}".format(recall\_score\_test\_forest))

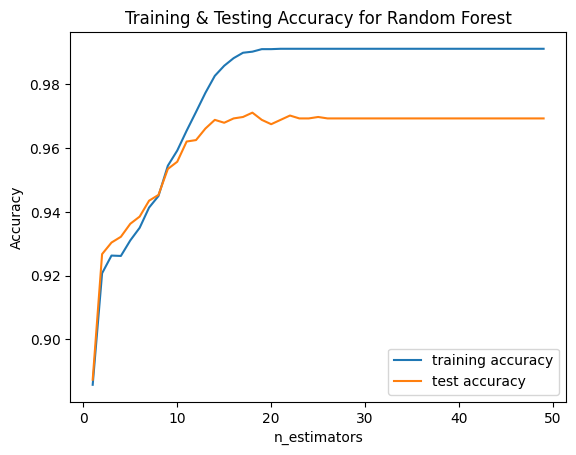
print()

precision\_score\_train\_forest = metrics.precision\_score(y\_train,y\_train\_forest)

precision\_score\_test\_forest = metrics.precision\_score(y\_test,y\_test\_forest)

print("Random Forest : precision on training Data: {:.3f}".format(precision\_score\_train\_forest))

print("Random Forest : precision on test Data: {:.3f}".format(precision\_score\_test\_forest))



**Fig 6.1: Training & Testing Accuracy for Random Forest**

**LINEAR REGRESSION**

# Linear regression model

from sklearn.linear\_model import LogisticRegression

#from sklearn.pipeline import Pipeline

# instantiate the model

log = LogisticRegression()

# fit the model

log.fit(X\_train,y\_train)

#predicting the target value from the model for the samples

y\_train\_log = log.predict(X\_train)

y\_test\_log = log.predict(X\_test)

#computing the accuracy, f1\_score, Recall, precision of the model performance

acc\_train\_log = metrics.accuracy\_score(y\_train,y\_train\_log)

acc\_test\_log = metrics.accuracy\_score(y\_test,y\_test\_log)

print("Logistic Regression : Accuracy on training Data: {:.3f}".format(acc\_train\_log))

print("Logistic Regression : Accuracy on test Data: {:.3f}".format(acc\_test\_log))

print()

f1\_score\_train\_log = metrics.f1\_score(y\_train,y\_train\_log)

f1\_score\_test\_log = metrics.f1\_score(y\_test,y\_test\_log)

print("Logistic Regression : f1\_score on training Data: {:.3f}".format(f1\_score\_train\_log))

print("Logistic Regression : f1\_score on test Data: {:.3f}".format(f1\_score\_test\_log))

print()

recall\_score\_train\_log = metrics.recall\_score(y\_train,y\_train\_log)

recall\_score\_test\_log = metrics.recall\_score(y\_test,y\_test\_log)

print("Logistic Regression : Recall on training Data: {:.3f}".format(recall\_score\_train\_log))

print("Logistic Regression : Recall on test Data: {:.3f}".format(recall\_score\_test\_log))

print()

precision\_score\_train\_log = metrics.precision\_score(y\_train,y\_train\_log)

precision\_score\_test\_log = metrics.precision\_score(y\_test,y\_test\_log)

print("Logistic Regression : precision on training Data: {:.3f}".format(precision\_score\_train\_log))

print("Logistic Regression : precision on test Data: {:.3f}".format(precision\_score\_test\_log))

#computing the classification report of the model

print(metrics.classification\_report(y\_test, y\_test\_log))

# Lists to store training and testing accuracy

training\_accuracy\_logreg = []

test\_accuracy\_logreg = []

# Try different values of the regularization parameter C

C\_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

for C in C\_values:

# Record training set accuracy

training\_accuracy\_logreg.append(log.score(X\_train, y\_train))

# Record generalization accuracy

test\_accuracy\_logreg.append(log.score(X\_test, y\_test))

# Plotting the training & testing accuracy for different values of C

plt.plot(C\_values, training\_accuracy\_logreg, label="Training accuracy")

plt.plot(C\_values, test\_accuracy\_logreg, label="Testing accuracy")

plt.xscale('log') # Using a logarithmic scale for better visualization

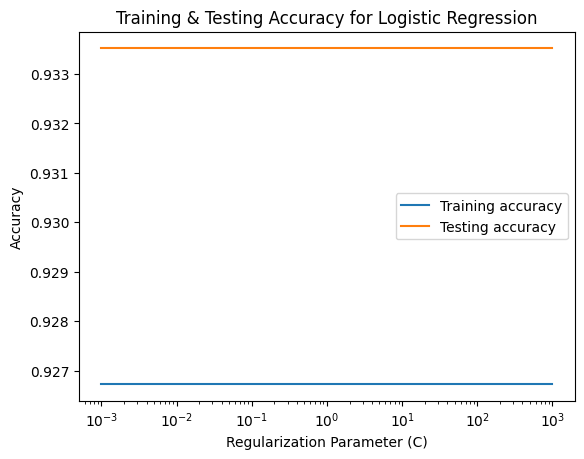
plt.xlabel("Regularization Parameter (C)")

plt.ylabel("Accuracy")

plt.title("Training & Testing Accuracy for Logistic Regression")

plt.legend()

plt.show()



**Fig 6.2: Training & Testing Accuracy for Logistic Regression**

#storing the results. The below mentioned order of parameter passing is important.

storeResults('Logistic Regression',acc\_test\_log,f1\_score\_test\_log,

recall\_score\_train\_log,precision\_score\_train\_log)

**DECISION TREE CLASSIFIER**

# Decision Tree Classifier model

from sklearn.tree import DecisionTreeClassifier

# instantiate the model

tree = DecisionTreeClassifier(max\_depth=30)

# fit the model

tree.fit(X\_train, y\_train)

#predicting the target value from the model for the samples

y\_train\_tree = tree.predict(X\_train)

y\_test\_tree = tree.predict(X\_test)

#computing the accuracy, f1\_score, Recall, precision of the model performance

acc\_train\_tree = metrics.accuracy\_score(y\_train,y\_train\_tree)

acc\_test\_tree = metrics.accuracy\_score(y\_test,y\_test\_tree)

print("Decision Tree : Accuracy on training Data: {:.3f}".format(acc\_train\_tree))

print("Decision Tree : Accuracy on test Data: {:.3f}".format(acc\_test\_tree))

print()

f1\_score\_train\_tree = metrics.f1\_score(y\_train,y\_train\_tree)

f1\_score\_test\_tree = metrics.f1\_score(y\_test,y\_test\_tree)

print("Decision Tree : f1\_score on training Data: {:.3f}".format(f1\_score\_train\_tree))

print("Decision Tree : f1\_score on test Data: {:.3f}".format(f1\_score\_test\_tree))

print()

recall\_score\_train\_tree = metrics.recall\_score(y\_train,y\_train\_tree)

recall\_score\_test\_tree = metrics.recall\_score(y\_test,y\_test\_tree)

print("Decision Tree : Recall on training Data: {:.3f}".format(recall\_score\_train\_tree))

print("Decision Tree : Recall on test Data: {:.3f}".format(recall\_score\_test\_tree))

print()

precision\_score\_train\_tree = metrics.precision\_score(y\_train,y\_train\_tree)

precision\_score\_test\_tree = metrics.precision\_score(y\_test,y\_test\_tree)

print("Decision Tree : precision on training Data: {:.3f}".format(precision\_score\_train\_tree))

print("Decision Tree : precision on test Data: {:.3f}".format(precision\_score\_test\_tree))

#computing the classification report of the model

print(metrics.classification\_report(y\_test, y\_test\_tree))

training\_accuracy = []

test\_accuracy = []

# try max\_depth from 1 to 30

depth = range(1,30)

for n in depth:

tree\_test = DecisionTreeClassifier(max\_depth=n)

tree\_test.fit(X\_train, y\_train)

# record training set accuracy

training\_accuracy.append(tree\_test.score(X\_train, y\_train))

# record generalization accuracy

test\_accuracy.append(tree\_test.score(X\_test, y\_test))

#plotting the training & testing accuracy for max\_depth from 1 to 30

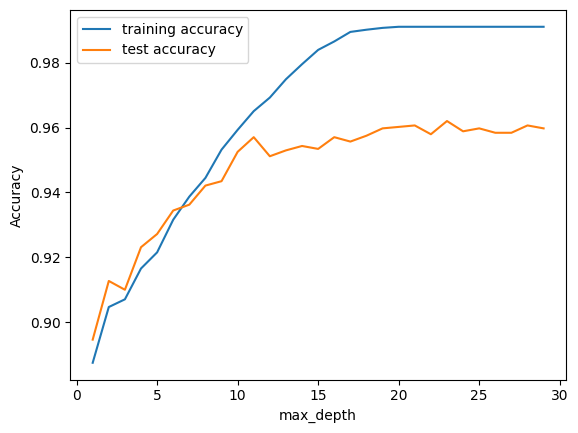
plt.plot(depth, training\_accuracy, label="training accuracy")

plt.plot(depth, test\_accuracy, label="test accuracy")

plt.ylabel("Accuracy")

plt.xlabel("max\_depth")

plt.legend();



**Fig 6.3: Training & Testing accuracy for Decision Tree**

#storing the results. The below mentioned order of parameter passing is important.

storeResults('Decision Tree',acc\_test\_tree,f1\_score\_test\_tree,

recall\_score\_train\_tree,precision\_score\_train\_tree)

**GAUSSIAN NAÏVE BAYES**

# Gaussian Naive Bayes Classifier model

from sklearn.naive\_bayes import GaussianNB

# instantiate the model

nb = GaussianNB()

# fit the model

nb.fit(X\_train, y\_train)

# predicting the target value from the model for the samples

y\_train\_nb = nb.predict(X\_train)

y\_test\_nb = nb.predict(X\_test)

# computing the accuracy, f1\_score, Recall, precision of the model performance

acc\_train\_nb = metrics.accuracy\_score(y\_train, y\_train\_nb)

acc\_test\_nb = metrics.accuracy\_score(y\_test, y\_test\_nb)

print("Gaussian Naive Bayes: Accuracy on training Data: {:.3f}".format(acc\_train\_nb))

print("Gaussian Naive Bayes: Accuracy on test Data: {:.3f}".format(acc\_test\_nb))

print()

f1\_score\_train\_nb = metrics.f1\_score(y\_train, y\_train\_nb)

f1\_score\_test\_nb = metrics.f1\_score(y\_test, y\_test\_nb)

print("Gaussian Naive Bayes: f1\_score on training Data: {:.3f}".format(f1\_score\_train\_nb))

print("Gaussian Naive Bayes: f1\_score on test Data: {:.3f}".format(f1\_score\_test\_nb))

print()

recall\_score\_train\_nb = metrics.recall\_score(y\_train, y\_train\_nb)

recall\_score\_test\_nb = metrics.recall\_score(y\_test, y\_test\_nb)

print("Gaussian Naive Bayes: Recall on training Data: {:.3f}".format(recall\_score\_train\_nb))

print("Gaussian Naive Bayes: Recall on test Data: {:.3f}".format(recall\_score\_test\_nb))

print()

precision\_score\_train\_nb = metrics.precision\_score(y\_train, y\_train\_nb)

precision\_score\_test\_nb = metrics.precision\_score(y\_test, y\_test\_nb)

print("Gaussian Naive Bayes: precision on training Data: {:.3f}".format(precision\_score\_train\_nb))

print("Gaussian Naive Bayes: precision on test Data: {:.3f}".format(precision\_score\_test\_nb))

# computing the classification report of the model

print(metrics.classification\_report(y\_test, y\_test\_nb))

training\_accuracy\_nb = []

test\_accuracy\_nb = []

# try different priors

priors = [None, [0.1, 0.9], [0.2, 0.8], [0.3, 0.7], [0.4, 0.6], [0.5, 0.5]]

for prior in priors:

nb\_test = GaussianNB(priors=prior)

nb\_test.fit(X\_train, y\_train)

# record training set accuracy

training\_accuracy\_nb.append(nb\_test.score(X\_train, y\_train))

# record generalization accuracy

test\_accuracy\_nb.append(nb\_test.score(X\_test, y\_test))

# plotting the training & testing accuracy for different priors

plt.plot(range(len(priors)), training\_accuracy\_nb, label="training accuracy")

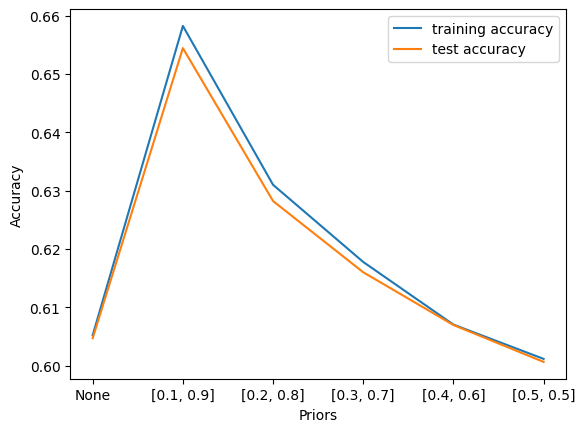
plt.plot(range(len(priors)), test\_accuracy\_nb, label="test accuracy")

plt.xticks(range(len(priors)), [str(prior) for prior in priors])

plt.ylabel("Accuracy")

plt.xlabel("Priors")

plt.legend();



**Fig 6.4: Training & Testing accuracy for Gaussian Naïve Bayes**

# storing the results. The below mentioned order of parameter passing is important.

storeResults('Gaussian Naive Bayes', acc\_test\_nb, f1\_score\_test\_nb,

recall\_score\_train\_nb, precision\_score\_train\_nb)

**MULTI-LINEAR PERCEPTION**

# Multi-layer Perceptron Classifier Model

from sklearn.neural\_network import MLPClassifier

# instantiate the model

mlp = MLPClassifier()

#mlp = GridSearchCV(mlpc, parameter\_space)

# fit the model

mlp.fit(X\_train,y\_train)

#computing the accuracy, f1\_score, Recall, precision of the model performance

acc\_train\_mlp = metrics.accuracy\_score(y\_train,y\_train\_mlp)

acc\_test\_mlp = metrics.accuracy\_score(y\_test,y\_test\_mlp)

print("Multi-layer Perceptron : Accuracy on training Data: {:.3f}".format(acc\_train\_mlp))

print("Multi-layer Perceptron : Accuracy on test Data: {:.3f}".format(acc\_test\_mlp))

print()

f1\_score\_train\_mlp = metrics.f1\_score(y\_train,y\_train\_mlp)

f1\_score\_test\_mlp = metrics.f1\_score(y\_test,y\_test\_mlp)

print("Multi-layer Perceptron : f1\_score on training Data: {:.3f}".format(f1\_score\_train\_mlp))

print("Multi-layer Perceptron : f1\_score on test Data: {:.3f}".format(f1\_score\_train\_mlp))

print()

recall\_score\_train\_mlp = metrics.recall\_score(y\_train,y\_train\_mlp)

recall\_score\_test\_mlp = metrics.recall\_score(y\_test,y\_test\_mlp)

print("Multi-layer Perceptron : Recall on training Data: {:.3f}".format(recall\_score\_train\_mlp))

print("Multi-layer Perceptron : Recall on test Data: {:.3f}".format(recall\_score\_test\_mlp))

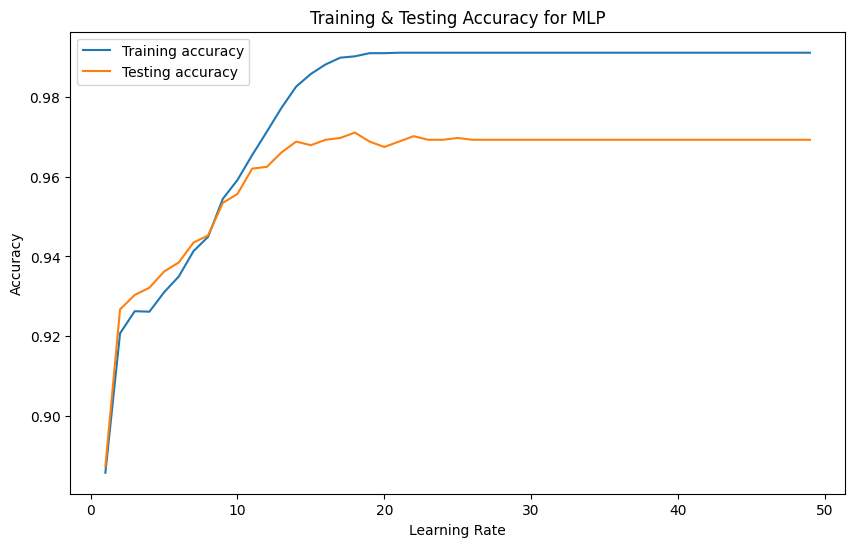
print()

precision\_score\_train\_mlp = metrics.precision\_score(y\_train,y\_train\_mlp)

precision\_score\_test\_mlp = metrics.precision\_score(y\_test,y\_test\_mlp)

print("Multi-layer Perceptron : precision on training Data: {:.3f}".format(precision\_score\_train\_mlp))

print("Multi-layer Perceptron : precision on test Data: {:.3f}".format(precision\_score\_test\_mlp))



**Fig 6.5: Training & Testing Accuracy for MLP**

#storing the results. The below mentioned order of parameter passing is important.

storeResults('Multi-layer Perceptron',acc\_test\_mlp,f1\_score\_test\_mlp,

recall\_score\_train\_mlp,precision\_score\_train\_mlp)

**#creating dataframe**

result = pd.DataFrame({ 'ML Model' : ML\_Model,

'Accuracy' : accuracy,

'f1\_score' : f1\_score,

'Recall' : recall,

'Precision': precision,

})

#Sorting the datafram on accuracy

sorted\_result=result.sort\_values(by=['Accuracy', 'f1\_score'],ascending=False).reset\_index(drop=True)

ML\_Model = ['Random Forest', 'LogisticRegression','DecisionTreeClassifier ','Gaussian Naive Bayes', 'Multi-layer Perceptron']

accuracy = [acc\_test\_forest, acc\_test\_log, acc\_test\_tree, acc\_test\_nb, acc\_test\_mlp]

# Creating a DataFrame

results = pd.DataFrame({

'ML Model': ML\_Model,

'Accuracy': accuracy

})

**# Sorting the DataFrame based on Accuracy**

sorted\_results=results.sort\_values(by=['Accuracy'],ascending=False).reset\_index(drop=True)

**# Plotting the bar chart**

plt.figure(figsize=(10, 6))

bars = plt.bar(sorted\_result['ML Model'], sorted\_result['Accuracy'], color=['skyblue', 'lightgreen', 'lightcoral', 'lightsalmon'])

plt.title('Testing Accuracy Comparison of ML Models')

plt.xlabel('ML Model')

plt.ylabel('Testing Accuracy')

plt.ylim(0, 1) # Set the y-axis limit between 0 and 1 for accuracy values

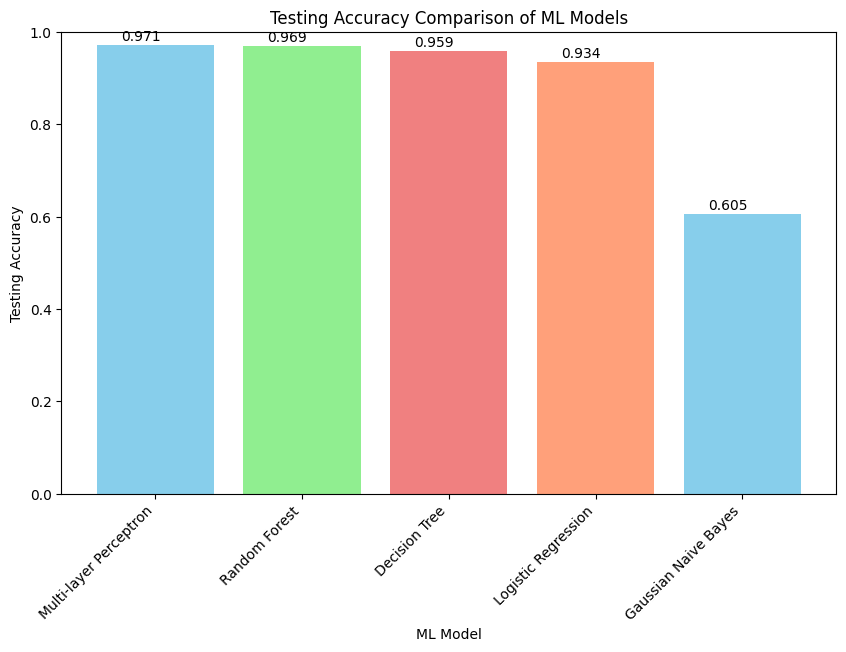
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability

**# Displaying precise values on top of each bar**

for bar, value in zip(bars, sorted\_result['Accuracy']):

plt.text(bar.get\_x() + bar.get\_width() / 2 - 0.1, bar.get\_height() + 0.01, f'{value:.3f}', ha='center', color='black')

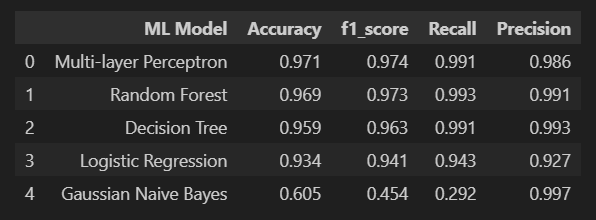
plt.show()



**Fig 6.6: Testing Accuracy Comparison of ML Models**

# dispalying total result

sorted\_result



**Fig 6.7: Various ML Models Results**