**CORE FUNCTIONALITIES OF PHASE 2**

* In the following modules one the data is loaded, and the packages and the libraries are imported the functions like read(), show(), plot() will read the data, visualize the data and the data is plotted in forms of the graph respectively.
* The csv file is read, and seaborn library is used to plot the graphs. The train and test data split is imported.
* True positives, True Negatives, False Positives , False Negatives will be obtained.
* The dataset is displayed as data graph using the obtained TP, TN, FP, FN and accuracy results are displayed with a heat map

Algorithm Implementation

Train Data/Test Data

Read Data

Import Packages

**Module – 3**

# **Performance measurements of Logistic regression**

*#import library packages*

**import** pandas **as** p

**import** matplotlib.pyplot **as** plt

**import** numpy **as** n

**import** seaborn **as** sns

data **=** p**.**read\_csv("demo.csv")

**import** warnings

warnings**.**filterwarnings('ignore')

data**.**head(5)

data**.**tail()

data["flood"]**.**value\_counts()

data**.**isnull()**.**sum()

data**.**duplicated()**.**sum()

data**.**columns

data**.**shape

**del** data['JAN']

**del** data['Unnamed: 0']

**del** data['FEB']

**del** data['MAR']

**del** data['APR']

**del** data['MAY']

**del** data['JUN']

**del** data['JUL']

**del** data['AUG']

**del** data['SEP']

**del** data['OCT']

**del** data['NOV']

**del** data['DEC']

**del** data['Avg\_june10days']

**del** data['maytojune']

data**.**head()

**from** sklearn.preprocessing **import** LabelEncoder

var\_mod**=**["STATE\_UT\_NAME"]

le **=** LabelEncoder()

**for** i **in** var\_mod:

data[i] **=** le**.**fit\_transform(data[i])**.**astype(int)

data**.**head()

**from** sklearn.metrics **import** confusion\_matrix, classification\_report, matthews\_corrcoef, cohen\_kappa\_score, accuracy\_score, average\_precision\_score, roc\_auc\_score

X **=** data**.**drop(labels**=**'flood', axis**=**1)

*#Response variable*

y **=** data**.**loc[:,'flood']

*#We'll use a test size of 30%. We also stratify the split on the response variable, which is very important to do because there are so few fraudulent transactions.*

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.3, random\_state**=**1, stratify**=**y)

Logistic regression

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.model\_selection **import** cross\_val\_score

logR**=** LogisticRegression()

logR**.**fit(X\_train,y\_train)

predictLR **=** logR**.**predict(X\_test)

print("")

print('Classification report of Logistic Regression Results:')

print("")

print(classification\_report(y\_test,predictLR))

x **=** (accuracy\_score(y\_test,predictLR)**\***100)

print('Accuracy result of Logisticregression is:', x)

print("")

cm2**=**confusion\_matrix(y\_test,predictLR)

print('Confusion Matrix result of Logistic Regression is:\n',cm2)

print("")

sensitivity2 **=** cm2[0,0]**/**(cm2[0,0]**+**cm2[0,1])

print('Sensitivity : ', sensitivity2 )

print("")

specificity2 **=** cm2[1,1]**/**(cm2[1,0]**+**cm2[1,1])

print('Specificity : ', specificity2)

print("")

accuracy **=** cross\_val\_score(logR, X, y, scoring**=**'accuracy')

print('Cross validation test results of accuracy:')

print(accuracy)

*#get the mean of each fold*

print("")

print("Accuracy result of Logistic Regression is:",accuracy**.**mean() **\*** 100)

LR**=**accuracy**.**mean() **\*** 100

**def** graph():

**import** matplotlib.pyplot **as** plt

data**=**[LR]

alg**=**"Logistic Regression"

plt**.**figure(figsize**=**(5,5))

b**=**plt**.**bar(alg,data,color**=**("b"))

plt**.**title("Accuracy comparison Flood",fontsize**=**15)

plt**.**legend(b,data,fontsize**=**9)

graph()

TN **=** cm2[1][0]

FN **=** cm2[0][0]

TP **=** cm2[1][1]

FP **=** cm2[0][1]

print("True Positive :",TP)

print("True Negative :",TN)

print("False Positive :",FP)

print("False Negative :",FN)

print("")

TPR **=** TP**/**(TP**+**FN)

TNR **=** TN**/**(TN**+**FP)

FPR **=** FP**/**(FP**+**TN)

FNR **=** FN**/**(TP**+**FN)

print("True Positive Rate :",TPR)

print("True Negative Rate :",TNR)

print("False Positive Rate :",FPR)

print("False Negative Rate :",FNR)

print("")

PPV **=** TP**/**(TP**+**FP)

NPV **=** TN**/**(TN**+**FN)

print("Positive Predictive Value :",PPV)

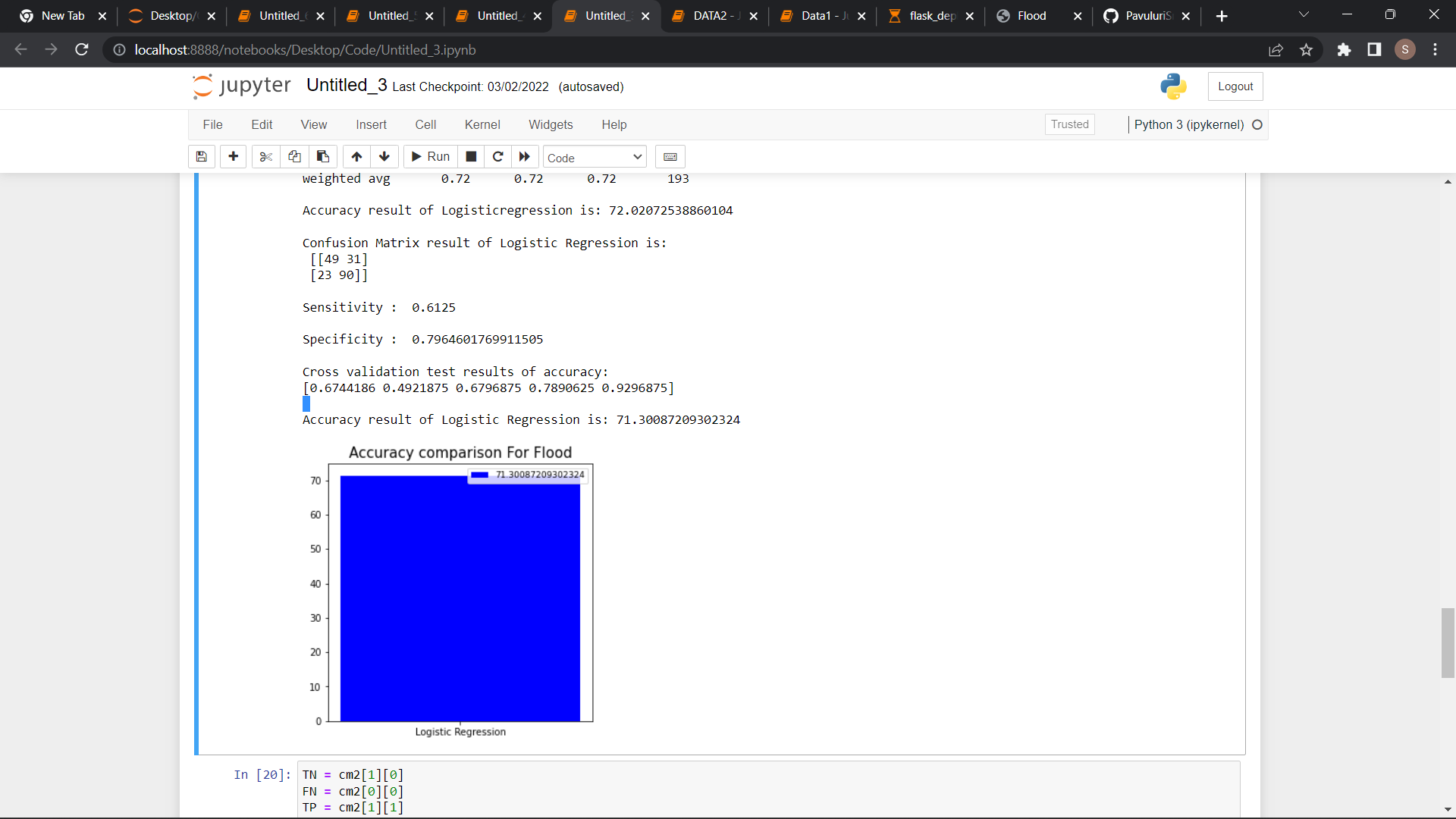
print("Negative predictive value :",NPV)

cm2**=**confusion\_matrix(y\_test, predictLR)

print(cm2)

sns**.**heatmap(cm2**/**n**.**sum(cm2), annot**=True**, cmap **=** 'Blues', annot\_kws**=**{"size": 16}, fmt**=**'.2%',)

plt.show()



**Module – 4**

**Performance measurements of Decision Tree:**

*#import library packages*

**import** pandas **as** p

**import** matplotlib.pyplot **as** plt

**import** numpy **as** n

**import** seaborn **as** sns

data **=** p**.**read\_csv("demo.csv")

**import** warnings

warnings**.**filterwarnings('ignore')

data**.**head(5)

data**.**tail()

data["flood"]**.**value\_counts()

data**.**isnull()**.**sum()

data**.**duplicated()**.**sum()

data**.**columns

data**.**shape

**del** data['JAN']

**del** data['Unnamed: 0']

**del** data['FEB']

**del** data['MAR']

**del** data['APR']

**del** data['MAY']

**del** data['JUN']

**del** data['JUL']

**del** data['AUG']

**del** data['SEP']

**del** data['OCT']

**del** data['NOV']

**del** data['DEC']

**del** data['Avg\_june10days']

**del** data['maytojune']

data**.**head()

**from** sklearn.preprocessing **import** LabelEncoder

var\_mod**=**["STATE\_UT\_NAME"]

le **=** LabelEncoder()

**for** i **in** var\_mod:

data[i] **=** le**.**fit\_transform(data[i])**.**astype(int)

data**.**head()

*#According to the cross-validated MCC scores, the random forest is the best-performing model, so now let's evaluate its performance on the test set.*

**from** sklearn.metrics **import** confusion\_matrix, classification\_report, matthews\_corrcoef, cohen\_kappa\_score, accuracy\_score, average\_precision\_score, roc\_auc\_score

X **=** data**.**drop(labels**=**'flood', axis**=**1)

*#Response variable*

y **=** data**.**loc[:,'flood']

*#We'll use a test size of 30%. We also stratify the split on the response variable, which is very important to do because there are so few fraudulent transactions.*

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.3, random\_state**=**1, stratify**=**y)

DecisionTree:

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.model\_selection **import** cross\_val\_score

DT**=**DecisionTreeClassifier()

DT**.**fit(X\_train,y\_train)

predictDT **=** DT**.**predict(X\_test)

print("")

print('Classification report DecisionTree classifier Results:')

print("")

print(classification\_report(y\_test,predictDT))

print("")

x **=** (accuracy\_score(y\_test,predictDT)**\***100)

print('Accuracy result of DecisionTree is:', x)

print("")

cm2**=**confusion\_matrix(y\_test,predictDT)

print('Confusion Matrix result of DecissionTree Classifier is:\n',cm2)

print("")

sensitivity2 **=** cm2[0,0]**/**(cm2[0,0]**+**cm2[0,1])

print('Sensitivity : ', sensitivity2 )

print("")

specificity2 **=** cm2[1,1]**/**(cm2[1,0]**+**cm2[1,1])

print('Specificity : ', specificity2)

print("")

accuracy **=** cross\_val\_score(DT, X, y, scoring**=**'accuracy')

print('Cross validation test results of accuracy:')

print(accuracy)

*#get the mean of each fold*

print("")

print("Accuracy result of DecisionTree Classifier is:",accuracy**.**mean() **\*** 100)

dt**=**accuracy**.**mean() **\*** 100

**def** graph():

**import** matplotlib.pyplot **as** plt

data**=**[dt]

alg**=**"Decision Tree"

plt**.**figure(figsize**=**(5,5))

b**=**plt**.**bar(alg,data,color**=**("b"))

plt**.**title("Accuracy comparison of Flood",fontsize**=**15)

plt**.**legend(b,data,fontsize**=**9)

graph()

TN **=** cm2[1][0]

FN **=** cm2[0][0]

TP **=** cm2[1][1]

FP **=** cm2[0][1]

print("True Positive :",TP)

print("True Negative :",TN)

print("False Positive :",FP)

print("False Negative :",FN)

print("")

TPR **=** TP**/**(TP**+**FN)

TNR **=** TN**/**(TN**+**FP)

FPR **=** FP**/**(FP**+**TN)

FNR **=** FN**/**(TP**+**FN)

print("True Positive Rate :",TPR)

print("True Negative Rate :",TNR)

print("False Positive Rate :",FPR)

print("False Negative Rate :",FNR)

print("")

PPV **=** TP**/**(TP**+**FP)

NPV **=** TN**/**(TN**+**FN)

print("Positive Predictive Value :",PPV)

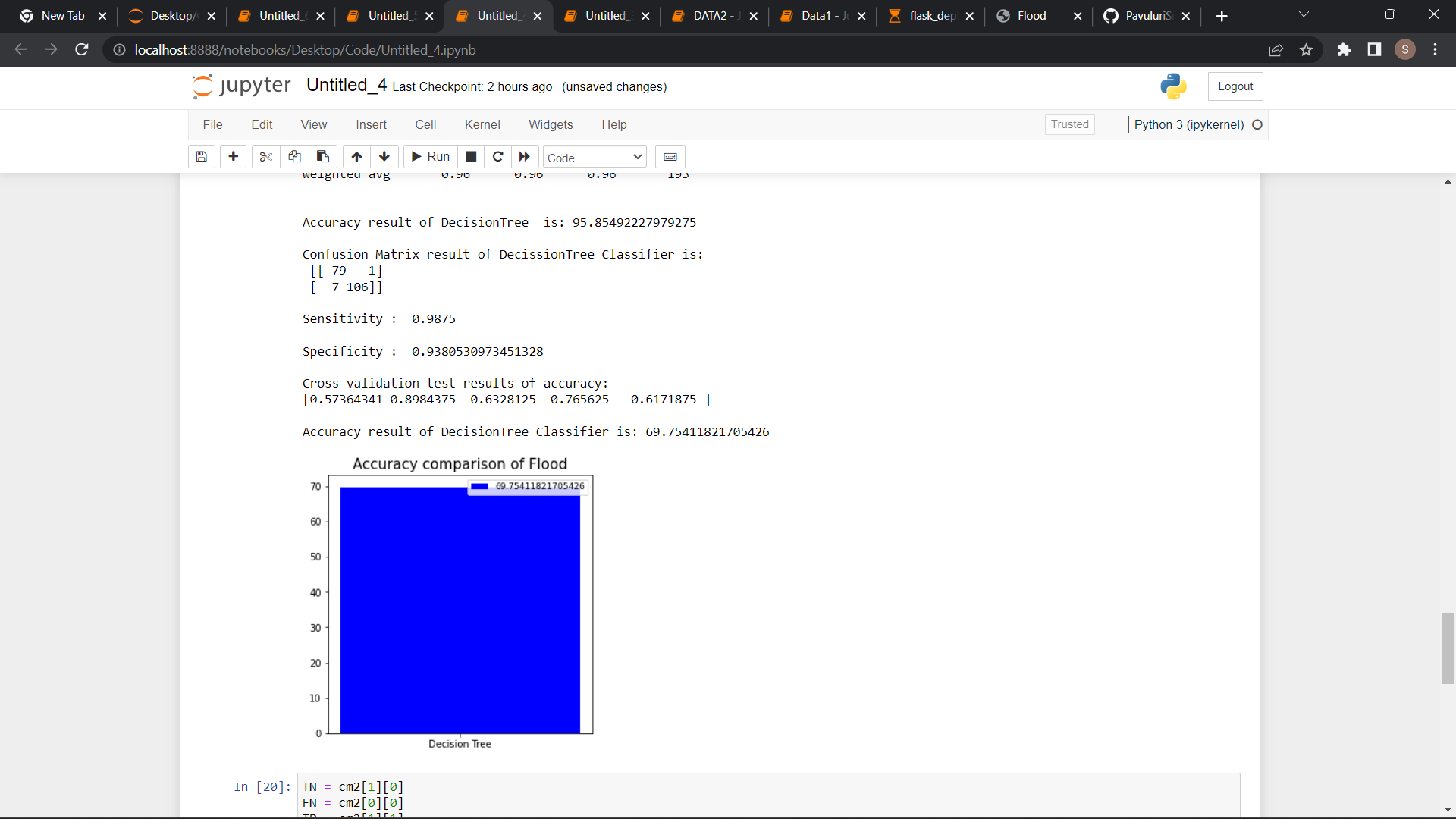
print("Negative predictive value :",NPV)

cm2**=**confusion\_matrix(y\_test, predictDT)

print(cm2)

sns**.**heatmap(cm2**/**n**.**sum(cm2), annot**=True**, cmap **=** 'Blues', annot\_kws**=**{"size": 16}, fmt**=**'.2%',)

plt**.**show()



**Module – 5**

**Performance Measurement of Random Forest**

*#import library packages*

**import** pandas **as** p

**import** matplotlib.pyplot **as** plt

**import** numpy **as** n

**import** seaborn **as** sns

data **=** p**.**read\_csv("demo.csv")

**import** warnings

warnings**.**filterwarnings('ignore')

data**.**head(5)

data**.**tail()

data["flood"]**.**value\_counts()

data**.**isnull()**.**sum()

data**.**duplicated()**.**sum()

data**.**columns

data**.**shape

**del** data['JAN']

**del** data['Unnamed: 0']

**del** data['FEB']

**del** data['MAR']

**del** data['APR']

**del** data['MAY']

**del** data['JUN']

**del** data['JUL']

**del** data['AUG']

**del** data['SEP']

**del** data['OCT']

**del** data['NOV']

**del** data['DEC']

**del** data['Avg\_june10days']

**del** data['maytojune']

data**.**head()

**from** sklearn.preprocessing **import** LabelEncoder

var\_mod**=**["STATE\_UT\_NAME"]

le **=** LabelEncoder()

**for** i **in** var\_mod:

data[i] **=** le**.**fit\_transform(data[i])**.**astype(int)

data**.**head()

*#According to the cross-validated MCC scores, the random forest is the best-performing model, so now let's evaluate its performance on the test set.*

**from** sklearn.metrics **import** confusion\_matrix, classification\_report, matthews\_corrcoef, cohen\_kappa\_score, accuracy\_score, average\_precision\_score, roc\_auc\_score

X **=** data**.**drop(labels**=**'flood', axis**=**1)

*#Response variable*

y **=** data**.**loc[:,'flood']

*#We'll use a test size of 30%. We also stratify the split on the response variable, which is very important to do because there are so few fraudulent transactions.*

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.3, random\_state**=**1, stratify**=**y)

Random Forest:

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix

**from** sklearn.model\_selection **import** cross\_val\_score

rfc **=** RandomForestClassifier()

rfc**.**fit(X\_train,y\_train)

predictR **=** rfc**.**predict(X\_test)

print("")

print('Classification report of Random Forest Results:')

print("")

print(classification\_report(y\_test,predictR))

x **=** (accuracy\_score(y\_test,predictR)**\***100)

print('Accuracy result of Random Forest is:', x)

print("")

cm1**=**confusion\_matrix(y\_test,predictR)

print('Confusion Matrix result of Random Forest is:\n',cm1)

print("")

sensitivity1 **=** cm1[0,0]**/**(cm1[0,0]**+**cm1[0,1])

print('Sensitivity : ', sensitivity1 )

print("")

specificity1 **=** cm1[1,1]**/**(cm1[1,0]**+**cm1[1,1])

print('Specificity : ', specificity1)

print("")

accuracy **=** cross\_val\_score(rfc, X, y, scoring**=**'accuracy')

print('Cross validation test results of accuracy:')

print(accuracy)

*#get the mean of each fold*

print("")

print("Accuracy result of Random Forest is:",accuracy**.**mean() **\*** 100)

RFC**=**accuracy**.**mean() **\*** 100

**def** graph():

**import** matplotlib.pyplot **as** plt

data**=**[RFC]

alg**=**"Random orest"

plt**.**figure(figsize**=**(5,5))

b**=**plt**.**bar(alg,data,color**=**("b"))

plt**.**title("Accuracy comparison of Flood")

plt**.**legend(b,data,fontsize**=**9)

graph()

TN **=** cm1[1][0]

FN **=** cm1[0][0]

TP **=** cm1[1][1]

FP **=** cm1[0][1]

print("True Positive :",TP)

print("True Negative :",TN)

print("False Positive :",FP)

print("False Negative :",FN)

print("")

TPR **=** TP**/**(TP**+**FN)

TNR **=** TN**/**(TN**+**FP)

FPR **=** FP**/**(FP**+**TN)

FNR **=** FN**/**(TP**+**FN)

print("True Positive Rate :",TPR)

print("True Negative Rate :",TNR)

print("False Positive Rate :",FPR)

print("False Negative Rate :",FNR)

print("")

PPV **=** TP**/**(TP**+**FP)

NPV **=** TN**/**(TN**+**FN)

print("Positive Predictive Value :",PPV)

print("Negative predictive value :",NPV)

cm2**=**confusion\_matrix(y\_test, predictR)

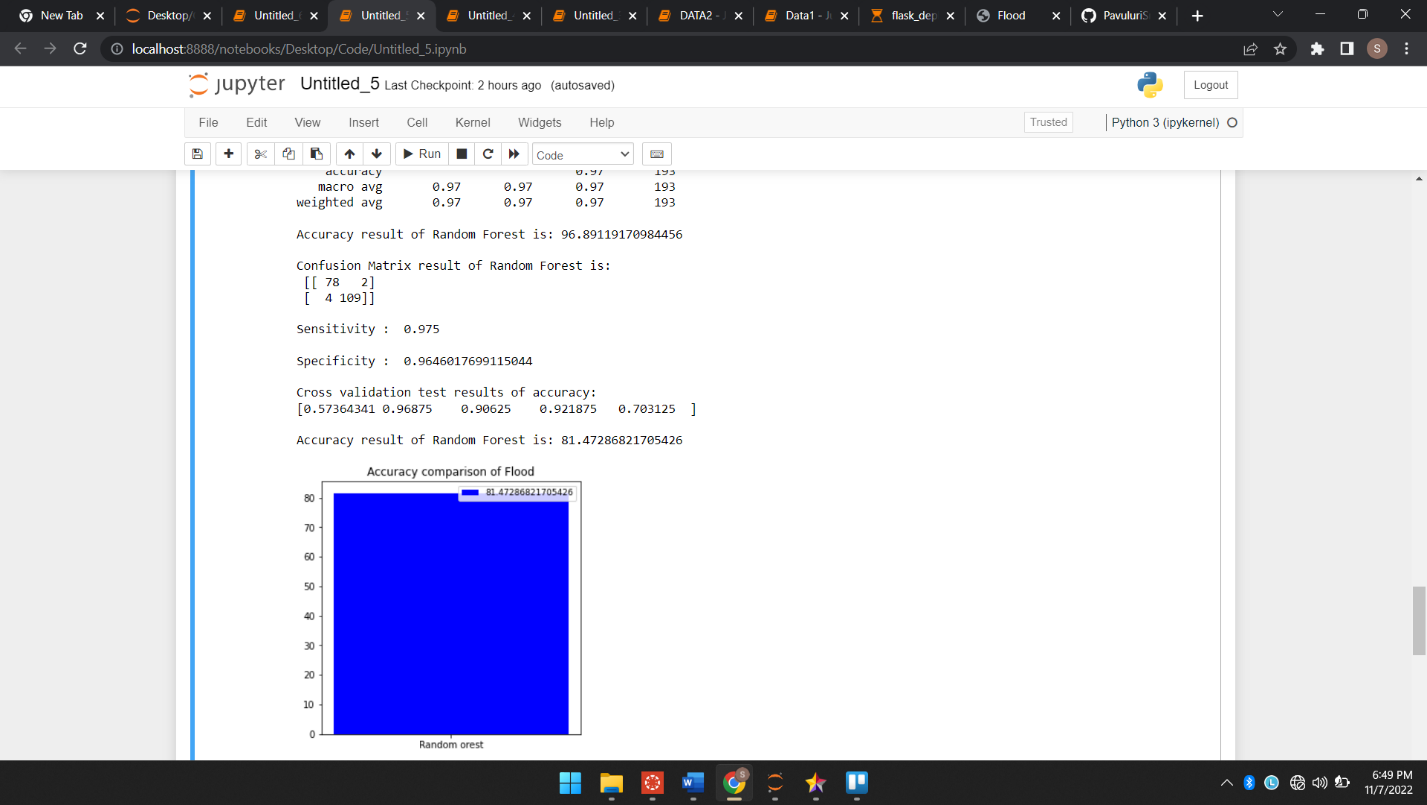
print(cm2)

sns**.**heatmap(cm2**/**n**.**sum(cm2), annot**=True**, cmap **=** 'Blues', annot\_kws**=**{"size": 16}, fmt**=**'.2%',)

plt**.**show()

**import** joblib

joblib**.**dump(rfc, 'flood.pkl')



**Module – 6**

**Performance measurement of SVM**

*#import library packages*

**import** pandas **as** p

**import** matplotlib.pyplot **as** plt

**import** numpy **as** n

**import** seaborn **as** sns

data **=** p**.**read\_csv("demo.csv")

**import** warnings

warnings**.**filterwarnings('ignore')

data**.**head(5)

data**.**tail()

data["flood"]**.**value\_counts()

data**.**isnull()**.**sum()

data**.**duplicated()**.**sum()

data**.**columns

data**.**shape

**del** data['JAN']

**del** data['Unnamed: 0']

**del** data['FEB']

**del** data['MAR']

**del** data['APR']

**del** data['MAY']

**del** data['JUN']

**del** data['JUL']

**del** data['AUG']

**del** data['SEP']

**del** data['OCT']

**del** data['NOV']

**del** data['DEC']

**del** data['Avg\_june10days']

**del** data['maytojune']

data**.**head()

**from** sklearn.preprocessing **import** LabelEncoder

var\_mod**=**["STATE\_UT\_NAME"]

le **=** LabelEncoder()

**for** i **in** var\_mod:

data[i] **=** le**.**fit\_transform(data[i])**.**astype(int)

data**.**head()

**from** sklearn.metrics **import** confusion\_matrix, classification\_report, matthews\_corrcoef, cohen\_kappa\_score, accuracy\_score, average\_precision\_score, roc\_auc\_score

X **=** data**.**drop(labels**=**'flood', axis**=**1)

*#Response variable*

y **=** data**.**loc[:,'flood']

*#We'll use a test size of 30%. We also stratify the split on the response variable, which is very important to do because there are so few fraudulent transactions.*

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.3, random\_state**=**1, stratify**=**y)

Support Vector Machines:

**from** sklearn.svm **import** SVC

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix

**from** sklearn.model\_selection **import** cross\_val\_score

s **=** SVC()

s**.**fit(X\_train,y\_train)

predicts **=** s**.**predict(X\_test)

print("")

print('Classification report of Support Vector Machines Results:')

print("")

print(classification\_report(y\_test,predicts))

x **=** (accuracy\_score(y\_test,predicts)**\***100)

print('Accuracy result of Support Vector Machines is:', x)

print("")

cm2**=**confusion\_matrix(y\_test,predicts)

print('Confusion Matrix result of Support Vector Machines is:\n',cm2)

print("")

sensitivity1 **=** cm2[0,0]**/**(cm2[0,0]**+**cm2[0,1])

print('Sensitivity : ', sensitivity1 )

print("")

specificity1 **=** cm2[1,1]**/**(cm2[1,0]**+**cm2[1,1])

print('Specificity : ', specificity1)

print("")

accuracy **=** cross\_val\_score(s, X, y, scoring**=**'accuracy')

print('Cross validation test results of accuracy:')

print(accuracy)

*#get the mean of each fold*

print("")

print("Accuracy result of Support Vector Machine is:",accuracy**.**mean() **\*** 100)

S**=**accuracy**.**mean() **\*** 100

**def** graph():

**import** matplotlib.pyplot **as** plt

data**=**[S]

alg**=**"Support Vector Machine"

plt**.**figure(figsize**=**(5,5))

b**=**plt**.**bar(alg,data,color**=**("b"))

plt**.**title("Accuracy comparison of Flood")

plt**.**legend(b,data,fontsize**=**9)

graph()

TN **=** cm2[1][0]

FN **=** cm2[0][0]

TP **=** cm2[1][1]

FP **=** cm2[0][1]

print("True Positive :",TP)

print("True Negative :",TN)

print("False Positive :",FP)

print("False Negative :",FN)

print("")

TPR **=** TP**/**(TP**+**FN)

TNR **=** TN**/**(TN**+**FP)

FPR **=** FP**/**(FP**+**TN)

FNR **=** FN**/**(TP**+**FN)

print("True Positive Rate :",TPR)

print("True Negative Rate :",TNR)

print("False Positive Rate :",FPR)

print("False Negative Rate :",FNR)

print("")

PPV **=** TP**/**(TP**+**FP)

NPV **=** TN**/**(TN**+**FN)

print("Positive Predictive Value :",PPV)

print("Negative predictive value :",NPV)

cm2**=**confusion\_matrix(y\_test, predicts)

print(cm2)

sns**.**heatmap(cm2**/**n**.**sum(cm2), annot**=True**, cmap **=** 'Blues', annot\_kws**=**{"size": 16}, fmt**=**'.2%',)

plt**.**show()