MACHINE LEARNING
(UCS611)

IONOSPHERE CLASSIFICATION

ASSIGNMENT - 3

Submitted by: -

Leeza 101715078

Naman Jain 101715093

Submitted to: -
Dr. Surbhi Sharma

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Department of Computer Science and Engineering
Thapar Institute of Engineering and Technology

Patiala (Punjab) -147004

January – May 2020

TABLE OF CONTENTS

OBJECTIVE: - Compare the training and testing phase performances of three classifiers on Ionosphere dataset.

Classifiers to be used: 1. KNN Classifier. 2. Decision Tree Classifier. 3. Random Forest Classifier

1. Performance with any five values of K and test ratio 30%, 40% and 50% for KNN.
2. Performance with any five values of max_depth and test ratio 30%, 40% and 50% for Decision Trees.
3. Performance with any five values of n_estimators and test ratio 30%, 40% and 50% for Random Forest.

SOFTWARE SPECIFICATIONS: -

1. Language: - Python
2. Coding Environment: -Anaconda 3(Spyder 4.0)
3. Packages: - Numpy, Matplotlib, Pandas.
4. ML model used: - K-Nearest Neighbours Classifier, Decision Tree Classifier, Random Forest classifier.

THEORY: -

1. **K-Nearest Neighbours Classifier: -** In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbour.

2. **Decision Tree Classifier: -** Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

3. **Random Forest classifier: -** Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.[1][2] Random decision forests correct for decision trees' habit of overfitting to their training set.

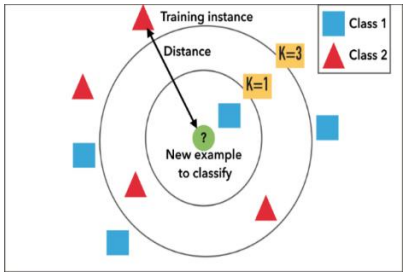


Fig. 1.a KNN Classifier

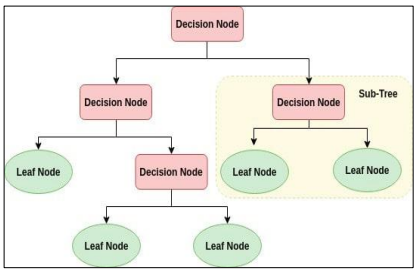


Fig. 1.b Decision Tree Classifier

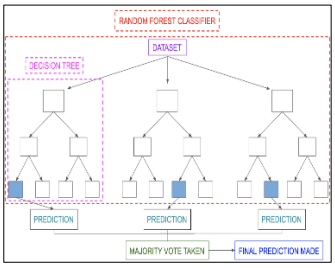


Fig. 1.c Random Forest Classifier

Models Snapshots and Graphs: -

Using the label encoder:

- **Good 'g' output is encoded as 1**
- **Bad 'b' output is encoded as 0**

1. **K-Nearest Neighbours Classifier: -**

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('ionosphere.csv', header = None)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30,
random_state = 0)

# Applying PCA
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_

# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 1, metric = 'minkowski', p
= 2)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
Confusion_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)


# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1,       X2,       classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
```

```
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('KNN (Training set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1,       X2,       classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('KNN (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

a. Test-size: - 30%

- **K-value: - 3**

Accuracy: - 77.35%

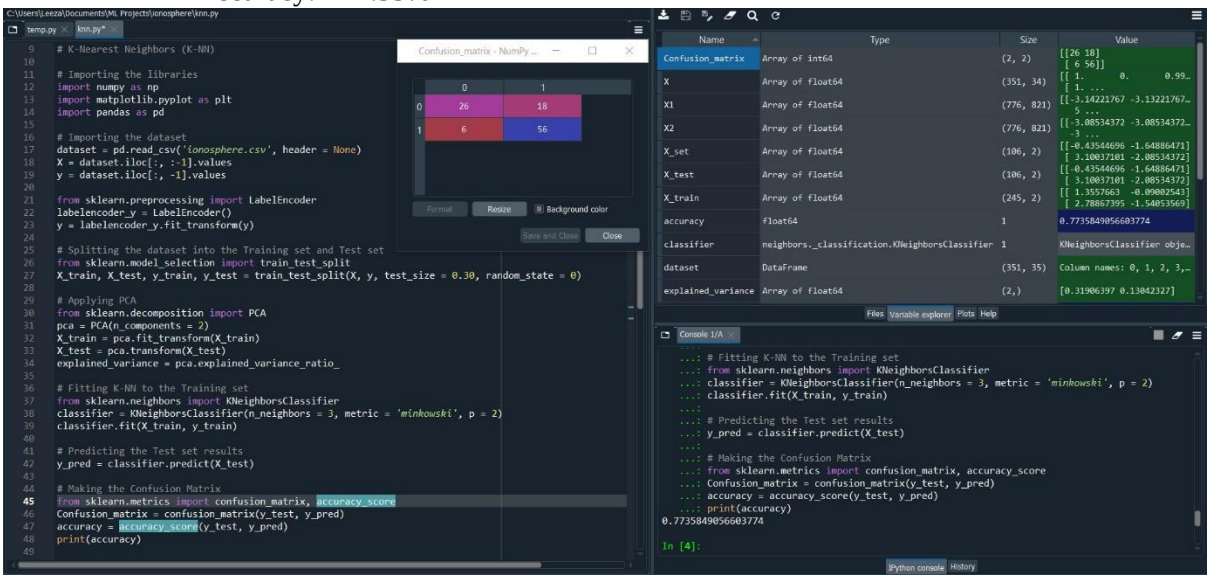


Fig. 1.1.1.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

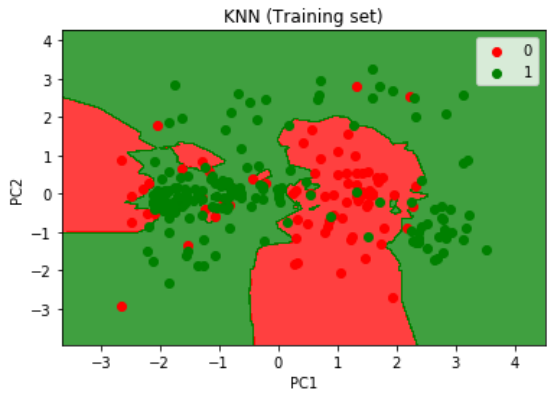


Fig. 1.1.1.b Training Set Graph

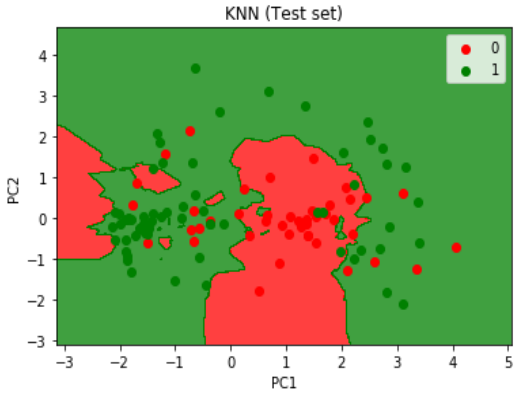


Fig. 1.1.1.c Test Set Graph

- **K-value: - 4**
  Accuracy: - 78.30%

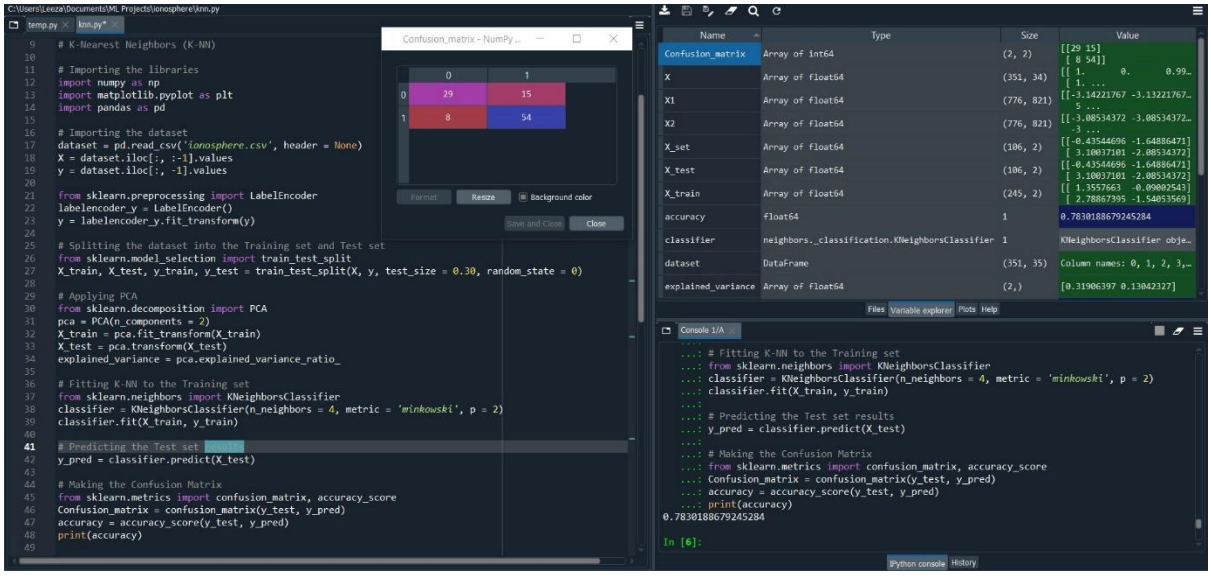


Fig. 1.1.2.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

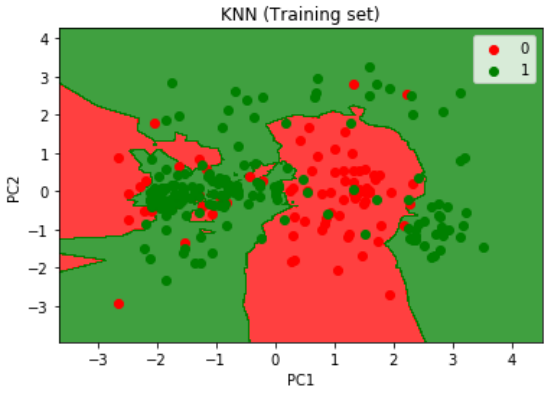


Fig. 1.1.2.b Training Set Graph

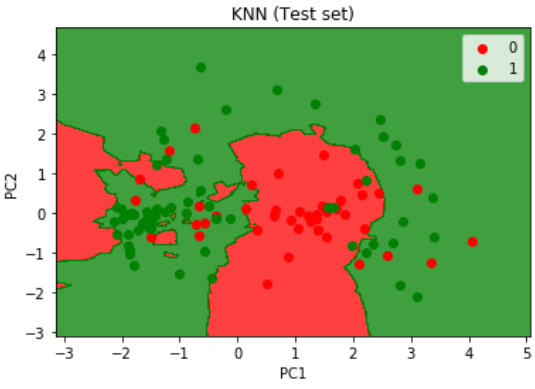


Fig. 1.1.2.c Test Set Graph

- **K-value: - 5**
  Accuracy: - 78.30%

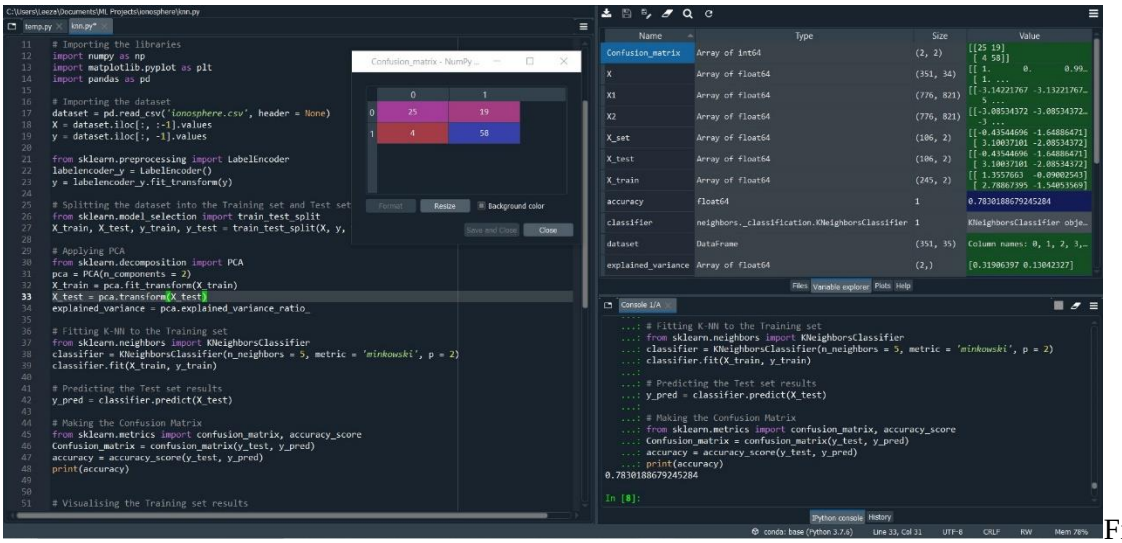


Fig. 1.1.3.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

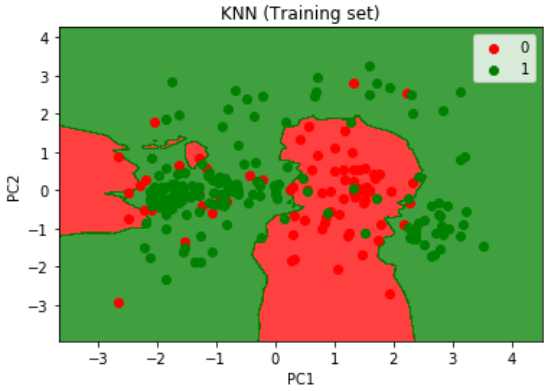


Fig. 1.1.3.b Training Set Graph

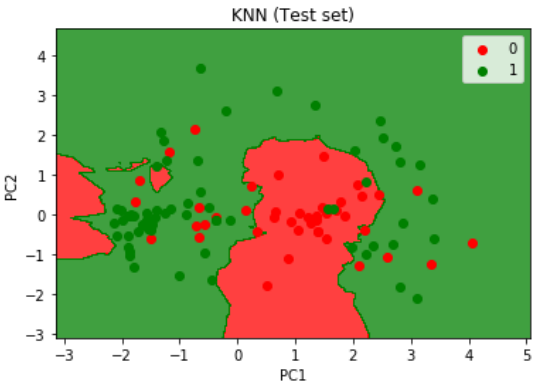


Fig. 1.1.3.c Test Set Graph

- **K-value: - 6**
  Accuracy: - 80.18%

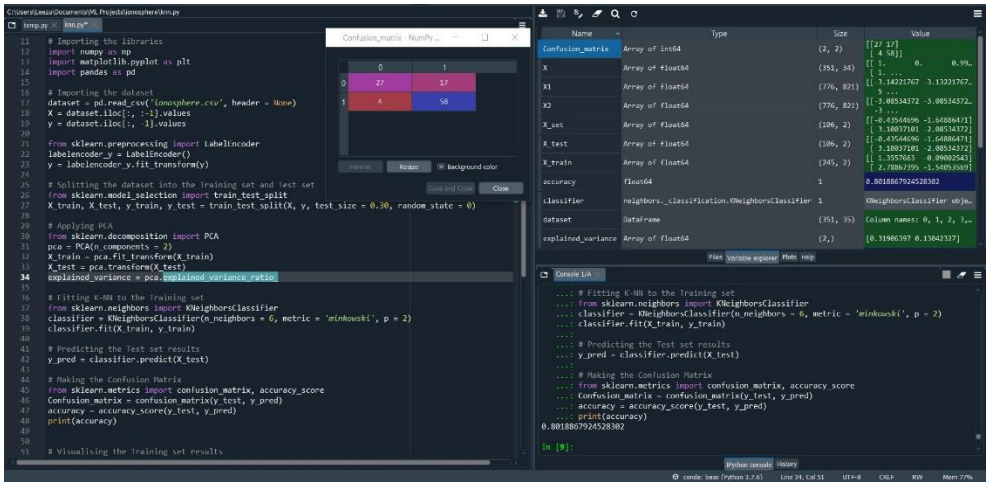


Fig. 1.1.4.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

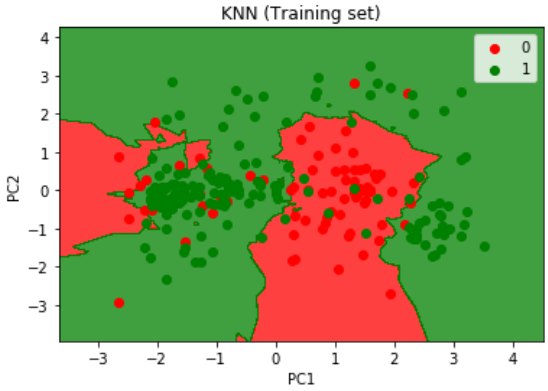


Fig. 1.1.4.b Training Set Graph

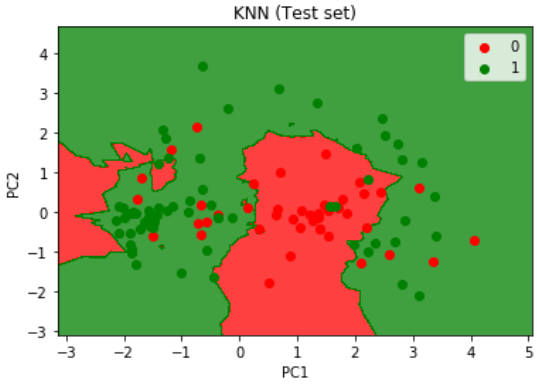
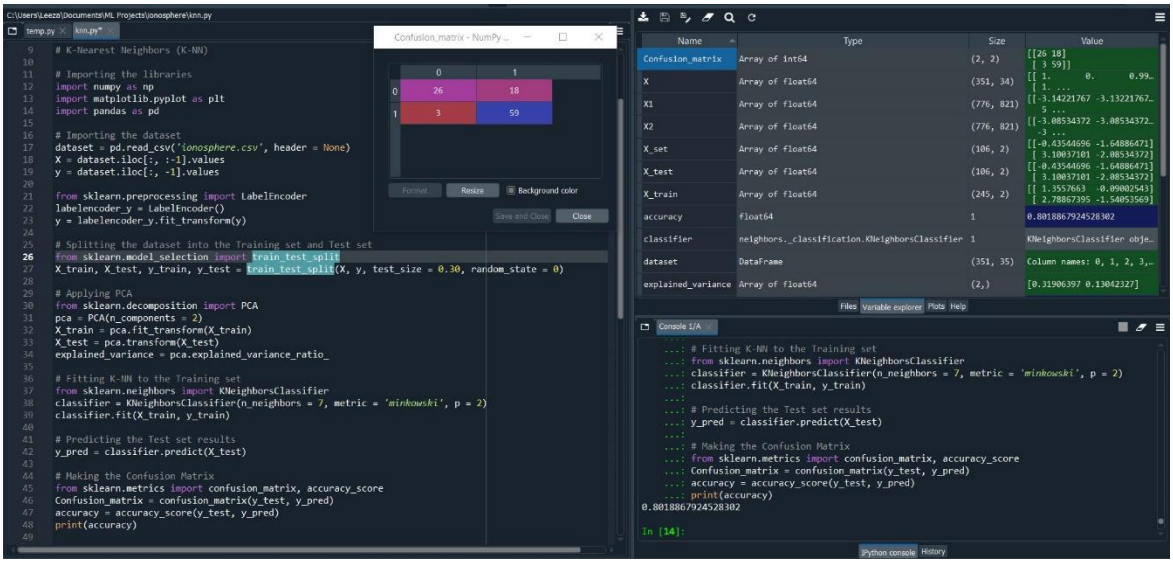


Fig. 1.1.4.c Test Set Graph

- **K-value: -7**
  Accuracy: - 80.18%



```
# K-Nearest Neighbors (K-NN)

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('ionosphere.csv', header = None)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 0)

# Applying PCA
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_

# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 7, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
Confusion_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

```
# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 7, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
Confusion_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
0.8018867924528302
```

Fig. 1.1.5.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

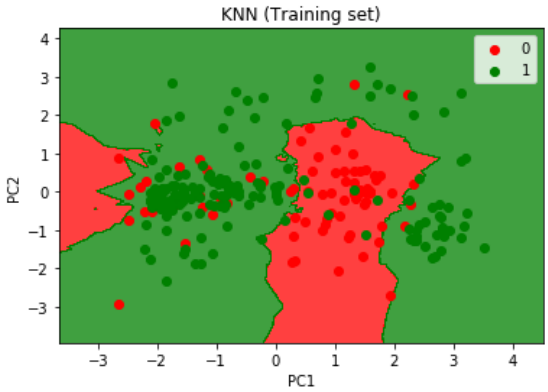


Fig. 1.1.5.b Training Set Graph

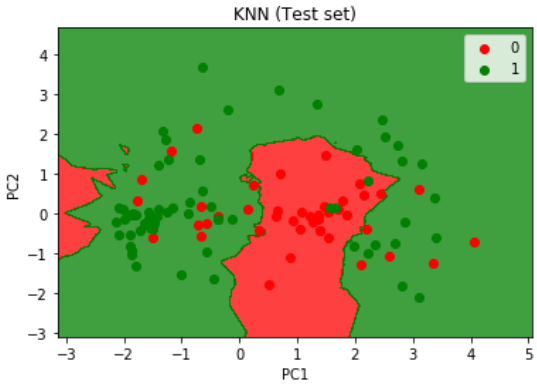


Fig. 1.1.5.c Test Set Graph

Conclusion: - The classification with test size = 30% gives best accuracy score = 80.18% with K-value = 6 and 7.

b. Test-size: - 40%

- **K-value: - 3**

  Accuracy: - 80.85%

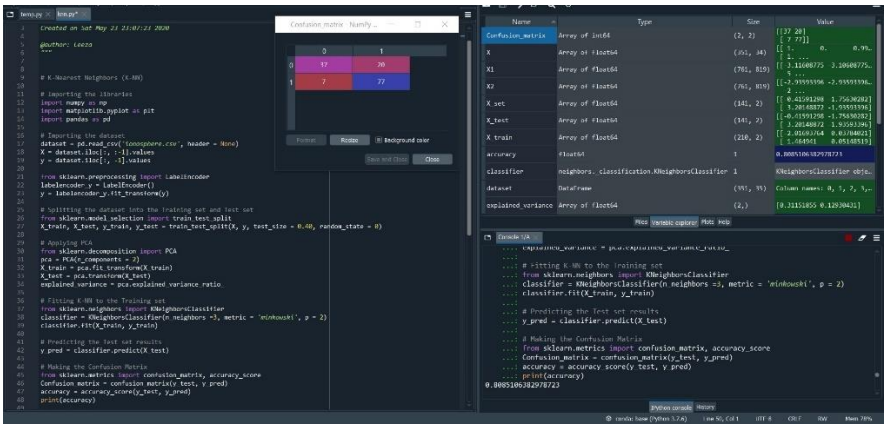


Fig. 1.2.1.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

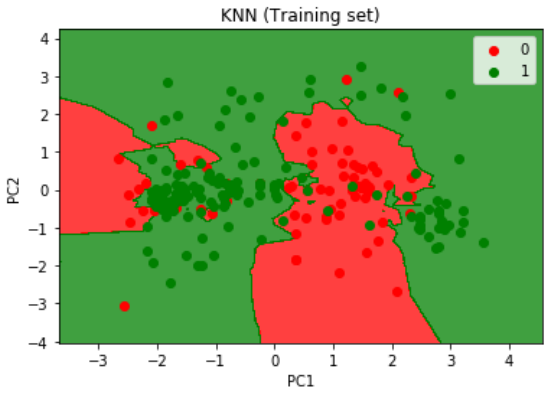


Fig. 1.2.1.b Training Set Graph

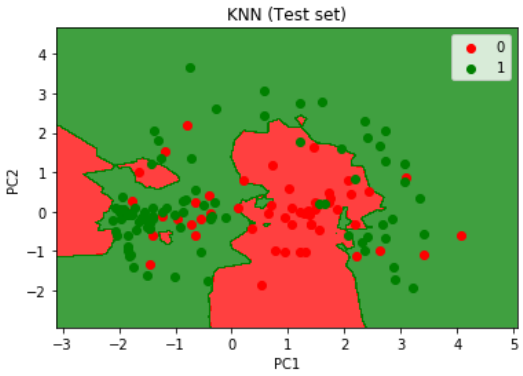


Fig. 1.2.1.c Test Set Graph

- **K-value: - 4**

  Accuracy: - 80.14%

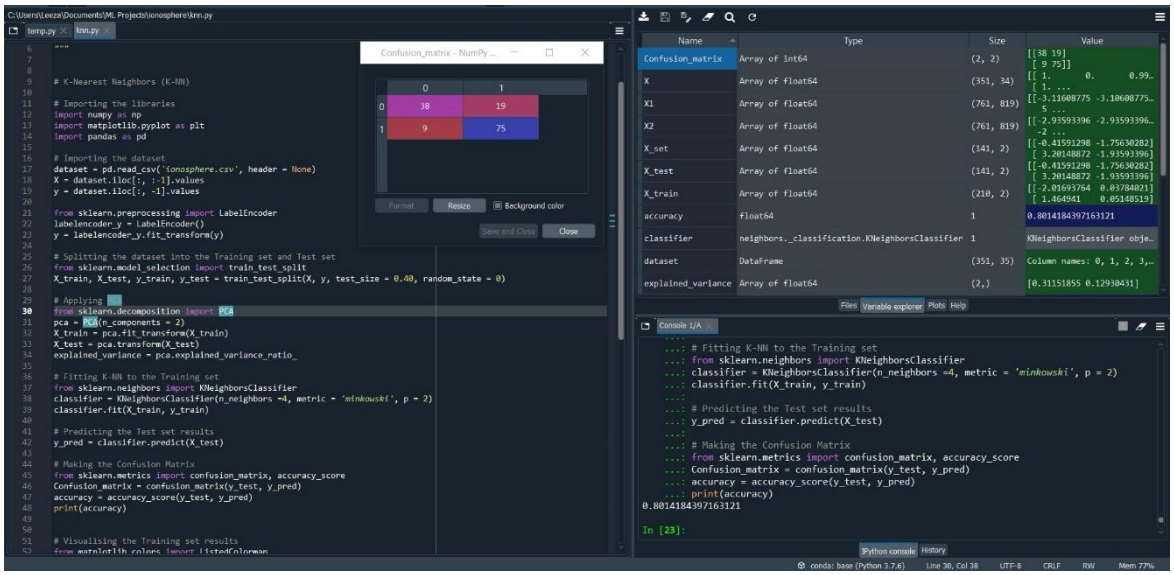


Fig. 1.2.2.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

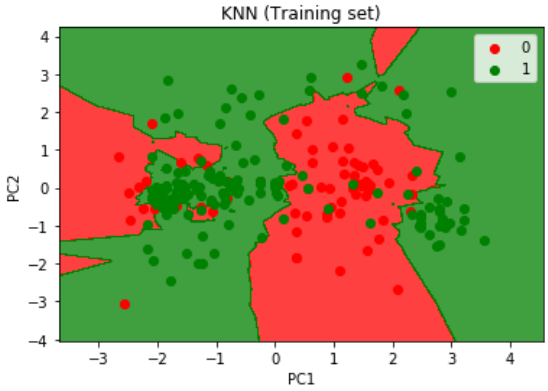


Fig. 1.2.2.b Training Set Graph

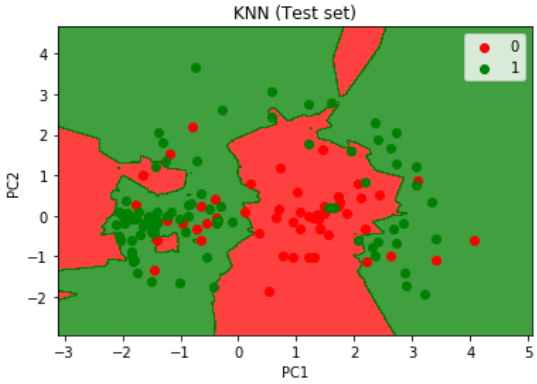


Fig. 1.2.2.c Test Set Graph

- **K-value: - 5**
  Accuracy: - 79.43%

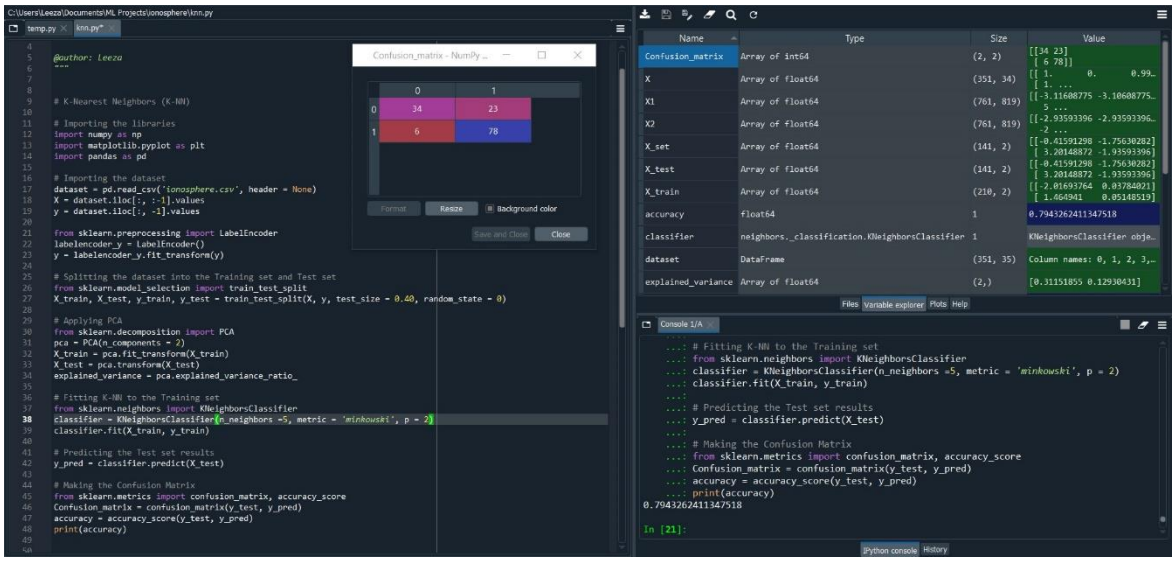


Fig. 1.2.3.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

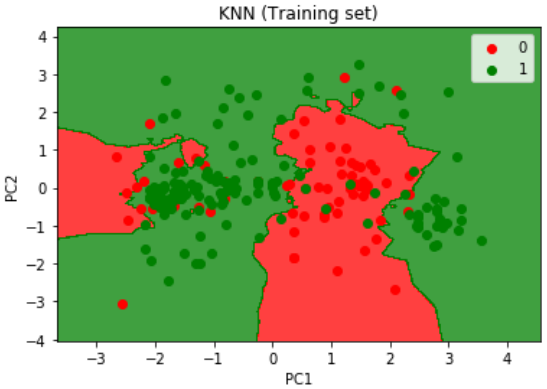


Fig. 1.2.3.b Training Set Graph

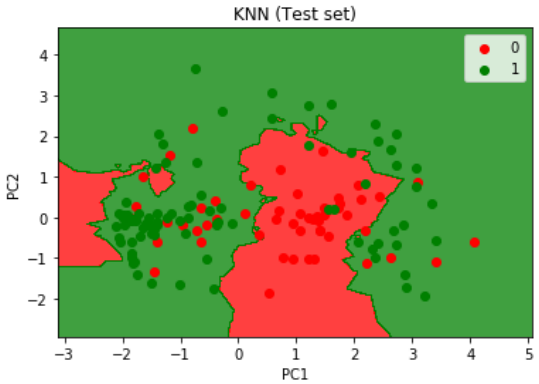


Fig. 1.2.3.c Test Set Graph

- **K-value: - 6**
  Accuracy: - 79.43%

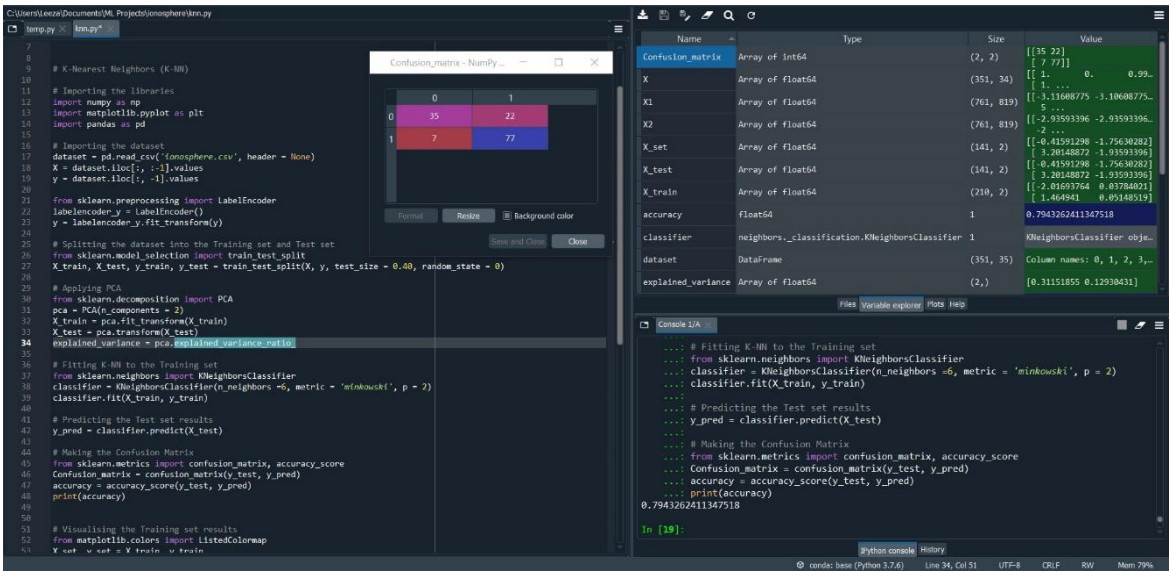


Fig. 1.2.4.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

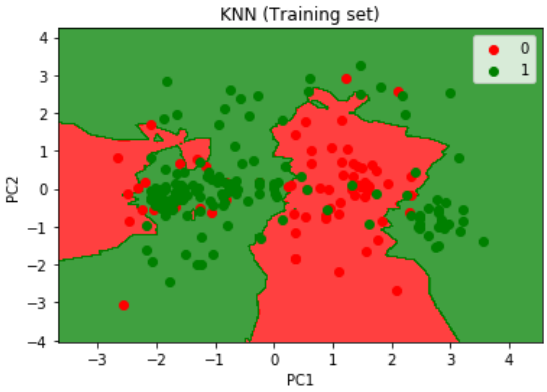


Fig. 1.2.4.b Training Set Graph

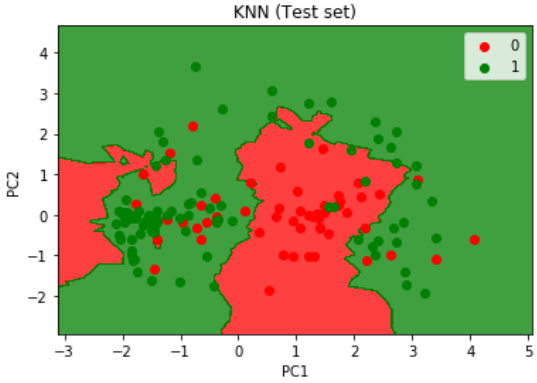


Fig. 1.2.4.c Test Set Graph

- **K-value: -7**
  Accuracy: - 79.43%

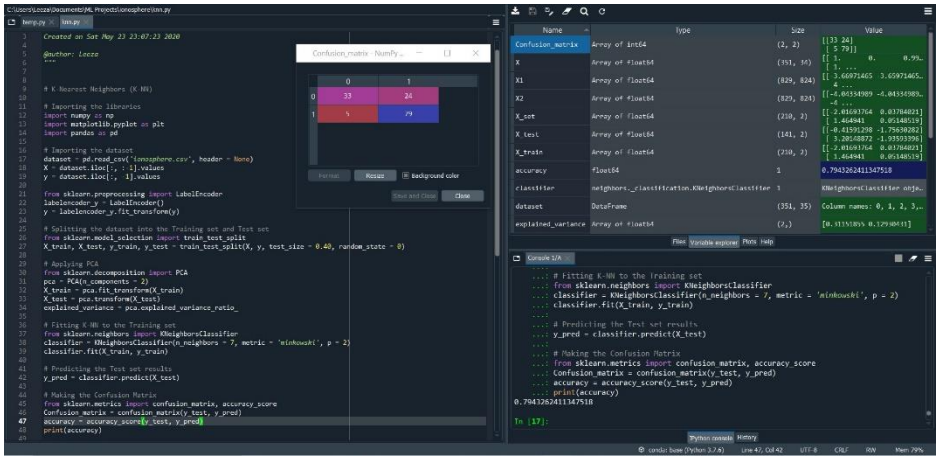


Fig. 1.2.5.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

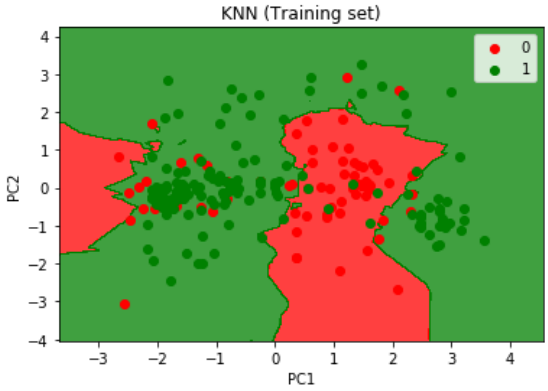


Fig. 1.2.5.b Training Set Graph

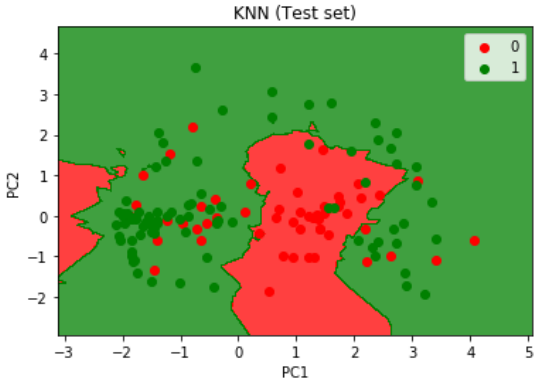


Fig. 1.2.5.c Test Set Graph

Conclusion: - The classification with test size = 40% gives best accuracy score = 80.85% with K-value = 3.

c. Test-size: - 50%

- **K-value: - 3**

  Accuracy: - 78.97%

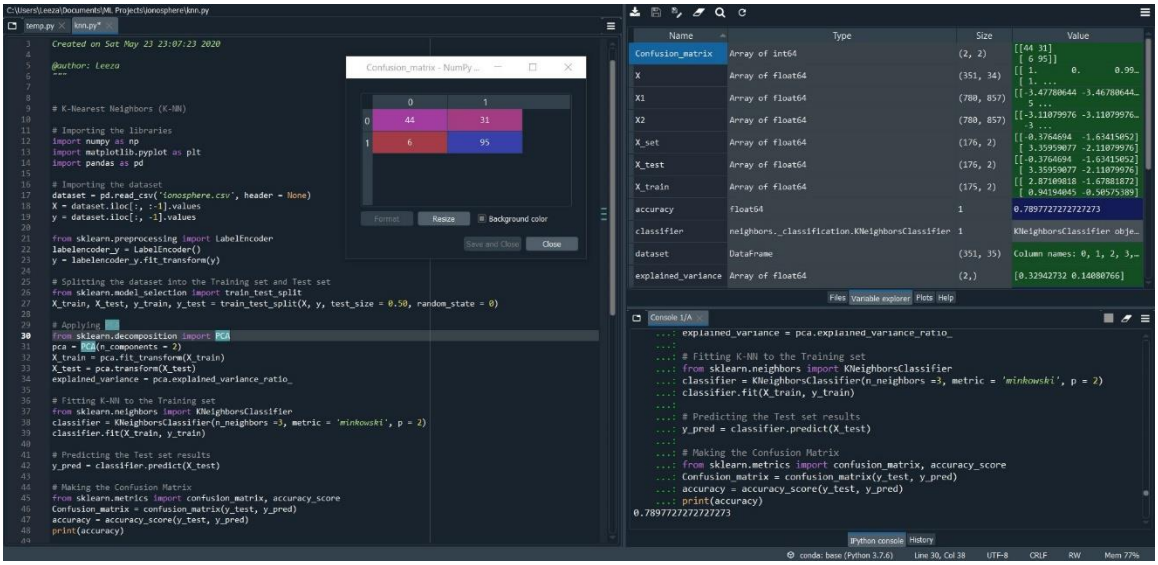


Fig. 1.3.1.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

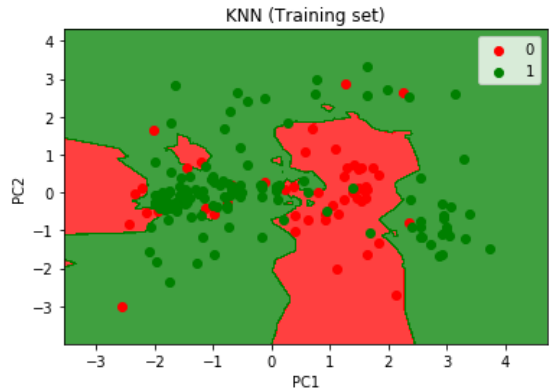


Fig. 1.3.1.b Training Set Graph

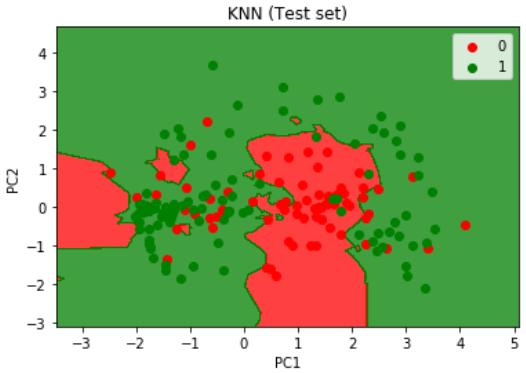


Fig. 1.3.1.c Test Set Graph

- **K-value: - 4**
  Accuracy: - 82.38%

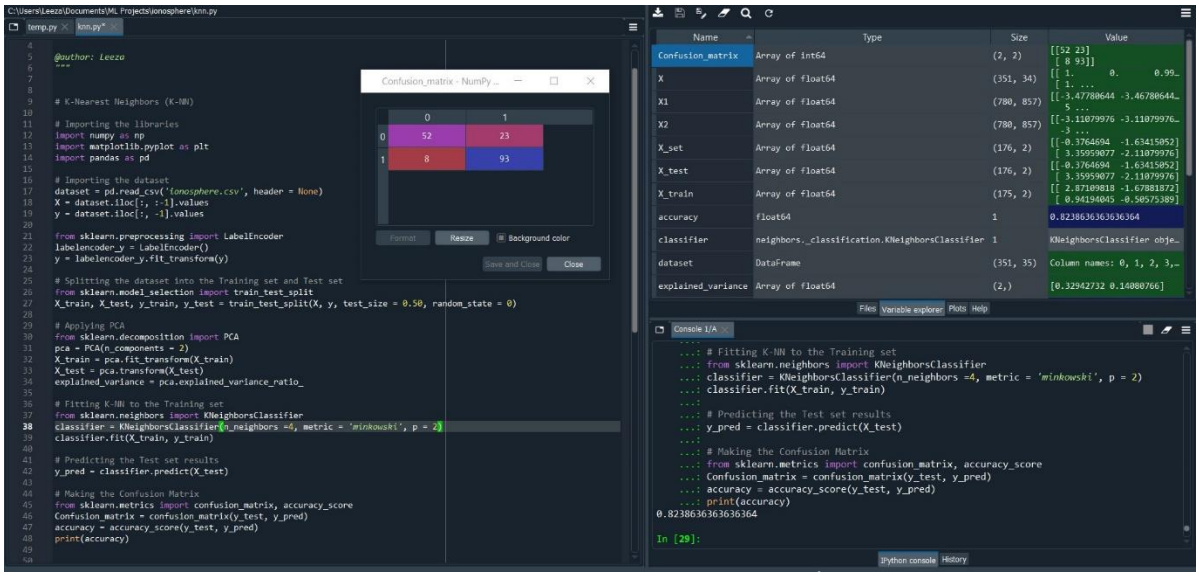


Fig. 1.3.2.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

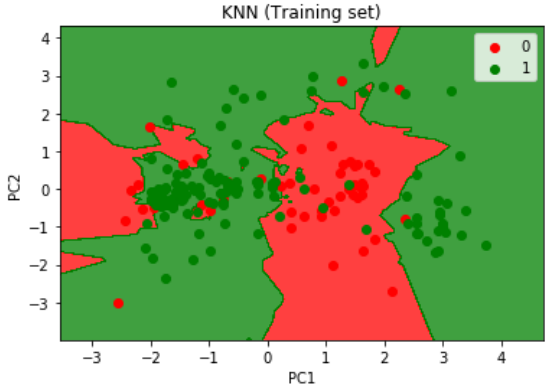


Fig 1.3.2.b Training Set Graph

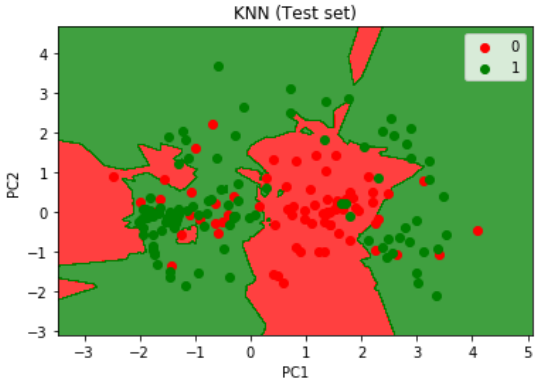


Fig. 1.3.2.c Test Set Graph

- **K-value: - 5**
  Accuracy: - 81.81%

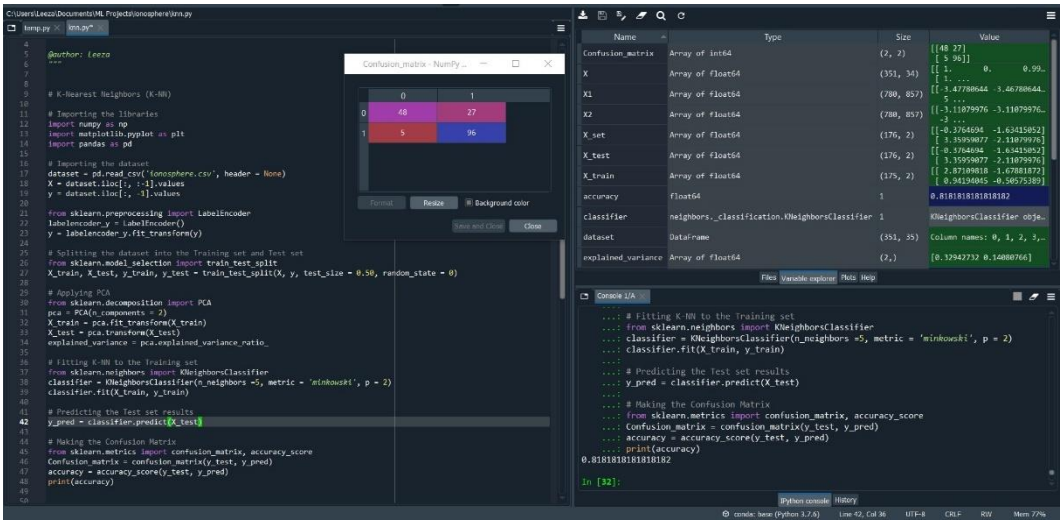


Fig. 1.3.3.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

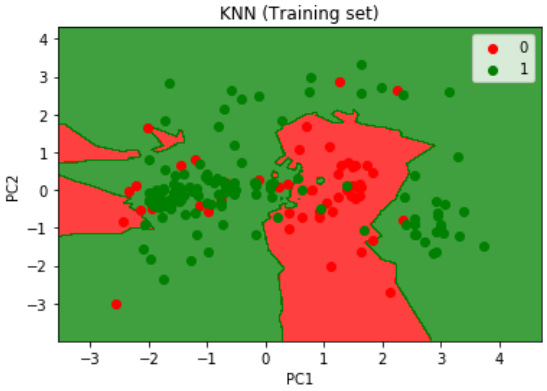


Fig. 1.3.3.b Training Set Graph

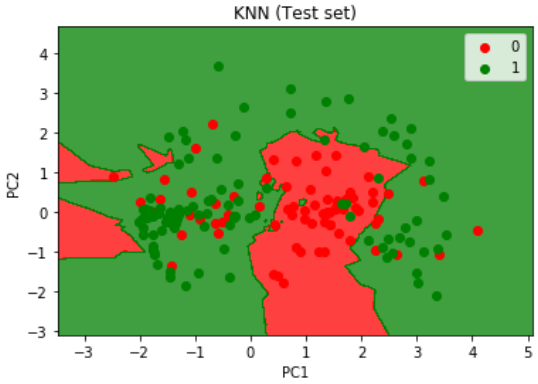


Fig. 1.3.3.c Test Set Graph

- **K-value: - 6**
  Accuracy: - 82.18%

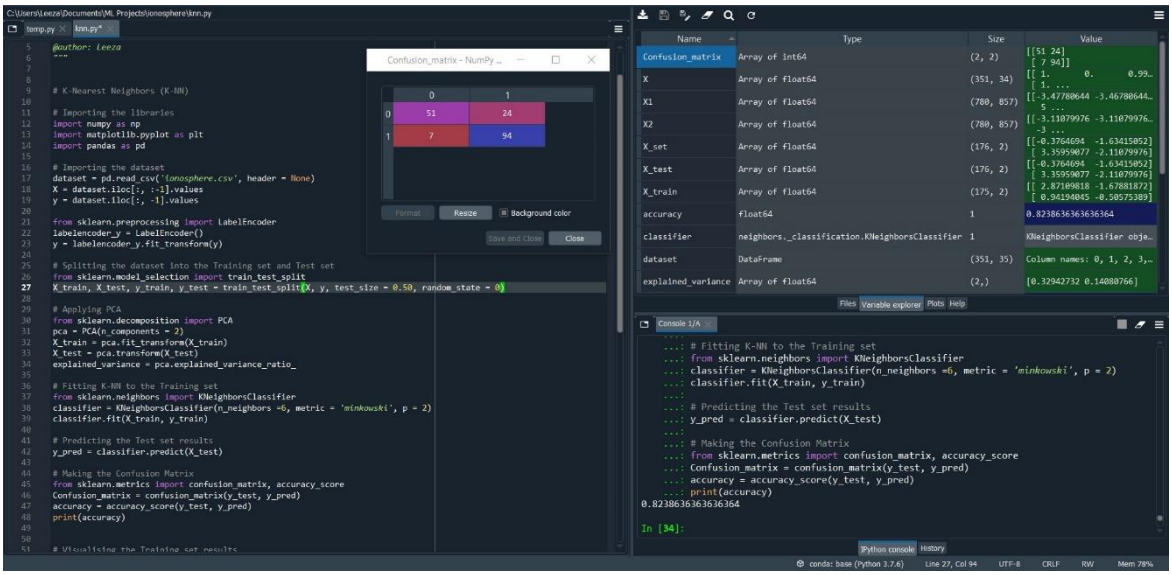


Fig. 1.3.4.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

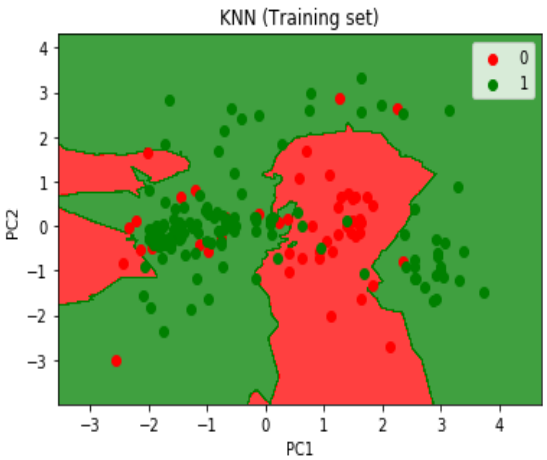


Fig. 1.3.4.b Training Set Graph

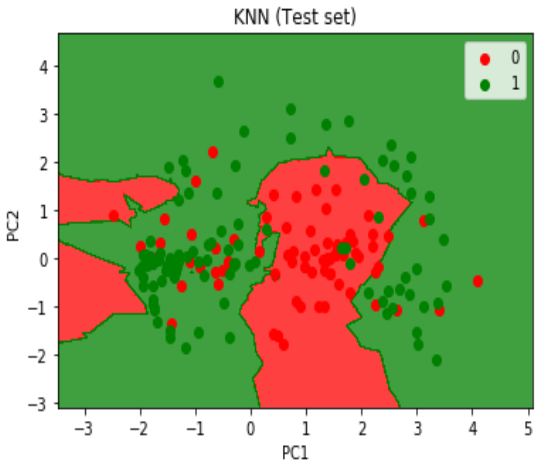


Fig. 1.3.4.c Test Set Graph

- **K-value: -7**
  Accuracy: - 80.68%

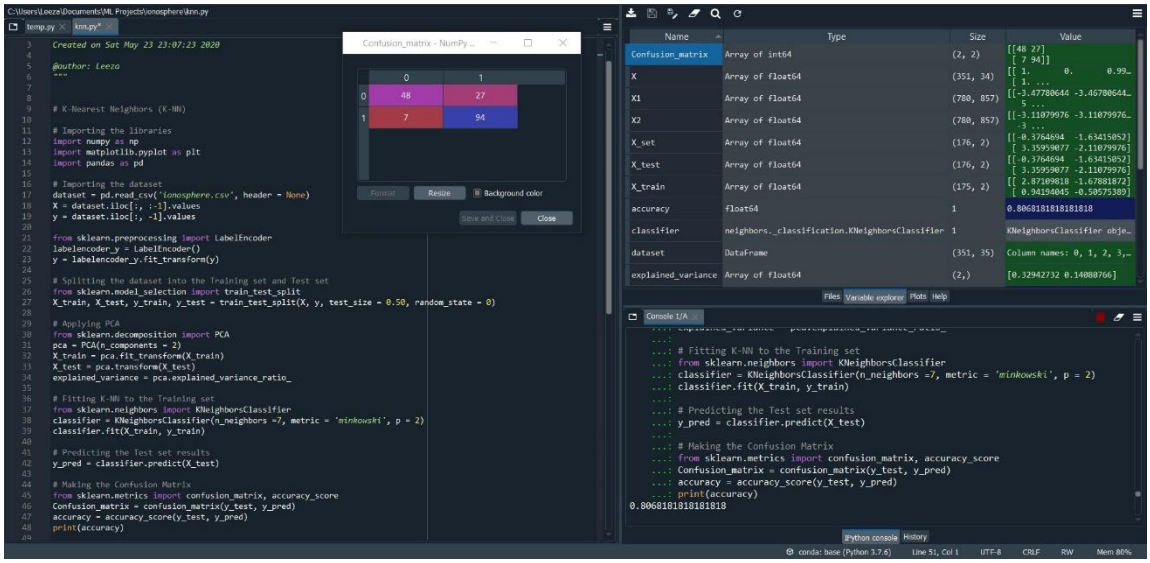


Fig. 1.3.5.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

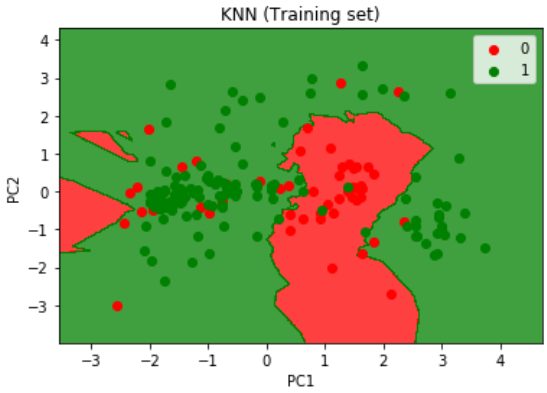


Fig. 1.3.5.b Training Set Graph

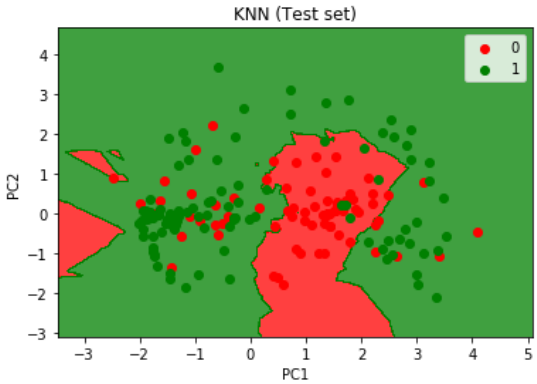


Fig. 1.3.5.c Test Set Graph

<u>Conclusion</u>: - The classification with test size = 50% gives best accuracy score = 82.38% with K-value = 4.

2. Decision Tree Classifier: -

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('ionosphere.csv', header = None)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
```

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
//values of test size, random_state = 0)


# Applying PCA
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_

# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy',max_depth =
//value of max_depth of the tree, random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
Confusion_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)


# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1,       X2,       classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree (Training set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1,       X2,       classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
```

```
            alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

a. Test-size: - 30%

- **max_depth = 1**
  Accuracy: - 70.75%

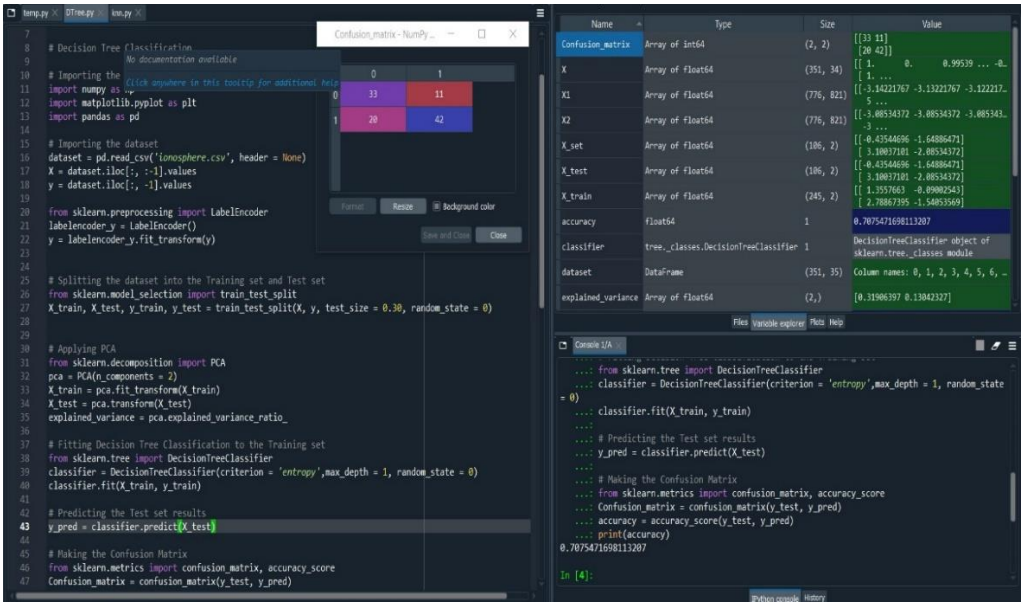


Fig. 2.1.1.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

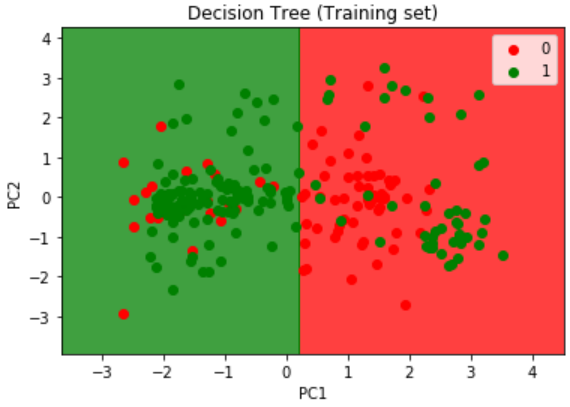


Fig. 2.1.1.b Training Set Graph

Fig. 2.1.1.c Test Set Graph

- **max_depth = 2**
  Accuracy: - 77.35%

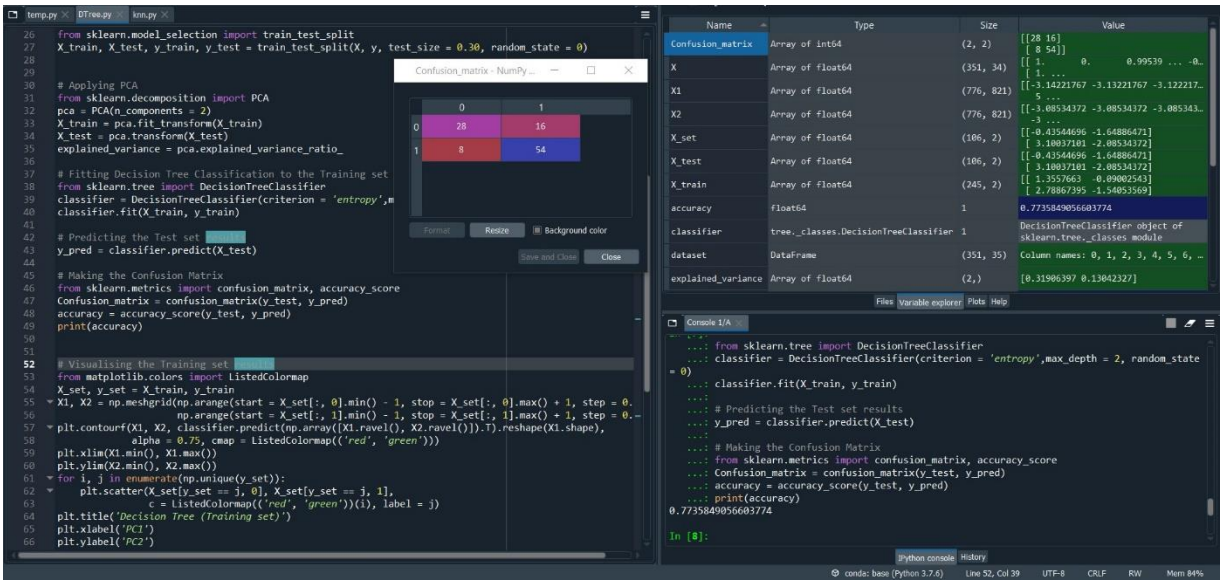


Fig. 2.1.2.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

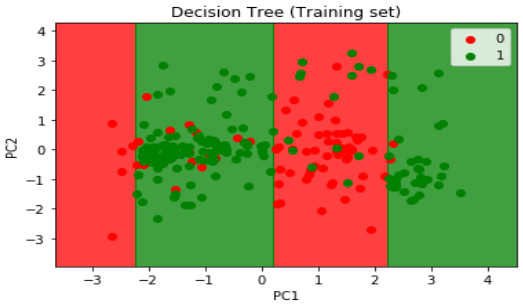


Fig. 2.1.2.b Training Set Graph

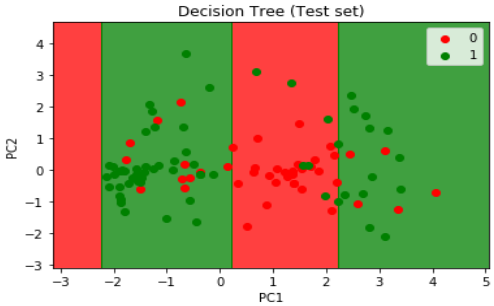


Fig. 2.1.2.c Test Set Graph

- **max_depth = 3**
  Accuracy: - 79.24%

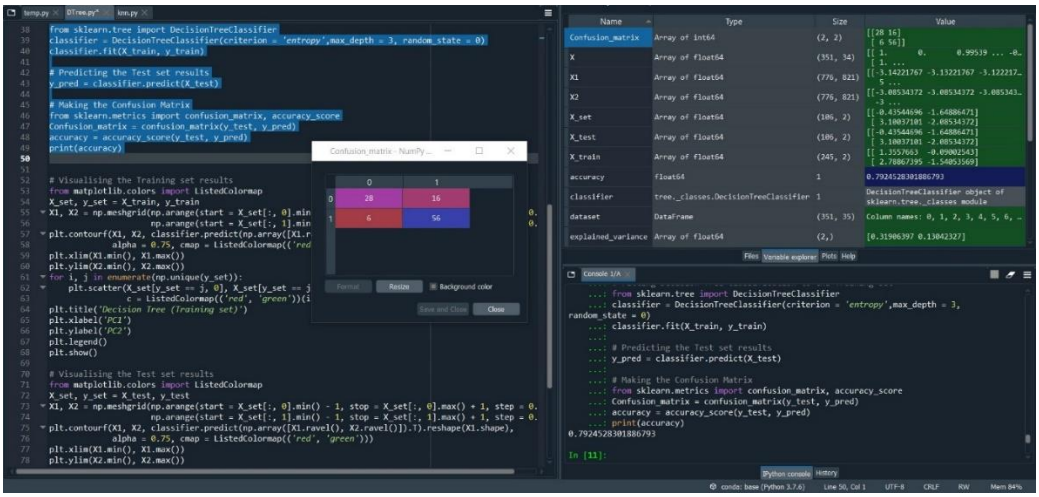


Fig. 2.1.3.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

Fig. 2.1.3.b Training Set Graph

Fig. 2.1.3.c Test Set Graph

- **max_depth = 6**
  Accuracy: - 78.30%

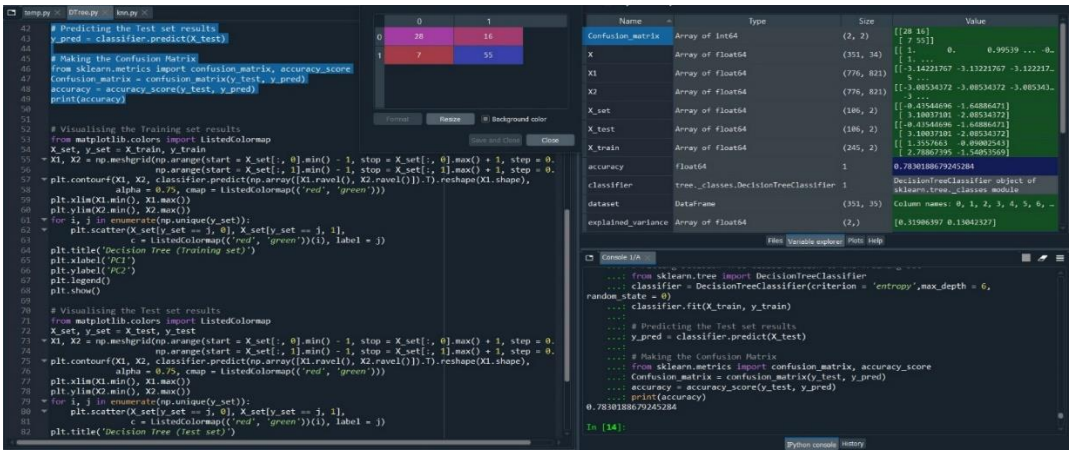


Fig. 2.1.4.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

Fig. 2.1.4.b Training Set Graph

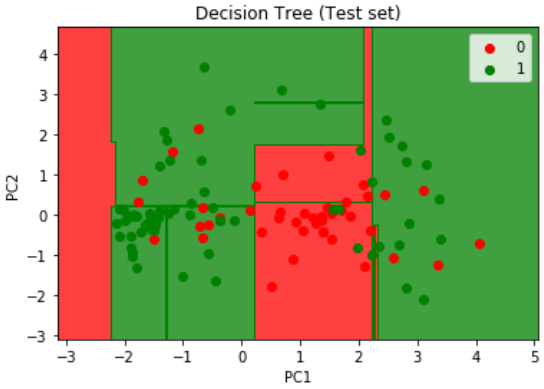


Fig. 2.1.4.c Test Set Graph

- **max_depth = 7**
  Accuracy: - 78.30%

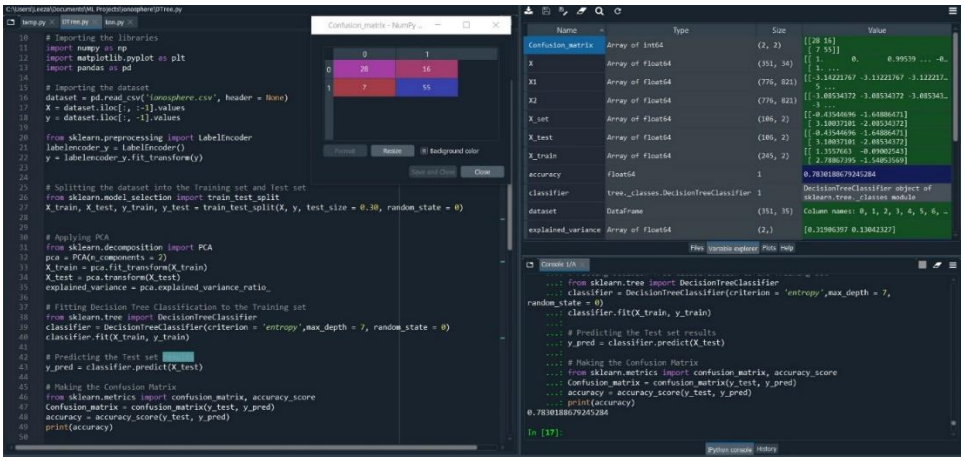


Fig. 2.1.5.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

Fig. 2.1.5.b Training Set Graph

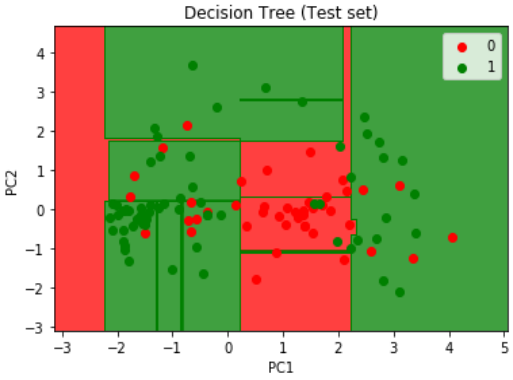


Fig. 2.1.5.c Test Set Graph

Conclusion: - The classification model with max_depth = 30% yields best accuracy of 79.24% among all the values of max_depth.

b. Test-size: - 40%

- **max_depth = 1**
  Accuracy: - 68.79%

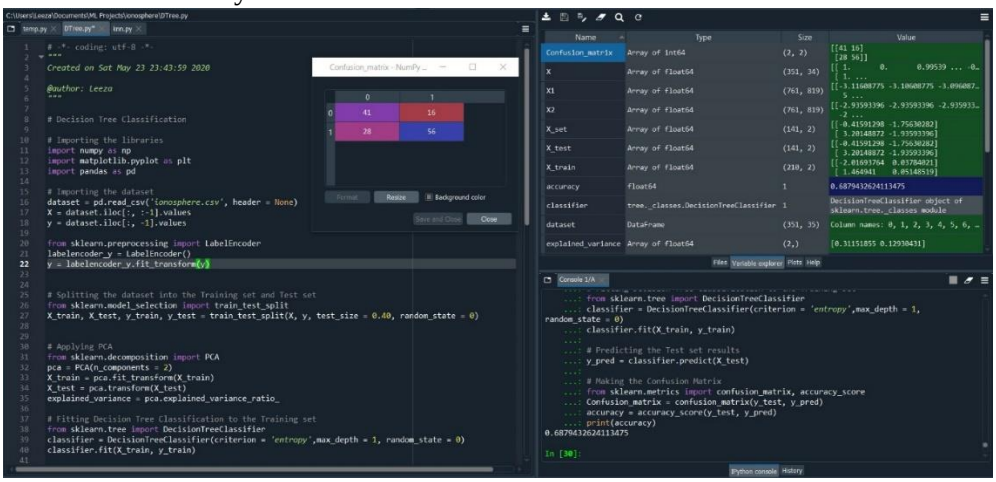


Fig. 2.2.1.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

Fig. 2.2.1.b Training Set Graph

Fig. 2.2.1.c Test Set Graph

- **max_depth = 2**
  Accuracy: - 75.88%

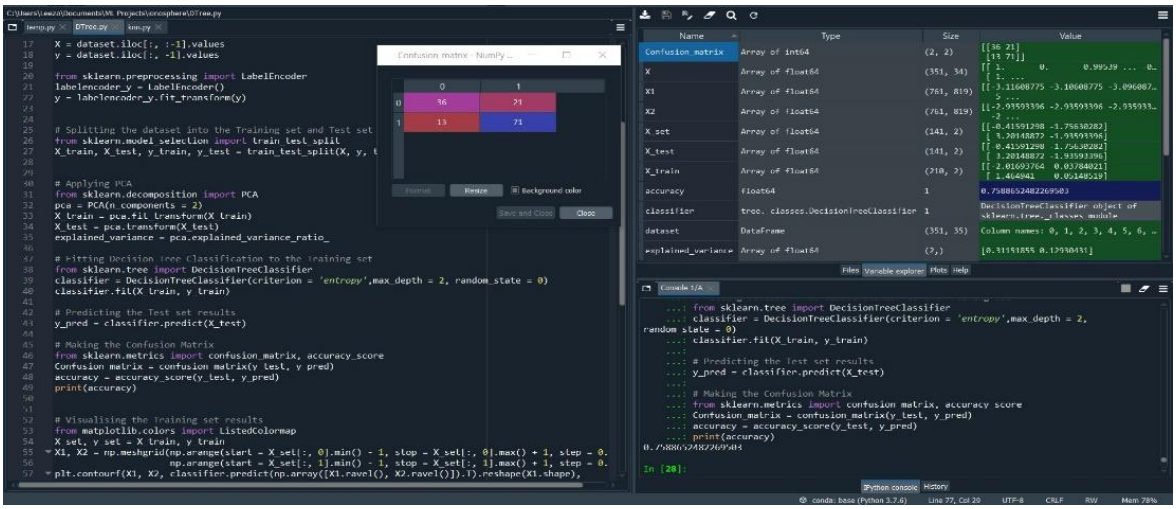


Fig. 2.2.2.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

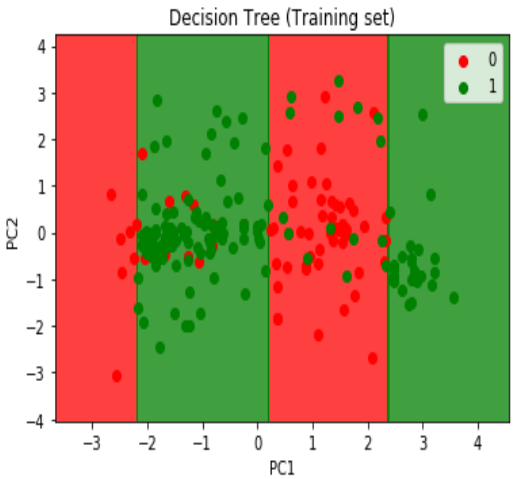


Fig. 2.2.2.b Training Set Graph

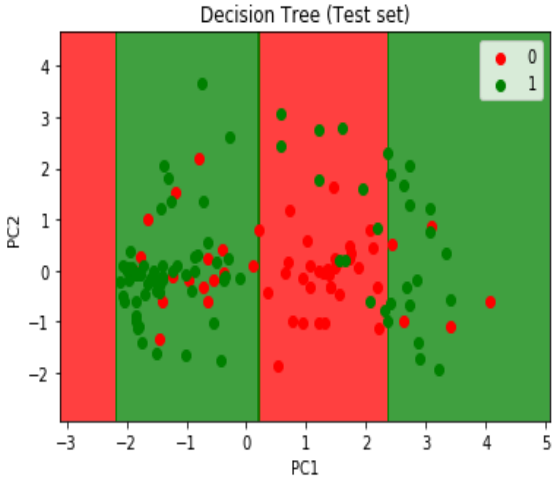


Fig. 2.2.2.c Test Set Graph

- **max_depth = 3**
  Accuracy: -79.43%

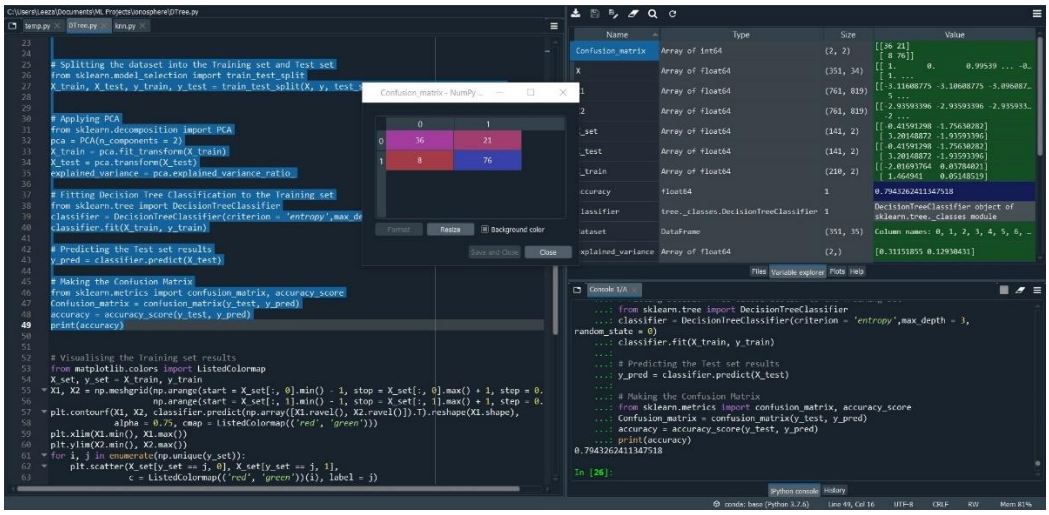


Fig. 2.2.3.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

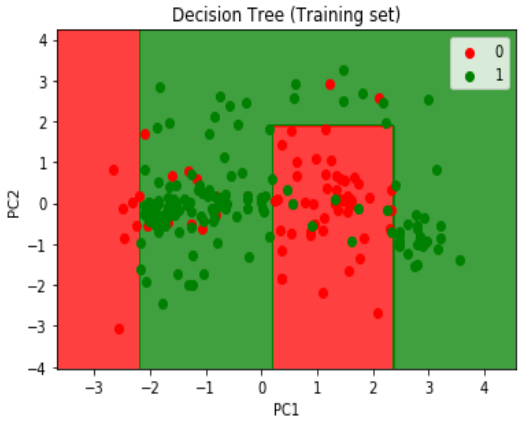


Fig. 2.2.3.b Training Set Graph

Fig. 2.2.3.c Test Set Graph

- **max_depth = 6**
  Accuracy: - 79.43%

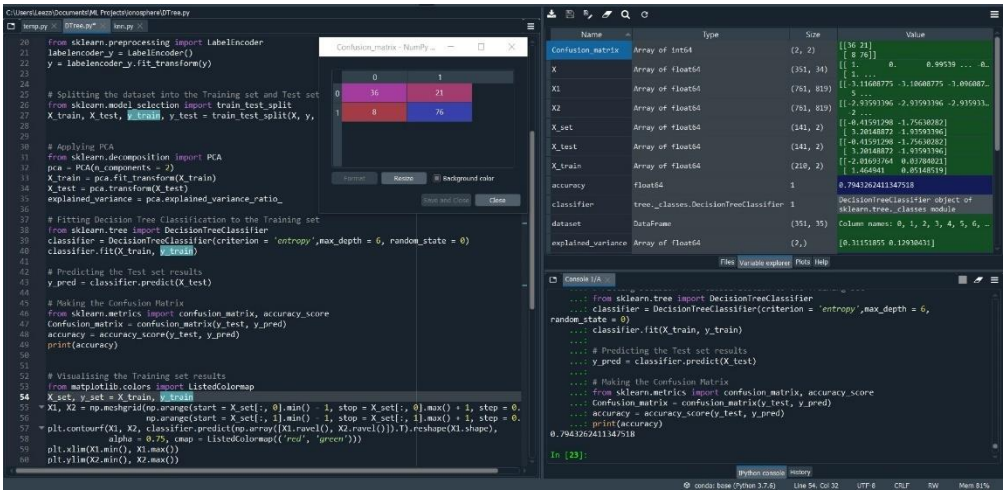


Fig. 2.2.4.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

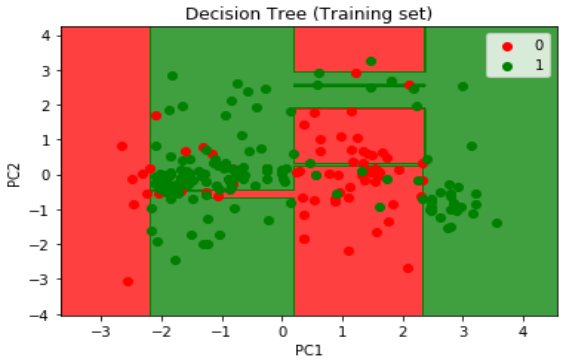


Fig. 2.2.4.b Training Set Graph

Fig. 2.2.4.c Test Set Graph

- **max_depth = 7**
  Accuracy: =78.72%

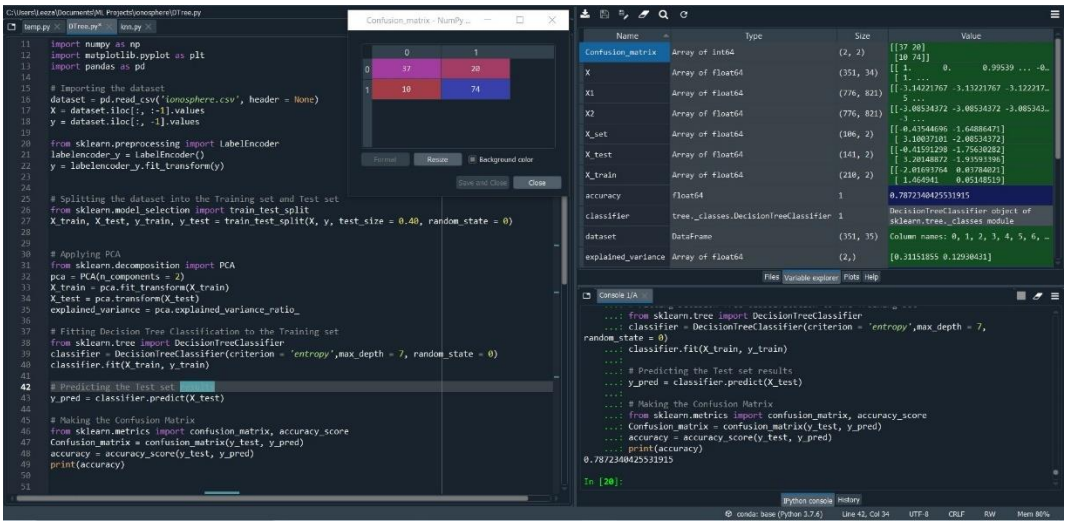


Fig. 2.2.5.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

Fig. 2.2.5.b Training Set Graph

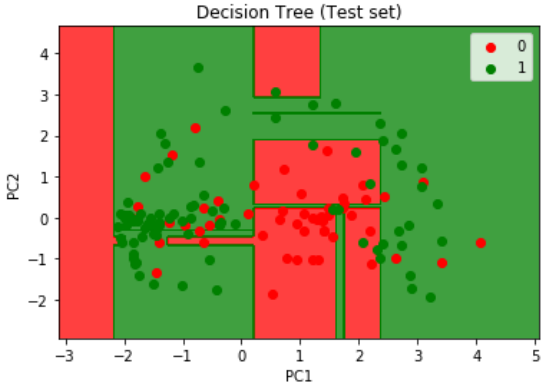


Fig. 2.2.5.c Test Set Graph

<u>Conclusion</u>: - The classification model with max_depth = 40% yields best accuracy of 79.43% among all the values of max_depth.

c. Test-size: - 50%

- **max_depth = 1**

  Accuracy: =78.72%

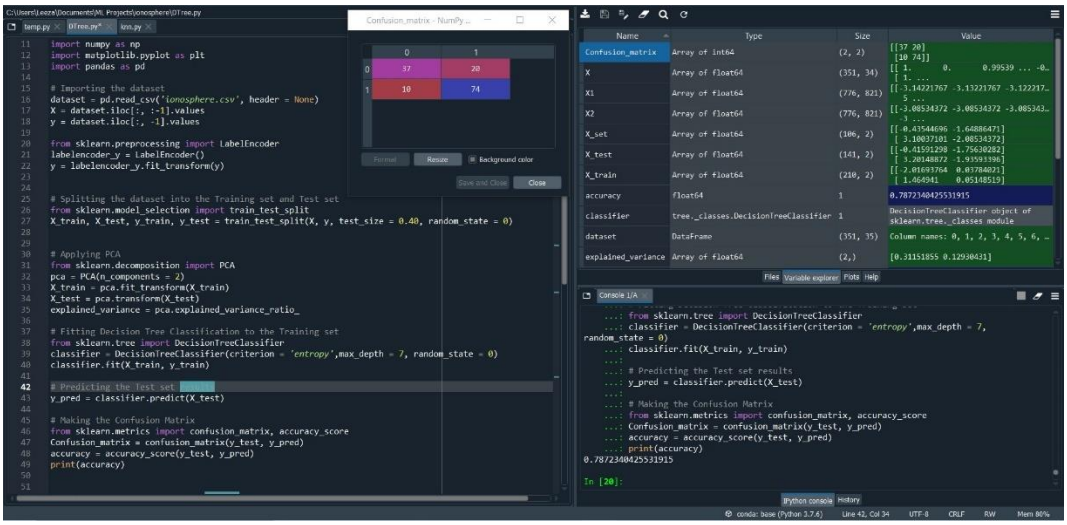


Fig. 2.3.1.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

Fig. 2.3.1.b Training Set Graph

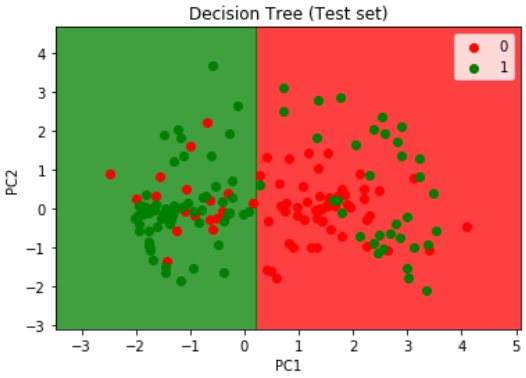


Fig. 2.3.1.c Test Set Graph

- **max_depth = 2**

  Accuracy: =69.31%

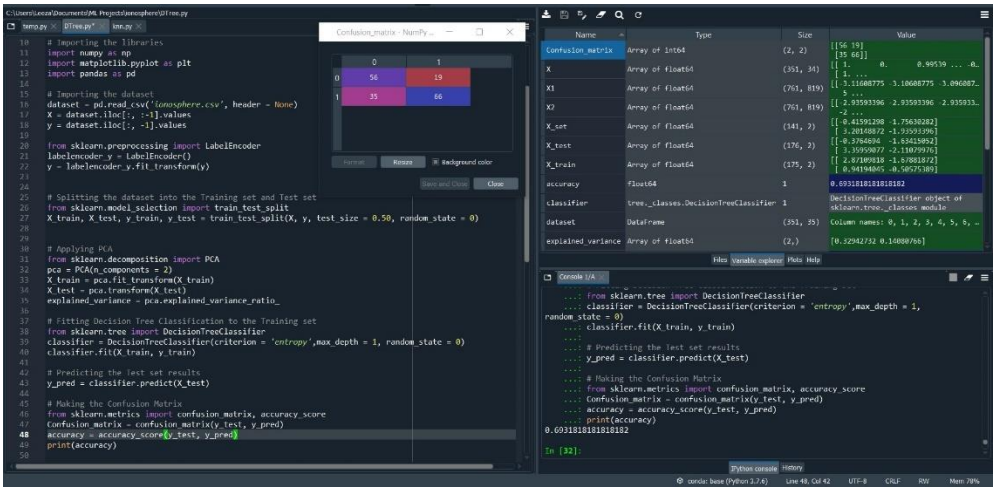


Fig. 2.3.2.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

Fig. 2.3.2.b Training Set Graph

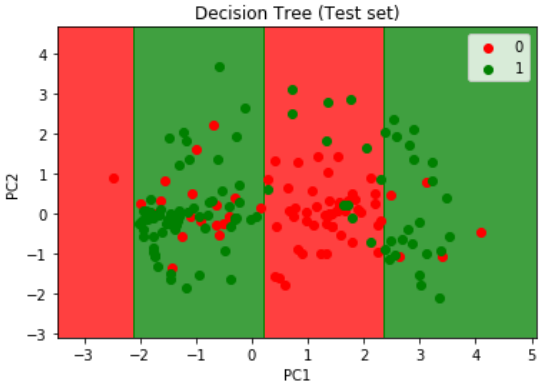


Fig. 2.3.2.c Test Set Graph

- **max_depth = 3**

Accuracy: =82.29%

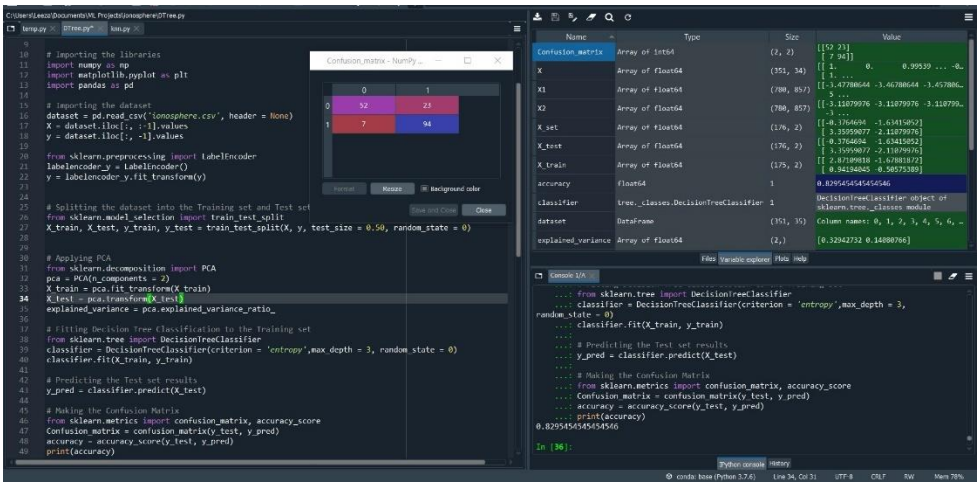


Fig. 2.3.3.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

Fig. 2.3.3.b Training Set Graph

Fig. 2.3.3.c Test Set Graph

- **max_depth = 6**

Accuracy: =78.97%

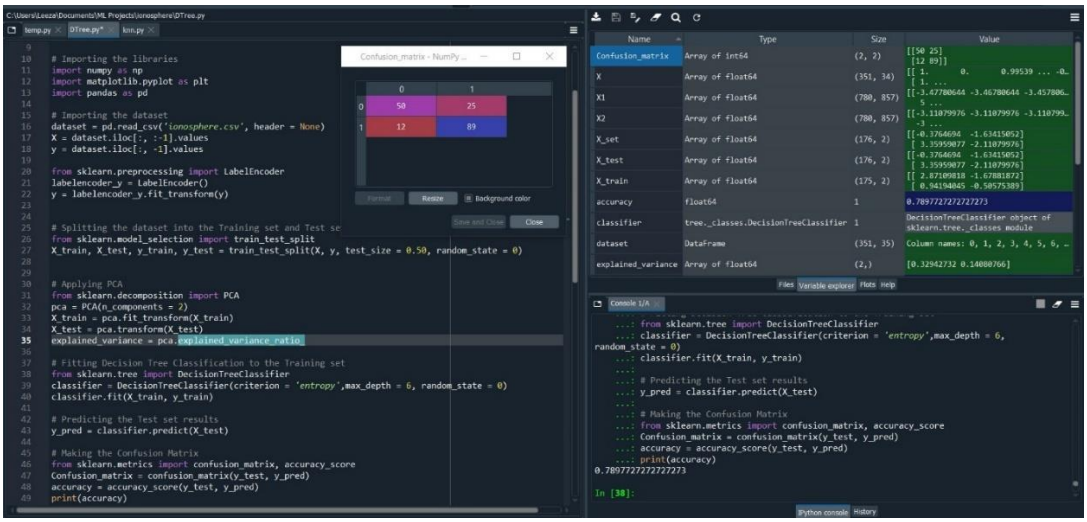


Fig. 2.3.4.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

Fig. 2.3.4.b Training Set Graph

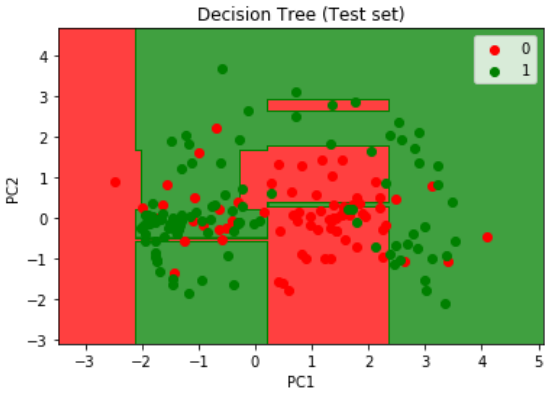


Fig. 2.3.4.c Test Set Graph

- **max_depth = 7**

Accuracy: =78.40%

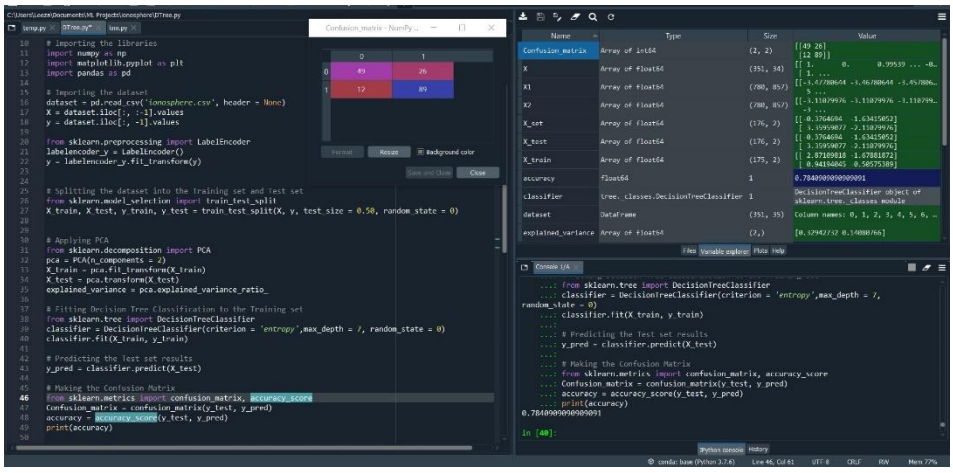


Fig. 2.3.5.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

Fig. 2.3.5.b Training Set Graph

Fig. 2.3.5.c Test Set Graph

Conclusion: - The classification model with max_depth = 50% yields best accuracy of 82.29% among all the values of max_depth.

3. Random Forest classifier: -

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('ionosphere.csv', header = None)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
//test size value, random_state = 0)

# Applying PCA
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_

# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators =//value of
n_estimators, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
Confusion_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)


# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest (Training set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

a. Test-size: - 30%

- **N_estimators: - 3**

  Accuracy: - 73.58%

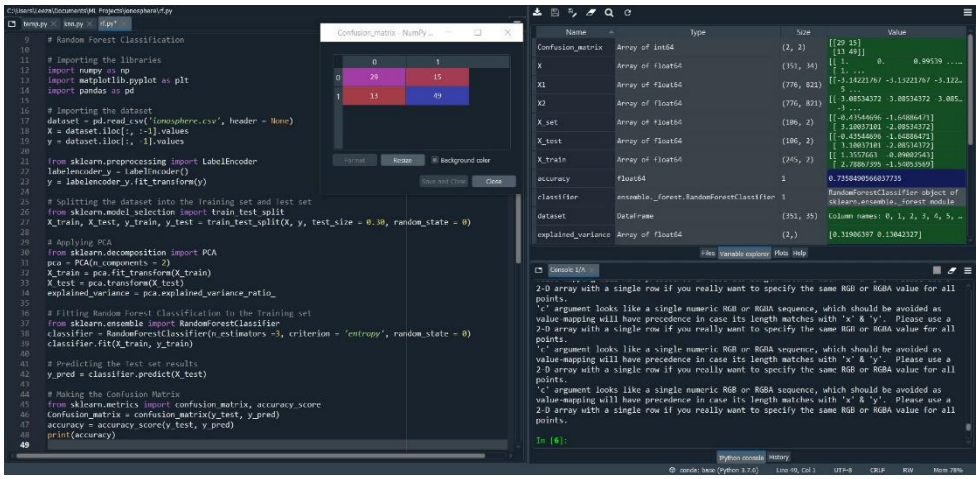


Fig. 3.1.1.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

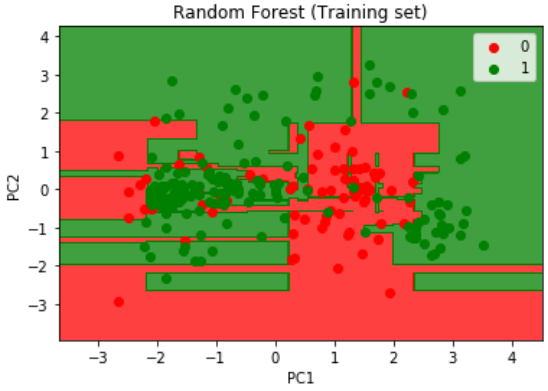


Fig. 3.1.1.b Training Set Graph

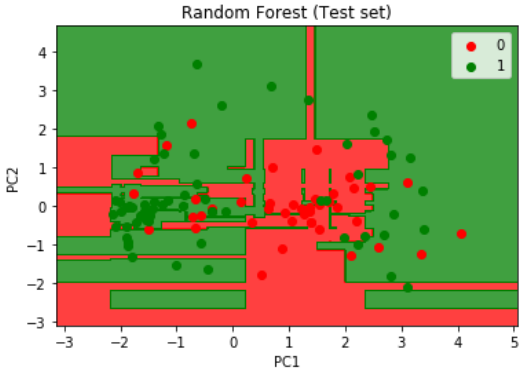


Fig. 3.1.1.c Test Set Graph

- **N_estimators: - 4**

  Accuracy: - 73.58%

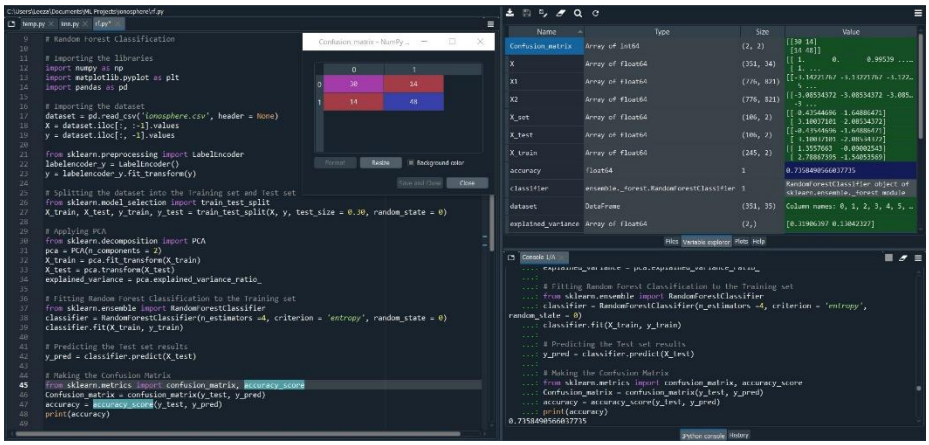


Fig. 3.1.2.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

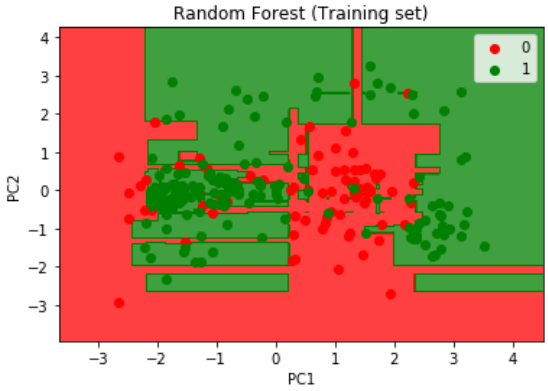


Fig. 3.1.2.b Training Set Graph

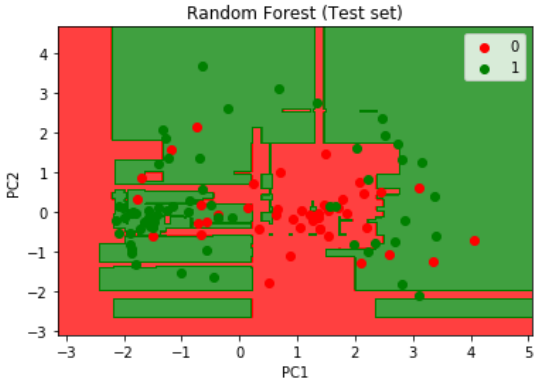
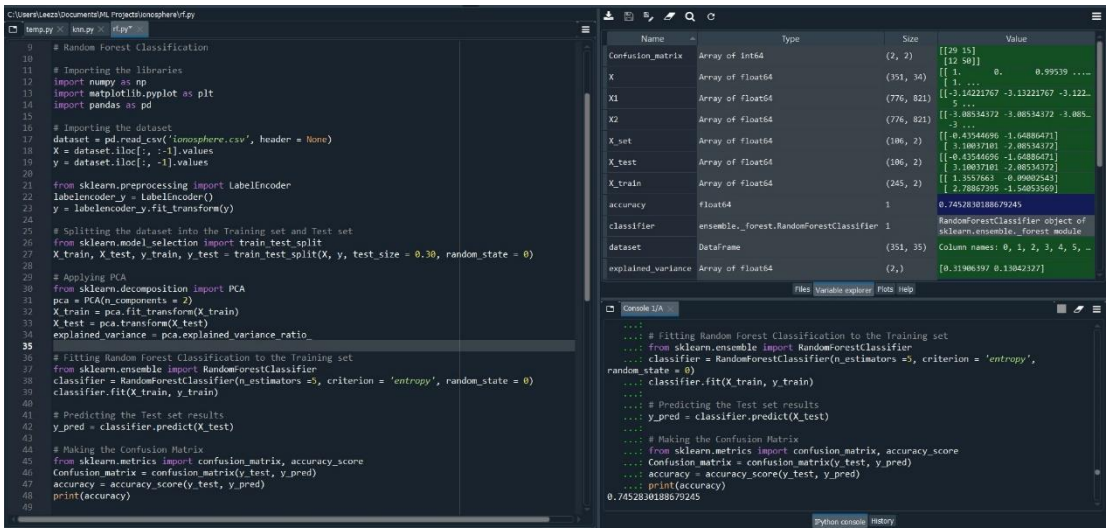


Fig. 3.1.2.c Test Set Graph

- **N_estimators: - 5**
  Accuracy: - 74.52%



```
# Random Forest Classification

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('ionosphere.csv', header = None)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 0)

# Applying PCA
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_

# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators =5, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
Confusion_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

```
...:
...: # Fitting Random Forest Classification to the Training set
...: from sklearn.ensemble import RandomForestClassifier
...: classifier = RandomForestClassifier(n_estimators =5, criterion = 'entropy',
random_state = 0)
...: classifier.fit(X_train, y_train)
...:
...: # Predicting the Test set results
...: y_pred = classifier.predict(X_test)
...:
...: # Making the Confusion Matrix
...: from sklearn.metrics import confusion_matrix, accuracy_score
...: Confusion_matrix = confusion_matrix(y_test, y_pred)
...: accuracy = accuracy_score(y_test, y_pred)
...: print(accuracy)
0.7452830188679245
```

Fig. 3.1.3.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

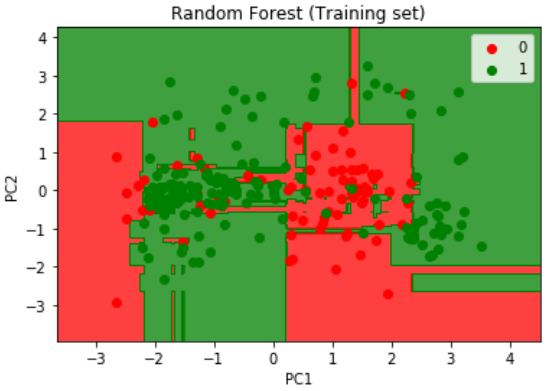


Fig. 3.1.3.b Training Set Graph

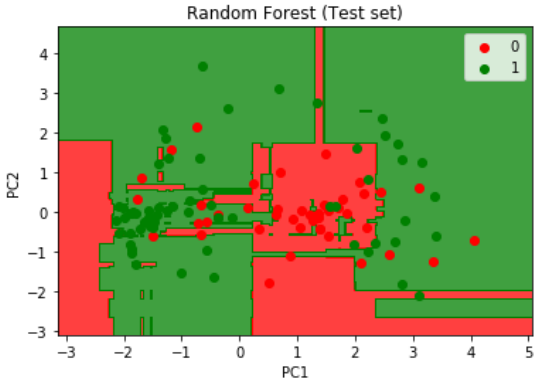


Fig. 3.1.3.c Test Set Graph

- **N_estimators: - 6**
  Accuracy: - 75.47%

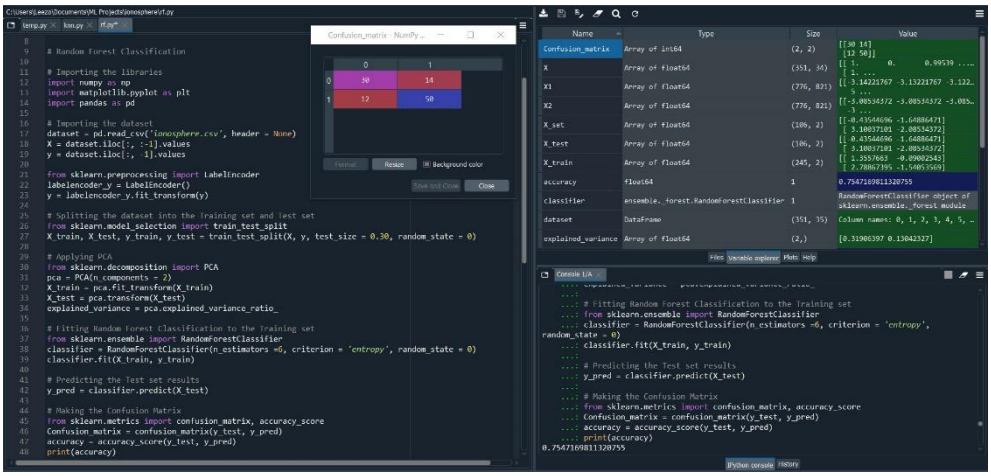


Fig. 3.1.4.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

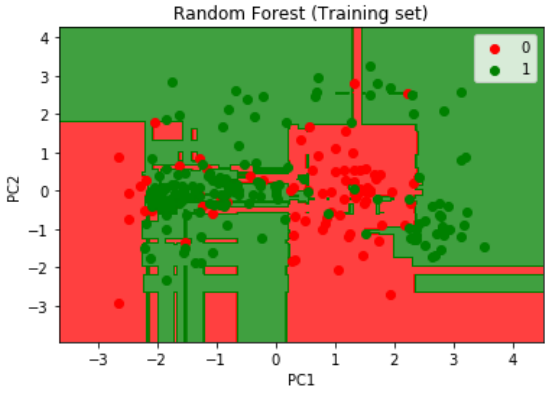


Fig. 3.1.4.b Training Set Graph

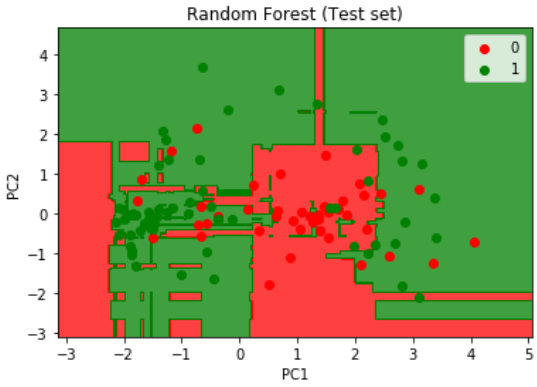


Fig. 3.1.4.c Test Set Graph

- **N_estimators: -7**
  Accuracy: - 75.47%

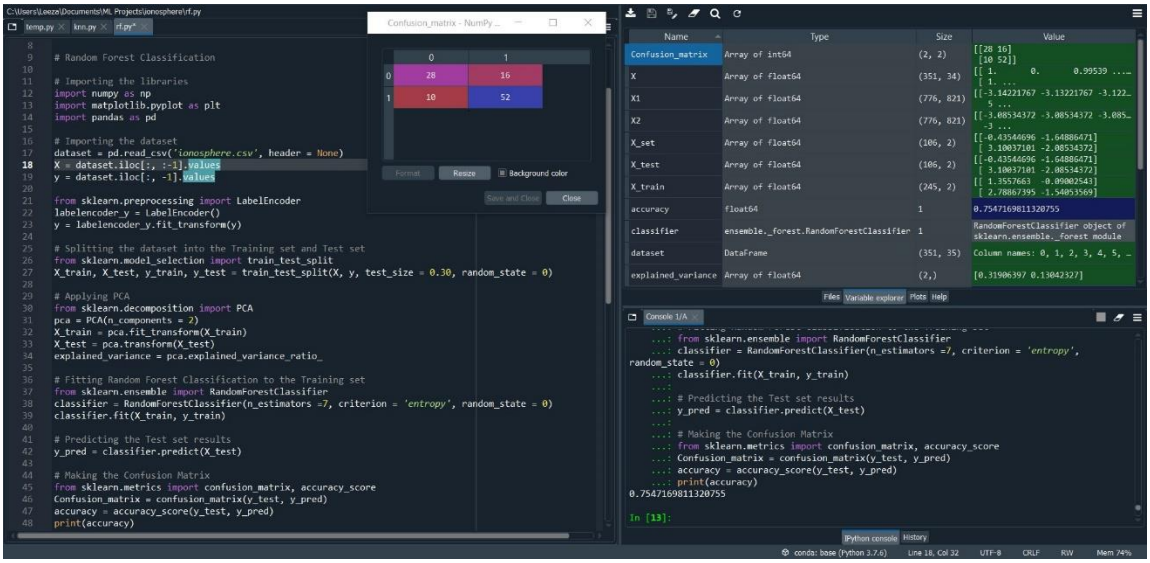


Fig. 3.1.5.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

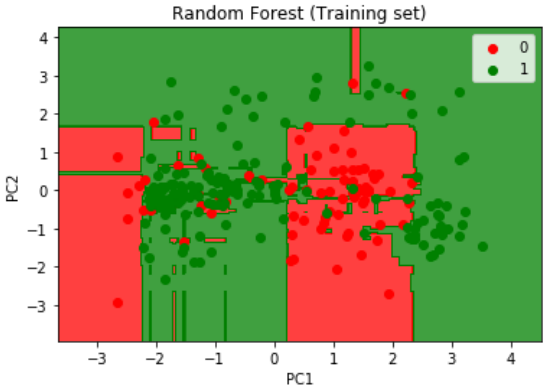


Fig. 3.1.5.b Training Set Graph

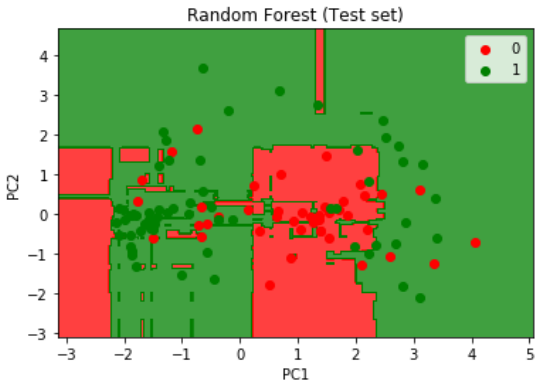


Fig. 3.1.5.c Test Set Graph

Conclusion: - The classification with test size = 30% gives best accuracy score = 75.47% with N_estimators = 6 and 7.

b. Test-size: - 40%

- **N_estimators: - 3**

  Accuracy: - 74.46%

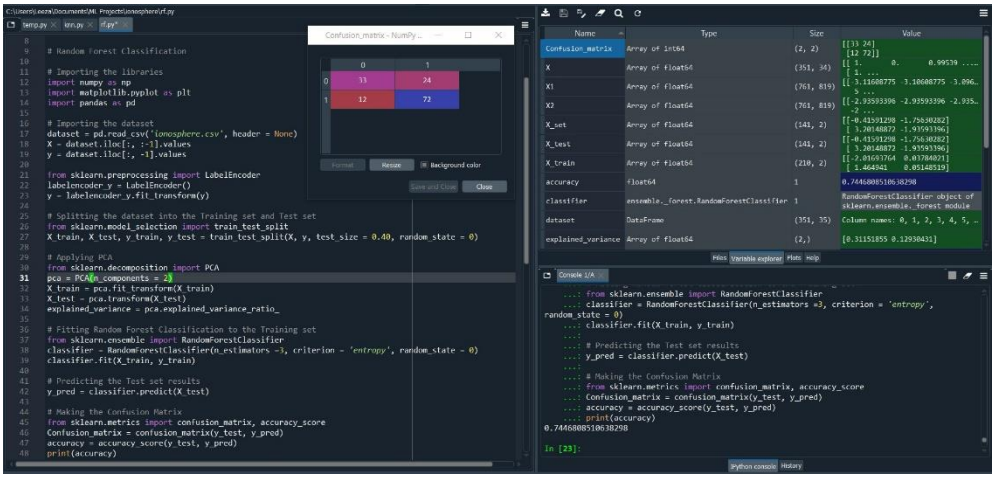


Fig. 3.2.1.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

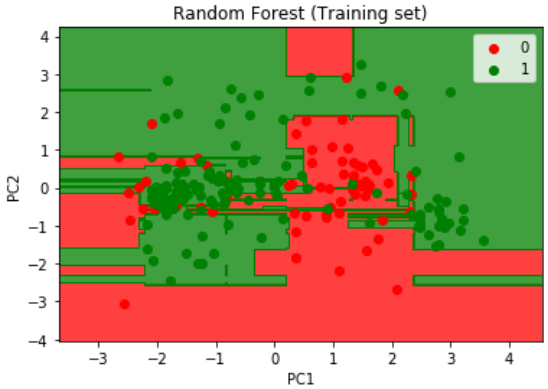


Fig. 3.2.1.b Training Set Graph

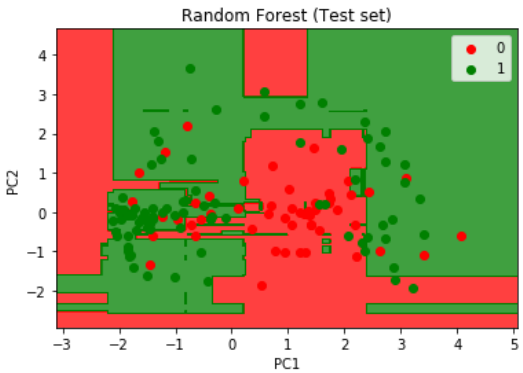


Fig. 3.2.1.c Test Set Graph

- **N_estimators: - 4**
  Accuracy: - 74.46%

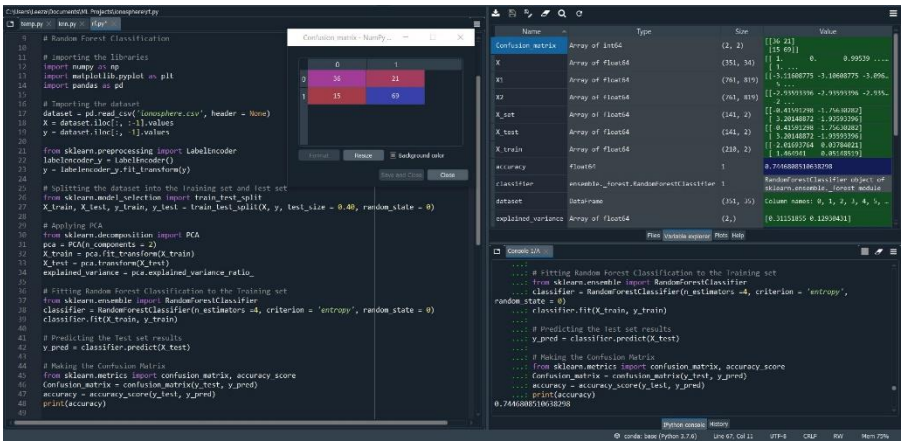


Fig. 3.2.2.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

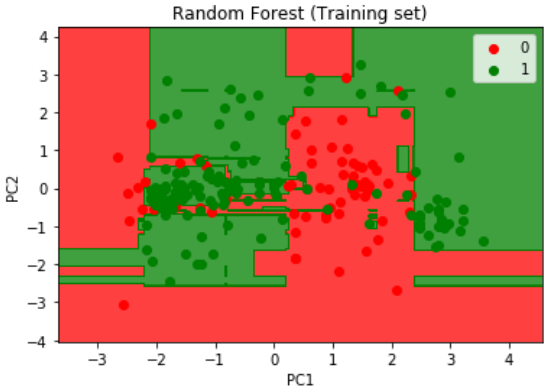


Fig. 3.2.2.b Training Set Graph

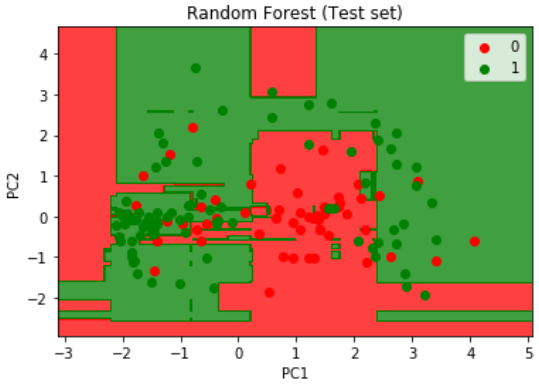


Fig. 3.2.2.c Test Set Graph

- **N_estimators: - 5**
  Accuracy: - 78.01%

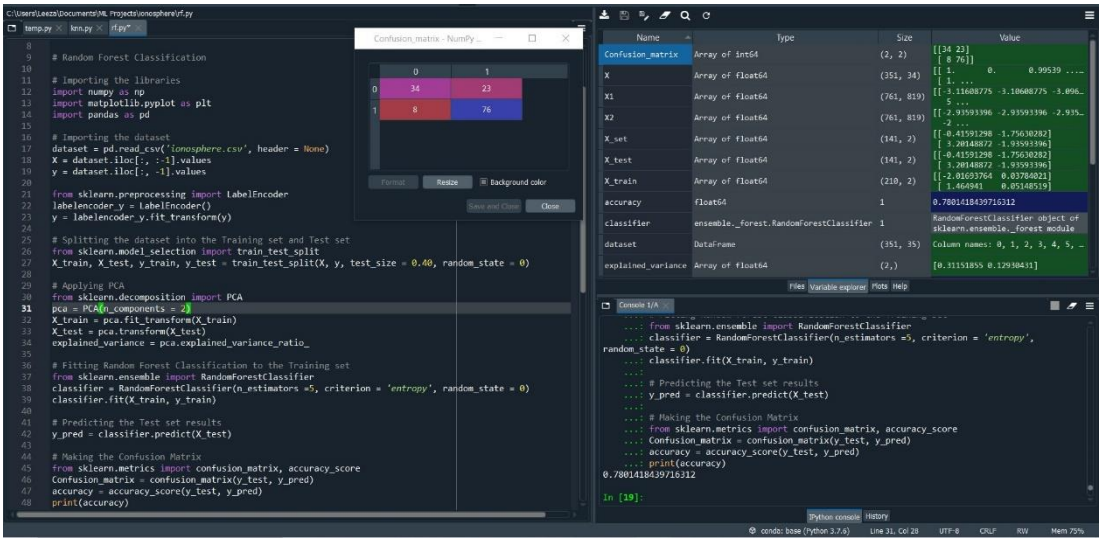


Fig. 3.2.3.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

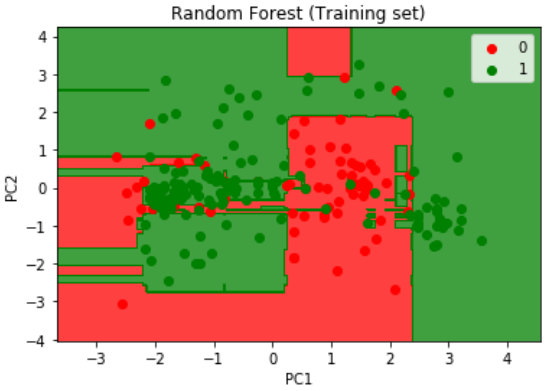


Fig. 3.2.3.b Training Set Graph

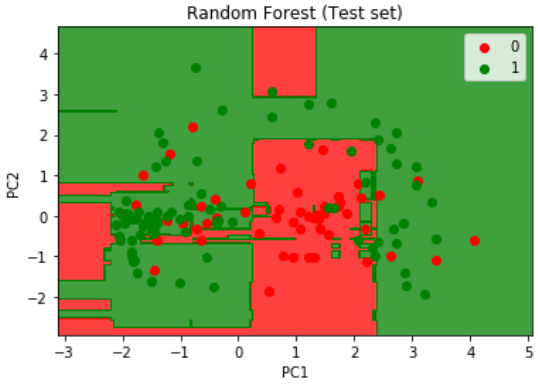


Fig. 3.2.3.c Test Set Graph

- **N_estimators: - 6**
  Accuracy: - 78.72%

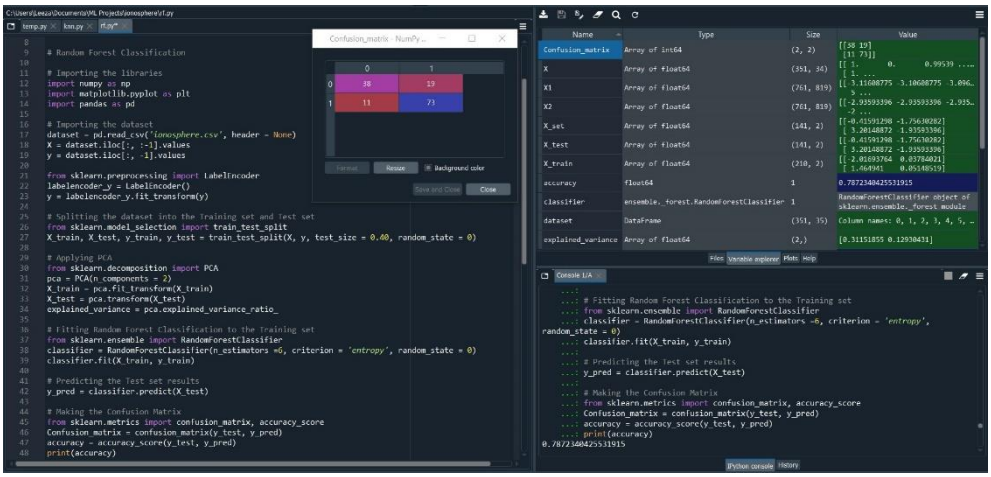


Fig. 3.2.4.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

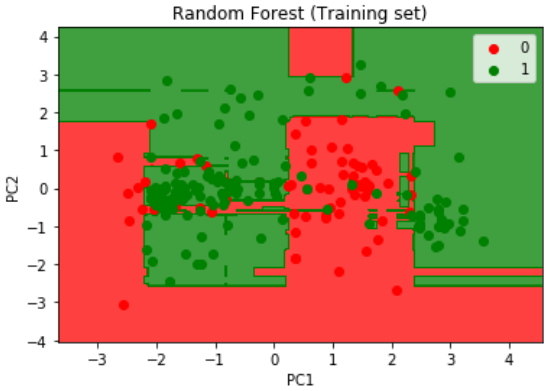


Fig. 3.2.4.b Training Set Graph

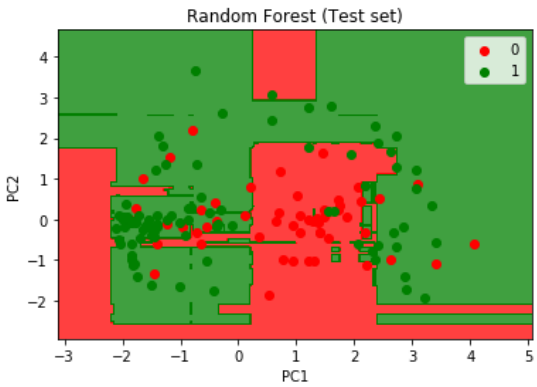


Fig. 3.2.4.c Test Set Graph

- **N_estimators: -7**
  Accuracy: - 76.59%

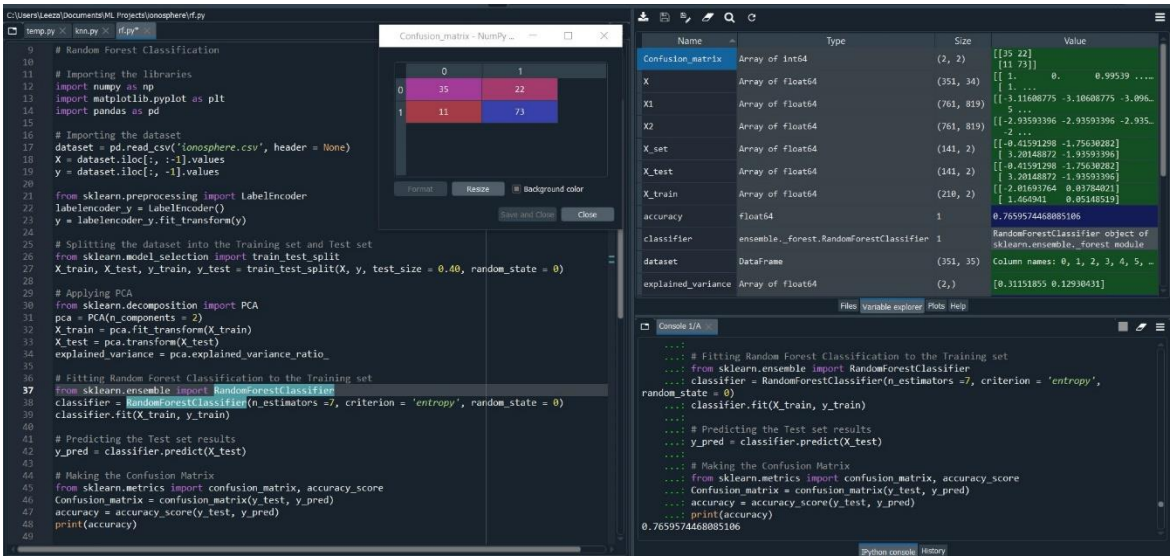


Fig. 3.2.5.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

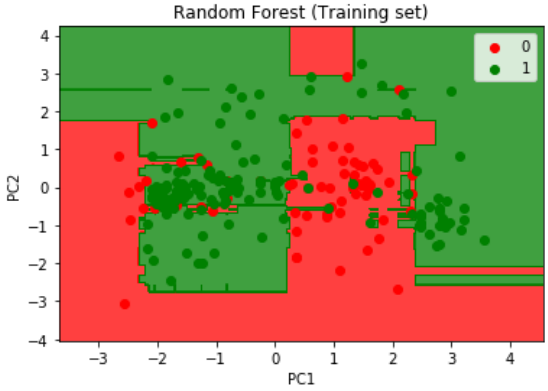


Fig. 3.2.5.b Training Set Graph

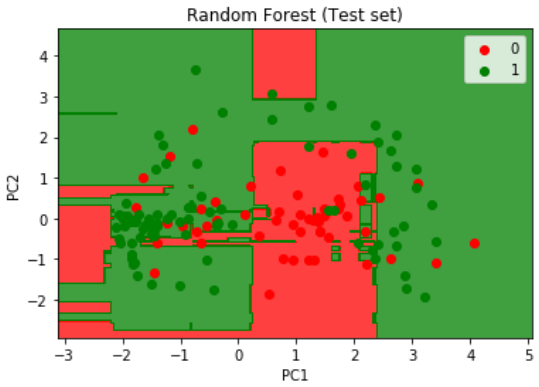


Fig. 3.2.5.c Test Set Graph

Conclusion: - The classification with test size = 30% gives best accuracy score = 78.72% with N_estimators = 6.

c. Test-size: - 50%

- **N_estimators: - 3**

  Accuracy: - 75%

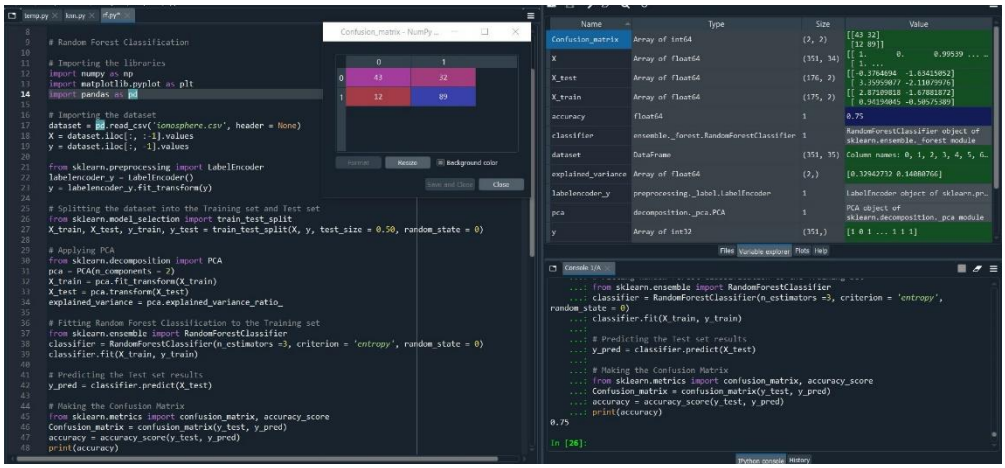


Fig. 3.3.1.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

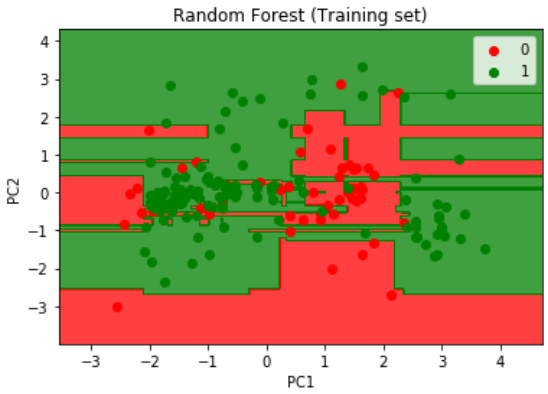


Fig. 3.3.1.b Training Set Graph

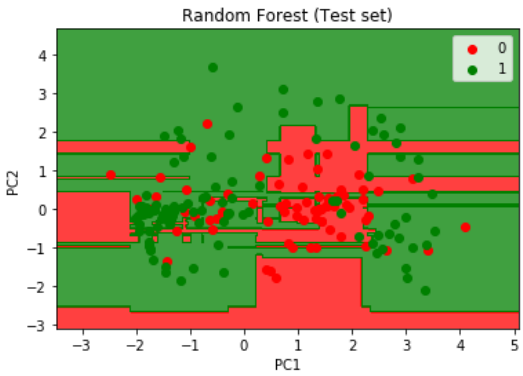


Fig. 3.3.1.c Test Set Graph

- **N_estimators: - 4**

  Accuracy: - 76.70%

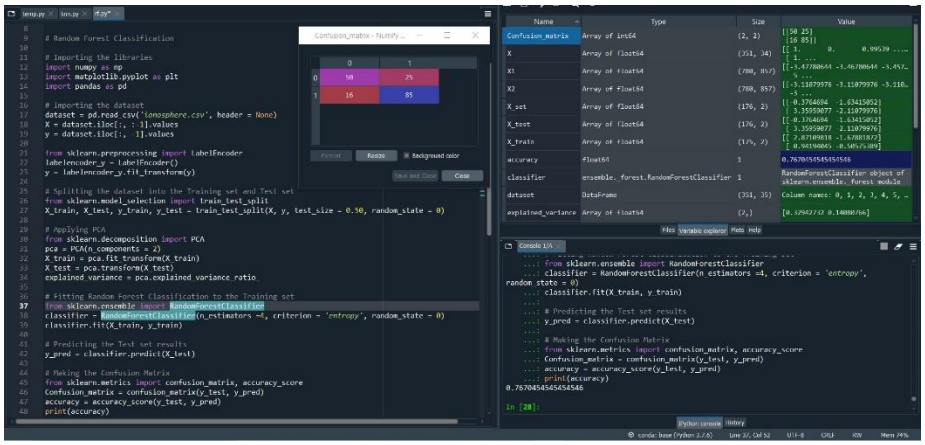


Fig. 3.3.2.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

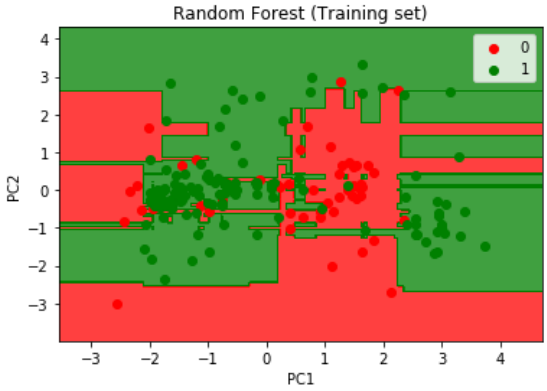


Fig. 3.3.2.b Training Set Graph

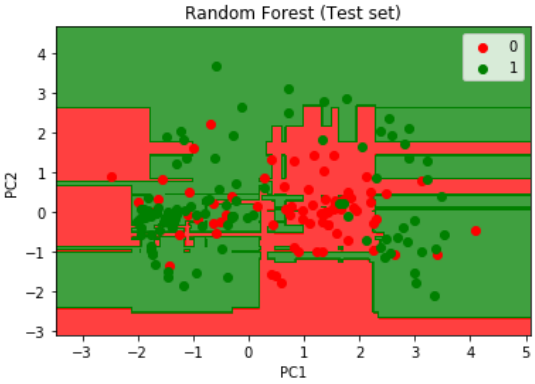


Fig. 3.3.2.c Test Set Graph

- **N_estimators: - 5**
  Accuracy: - 76.13%

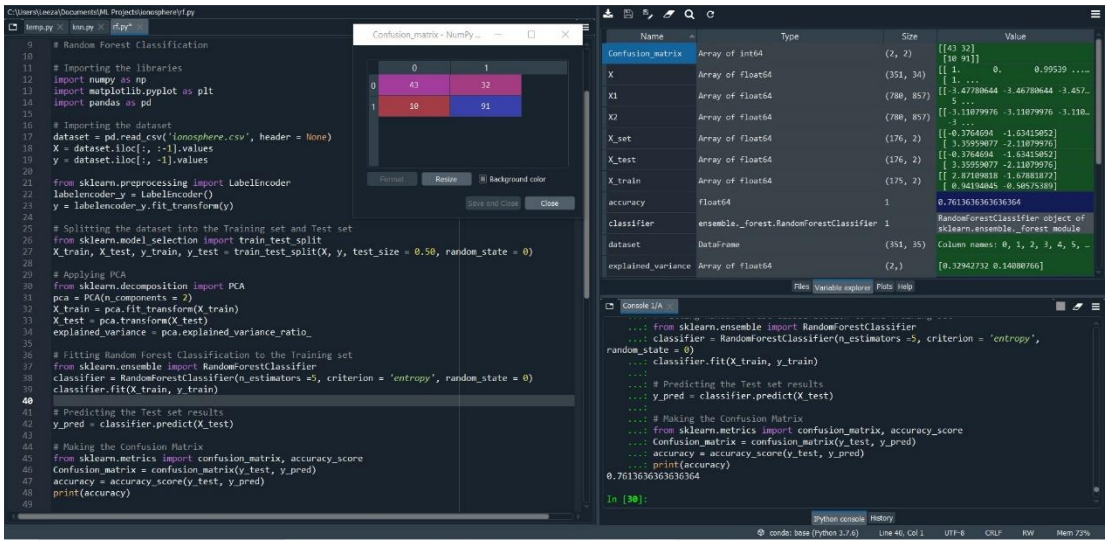


Fig. 3.3.3.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

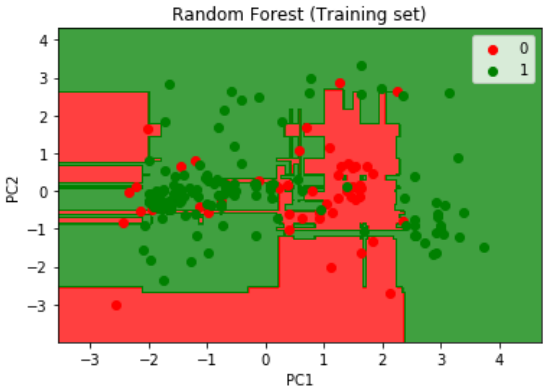


Fig. 3.3.3.b Training Set Graph

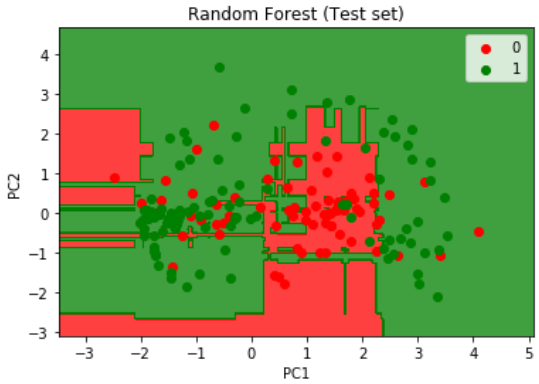


Fig. 3.3.3.c Test Set Graph

- **N_estimators: - 6**
  Accuracy: - 78.97%

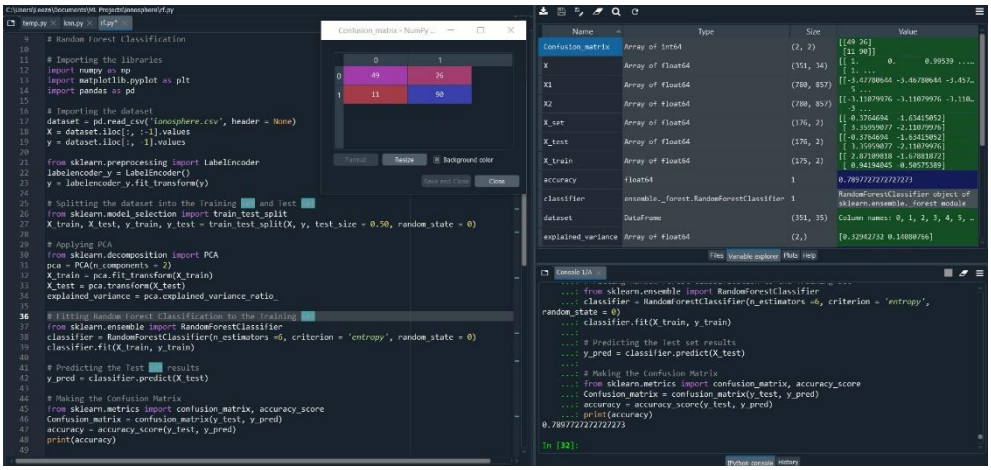


Fig. 3.3.4.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

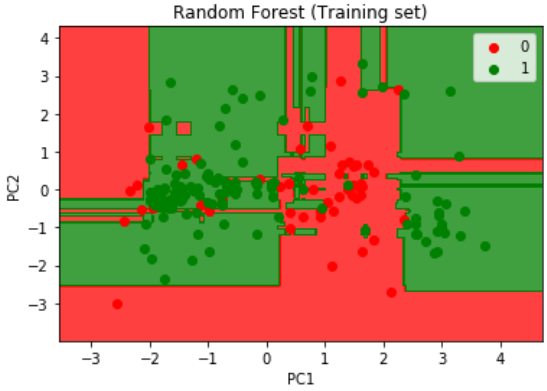


Fig. 3.3.4.b Training Set Graph

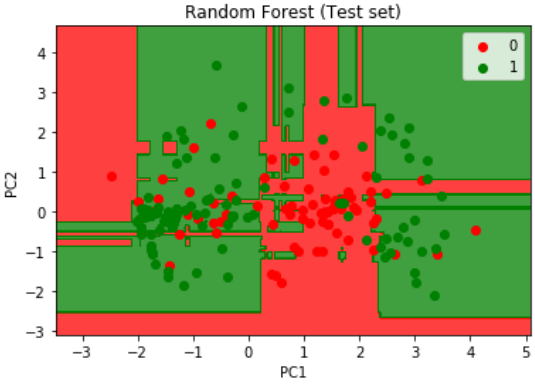


Fig. 3.3.4.c Test Set Graph

- **N_estimators: -7**
  Accuracy: - 79.54%

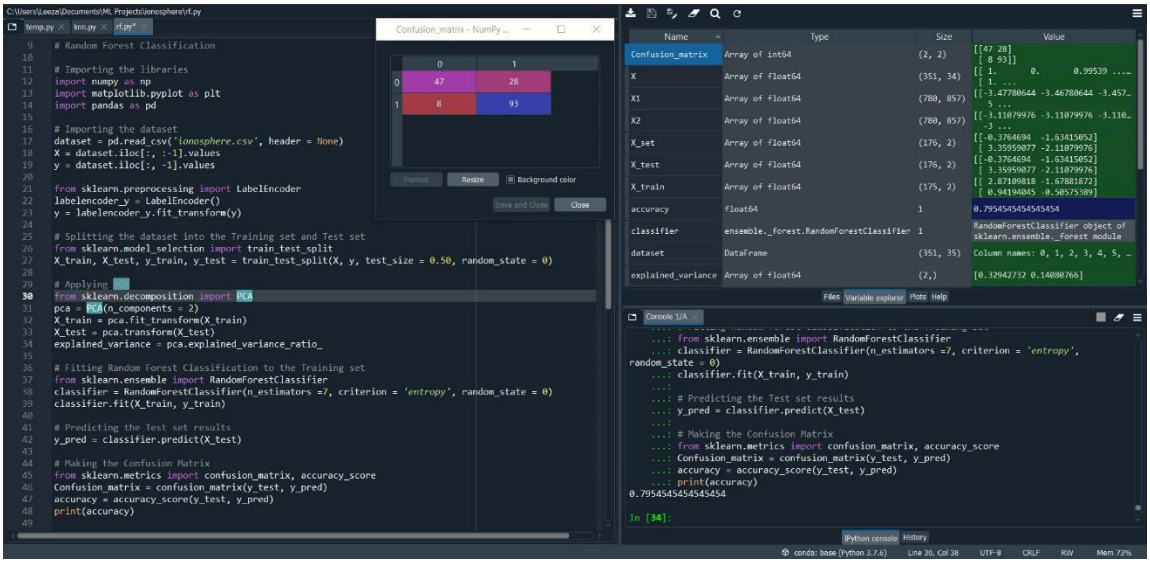


Fig. 3.3.5.a Snapshot of Error free code, confusion matrix and Accuracy in console log.

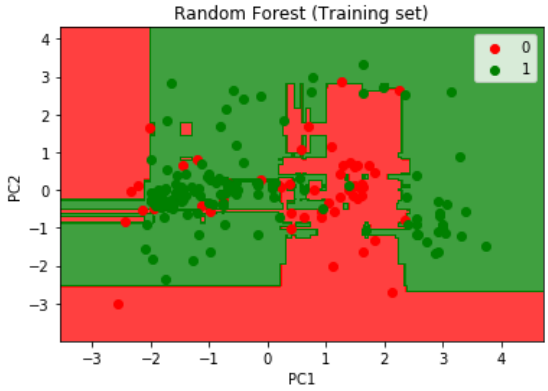


Fig. 3.3.5.b Training Set Graph

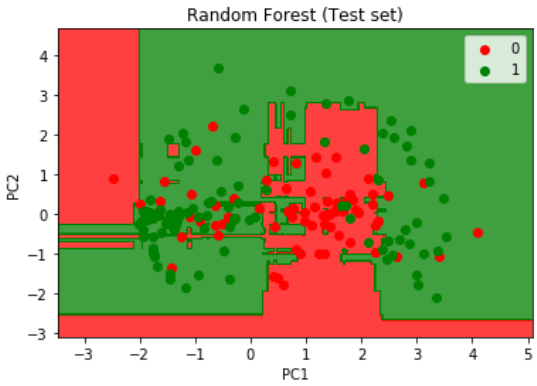


Fig. 3.3.5.c Test Set Graph

Conclusion: - The classification with test size = 30% gives best accuracy score = 79.54% with N_estimators = 7.

OBSERVATIONS: - The following points have been concluded from the above analysis of data using various classifier models: -

- With K nearest neighbour classifier model the best accuracy score is yielded with test size = 50% and k-value = 4 . **ACCURACY_SCORE – 82.38%**
- With decision tree classifier model the best accuracy score is yielded with test size = 50% and max_depth = 3. **ACCURACY_SCORE – 82.29%**
- With random forest classifier model the best accuracy score is yielded with test size = 50% and n_estimator = 7. **ACCURACY_SCORE – 79.54%**

Hence, the best classification model considering the above constraint values is K-Nearest Neighbours with accuracy of 82.38 %, test size = 50 % for k-value = 4.

NEED FOR APPLYING PCA: - The above models have been implemented with PCA (Principle Component Analysis) in order to visualize the models graphically. Since the original dataset had 34 features so it's not possible to plot them so we used PCA with n_components = 2 to have a plot for the dataset as it's easy to conclude from the visualization of dataset.

CONFUSION MATRIX: - A confusion matrix is a matrix (table) that can be used to measure the performance of a machine learning algorithm, usually a supervised learning one. Each row of the confusion matrix represents the instances of an actual class and each column represents the instances of a predicted class.

	Class 1 Predicted	*Class 2 Predicted*
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

TP: True Positive

FP: False Positive

FN: - False Negative

TN: - True Negative

Fig. 1.d Confusion Matrix

ACCURACY SCORE: -

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Fraction predicted correctly

REFERENCES: -

- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- https://www.geeksforgeeks.org/decision-tree/
- https://en.wikipedia.org/wiki/Random_forest
- https://medium.com/@sonish.sivarajkumar/k-nearest-neighbours-knn-algorithm-9900c1427726
- https://www.datacamp.com/community/tutorials/decision-tree-classification-python
- https://medium.com/analytics-vidhya/machine-learning-decision-trees-and-random-forest-classifiers-81422887a544
- https://www.python-course.eu/confusion_matrix.php#:~:text=A%20confusion%20matrix%20is%20a,instances%20of%20a%20predicted%20class.
- https://towardsdatascience.com/understanding-data-science-classification-metrics-in-scikit-learn-in-python-3bc336865019
- https://www.geeksforgeeks.org/confusion-matrix-machine-learning/