First Name: Harish

Last Name: Marepalli

SJSU ID: 016707314

Professor: Dr. Younghee Park

Improving Accuracy Using Random Forest Algorithm

Improving accuracy using Random Forest Algorithm:

Here, we have been given two datasets, Train_data.csv and Test_data.csv. Train_data.csv is used here to train the model on how to detect whether an incoming traffic is normal or anomaly. In these datasets we there are so many columns. The Train_data.csv contains 42 columns and the Test_data.csv contains 41 columns. The difference of one column is the ‘Class’ column, which is used to detect normal and anomaly traffic.

Here, we were given the code to find the Random Forest algorithms output. By using the normal approach, the output achieved was 94% and we have to improve it to 98-99% by making modifications.

The procedure that I followed in this activity is I have installed jupyter labs and written the code in python notebook file. By using the normal code, I got the output as 94.7%.

First, to improve the accuracy, I have included 3 more columns of train dataset in the code and these columns are ‘count’, ‘srv_count’, and ‘logged_in’. I have chosen only these columns is because I felt that these may contribute to the future prediction efficiently because remaining other column values are just 0’s and 1’s, which does not much contribute to the future prediction.

Some of the other columns might also contribute to the accurate prediction but we do not include more columns as it will be an overload/overfitting case. So, this is the reason I have included these columns.

Second, I have splitted the dataset into 70% and 30% to get the accurate result.

Third, in the modeling section, I have increased the max_depth from 2 to 7. The default value of number of trees is 100. The value of max_depth indicates the depth that we traverse for a tree. If the value of it is less, it means we are not deeply going through the random forest code. If the value is more, then we are diving deep into the process and can know how the process works and also the accurate output.

I have tried all the values of max depth from 2 to 7 and stopped at 7 because it is considered to be the limit. If the value is more than 7, then it will be considered as overfitting, which we don’t want to do.

After making all these changes in the code, the output that I achieved is 0.9886213283937549 and that comes to be 98.8% which can be approximated to **99%**.

I have also written a few lines to display the confusion matrix and the matrix is as follows:

```
[[4017,     18],
 [  68,   3455]]
```

The scheme of confusion matrix is:

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Here, the number of True positive (TP) values are 4017.
The number of False negative (FN) values are 18.
The number of False positive (FP) values are 68.
The number of True negative (TN) values are 3455.

Code is provided in the Appendix.

Snippets:
1. Figure 1 shows the output before the improvement.

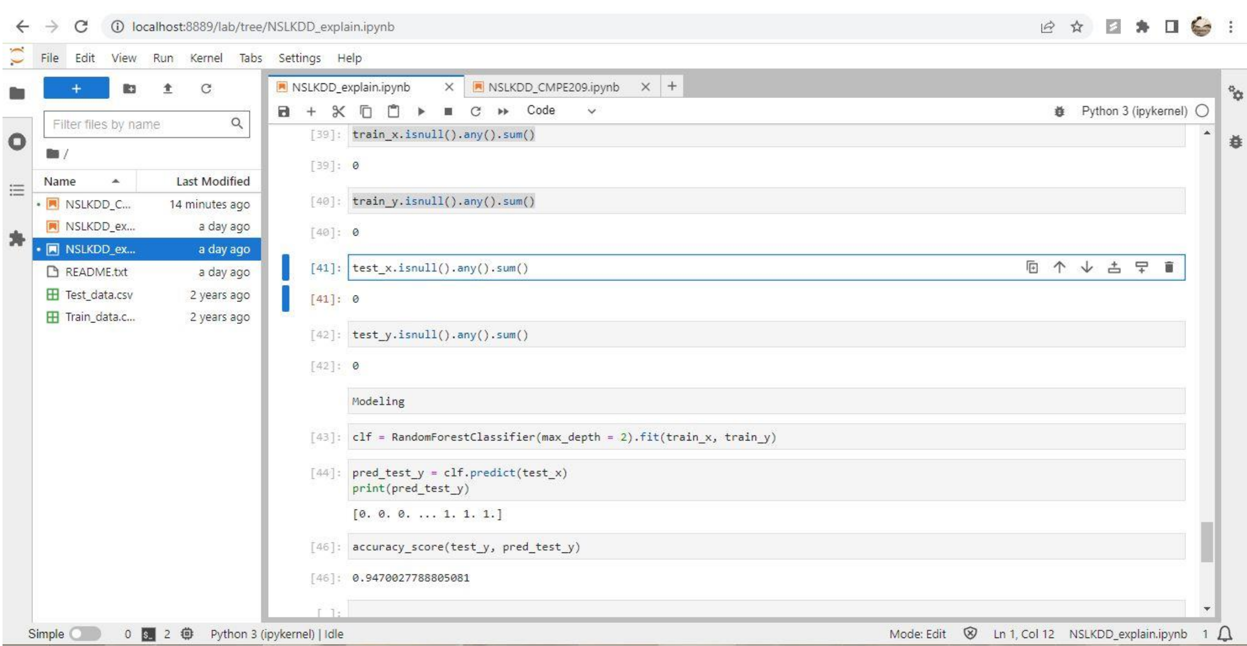

Fig. 1: Output before improvement

2. Figure 2 shows the improved code.

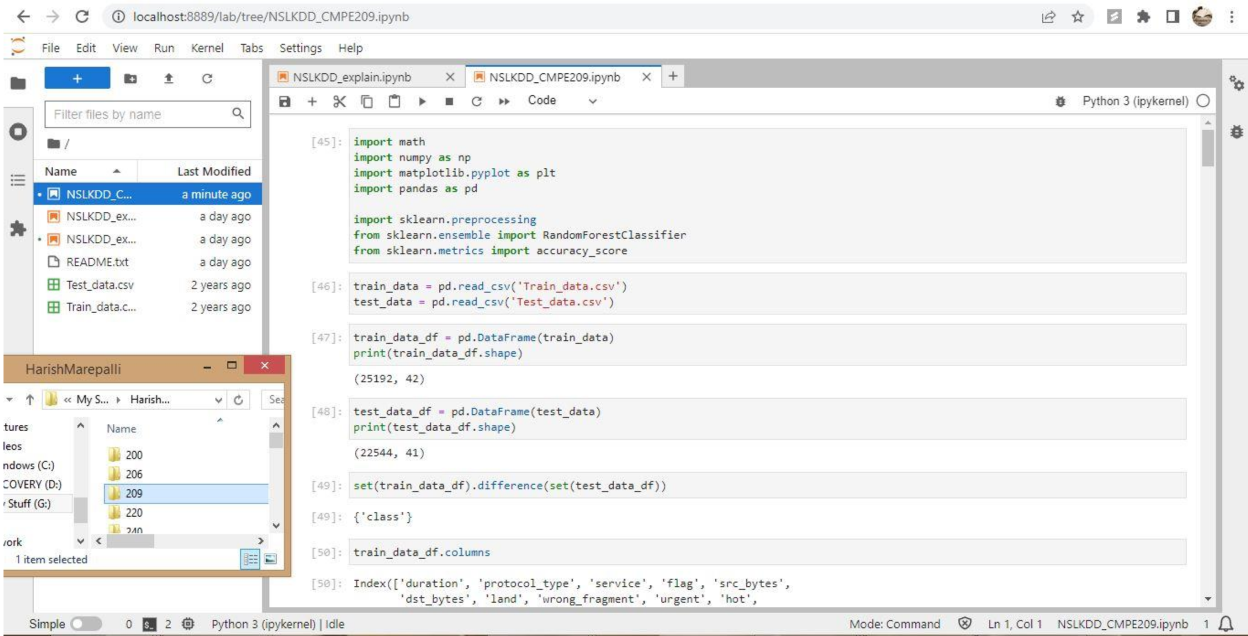

Fig. 2: Improved code 1

3. Figure 3 shows the continued code.

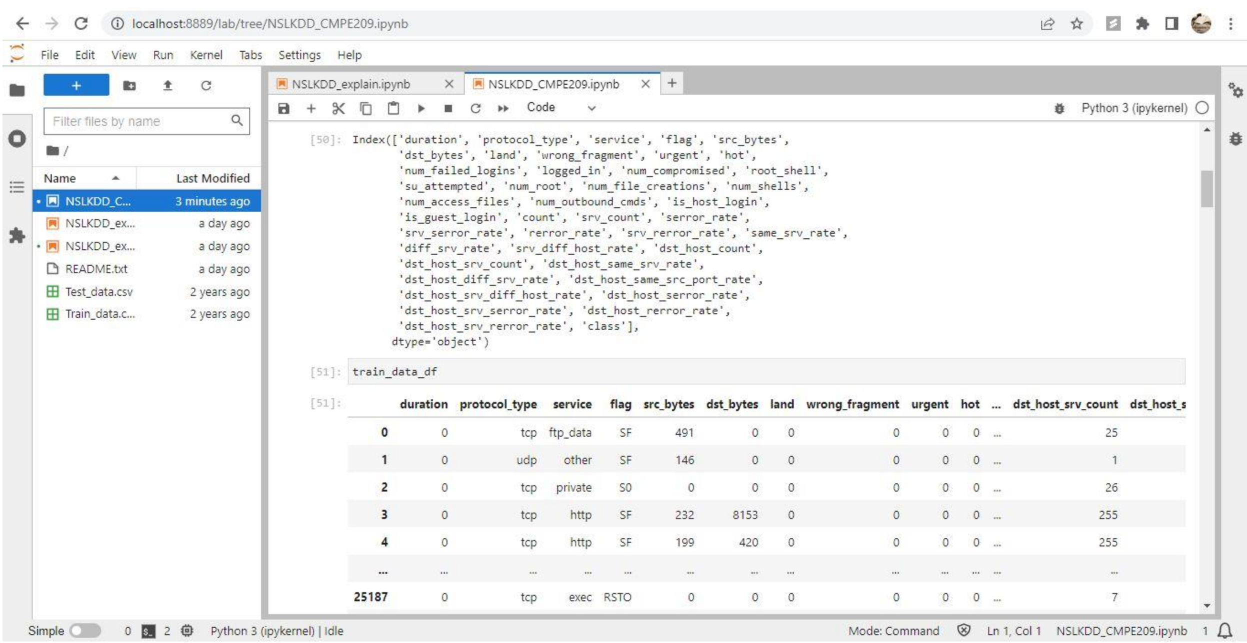

Fig. 3: Improved code 2

4. Figure 4 shows the continued code.

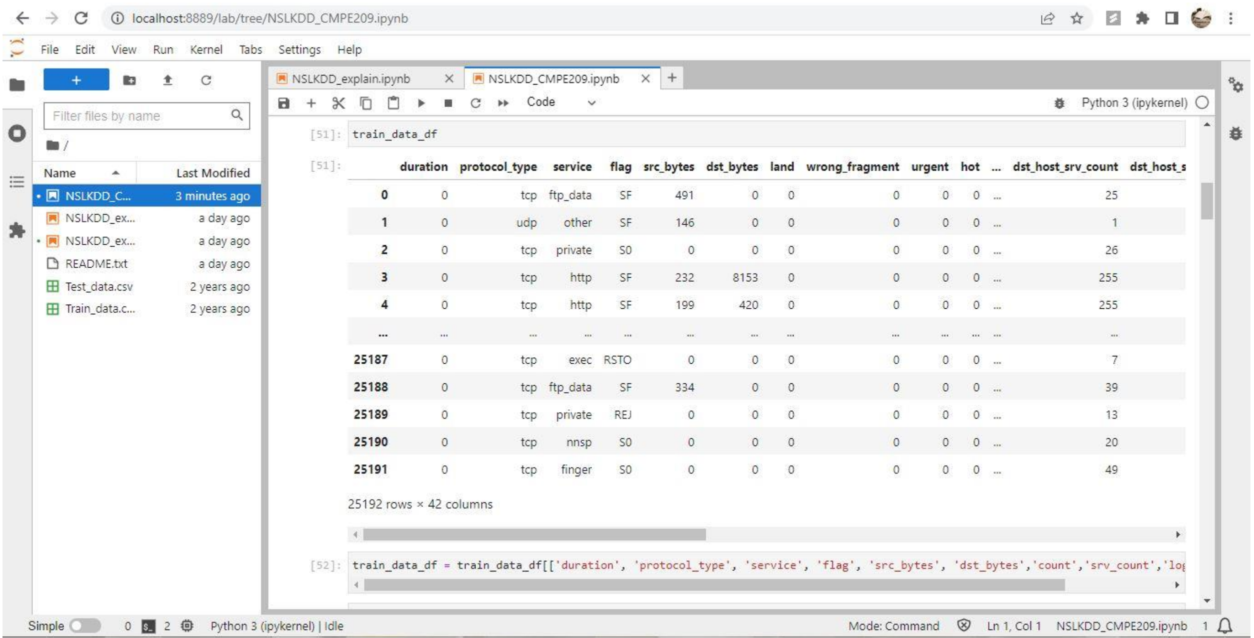

Fig. 4: Improved code 3

5. Figure 5 shows the added columns.

Fig. 5: Added columns

6. Figure 6 shows the continued code.

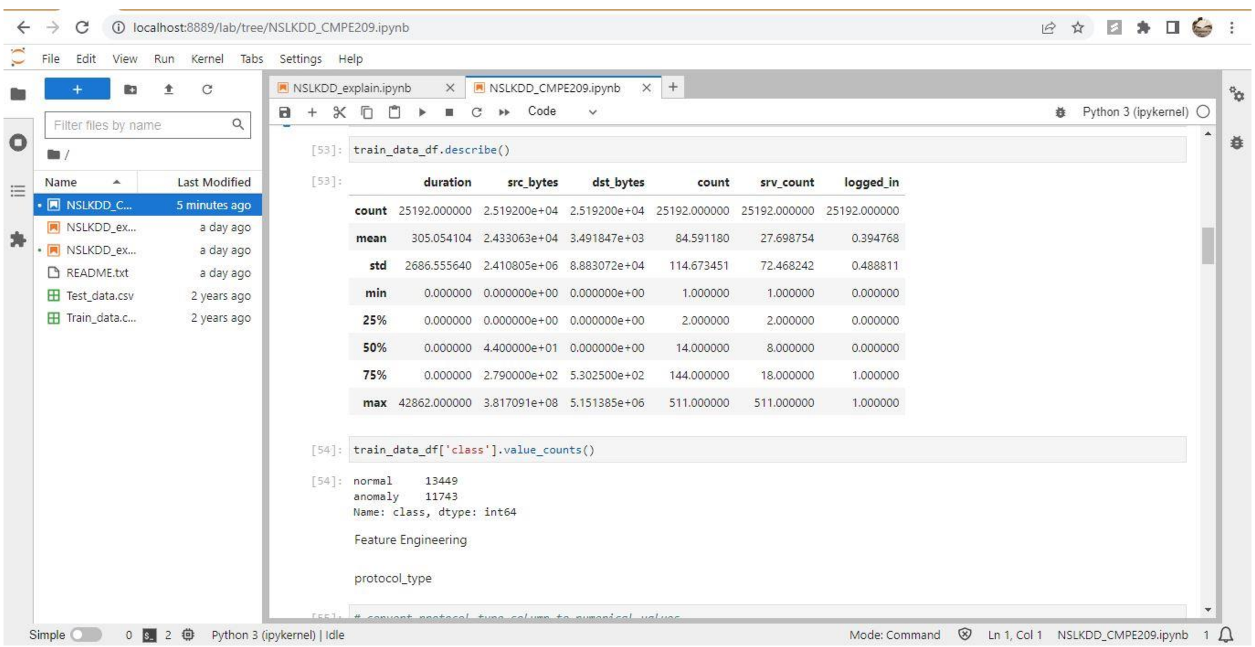

Fig. 6: Improved code 4

7. Figure 7 shows the continued code.

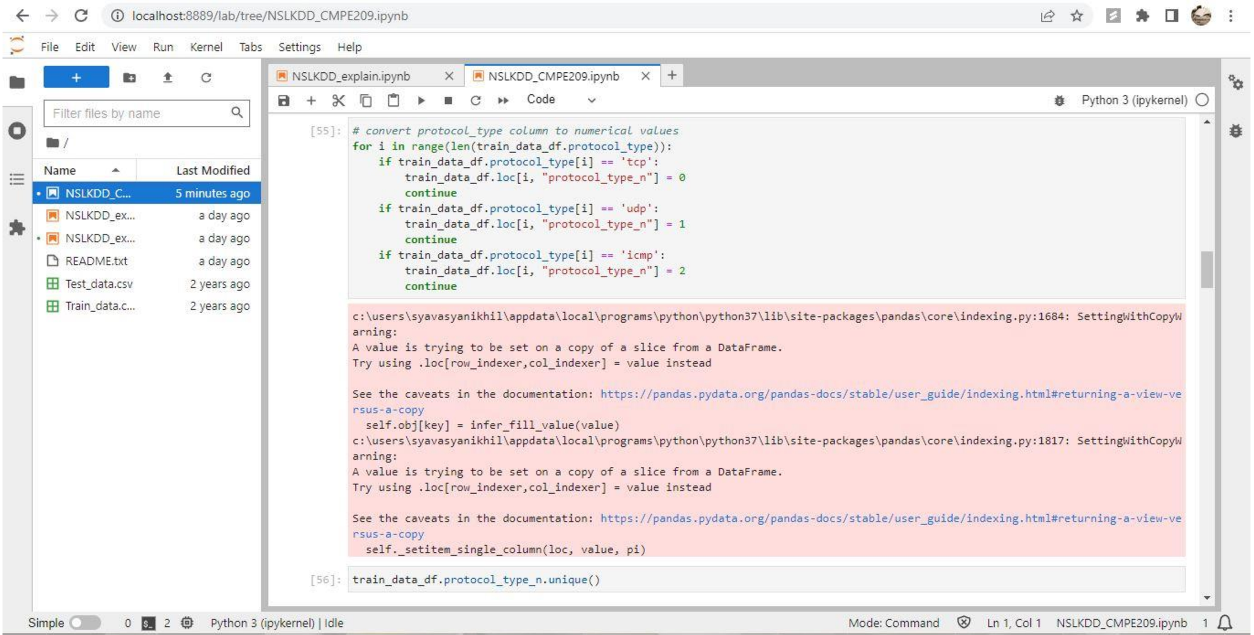

Fig. 7: Improved code 5

8. Figure 8 shows the continued code.

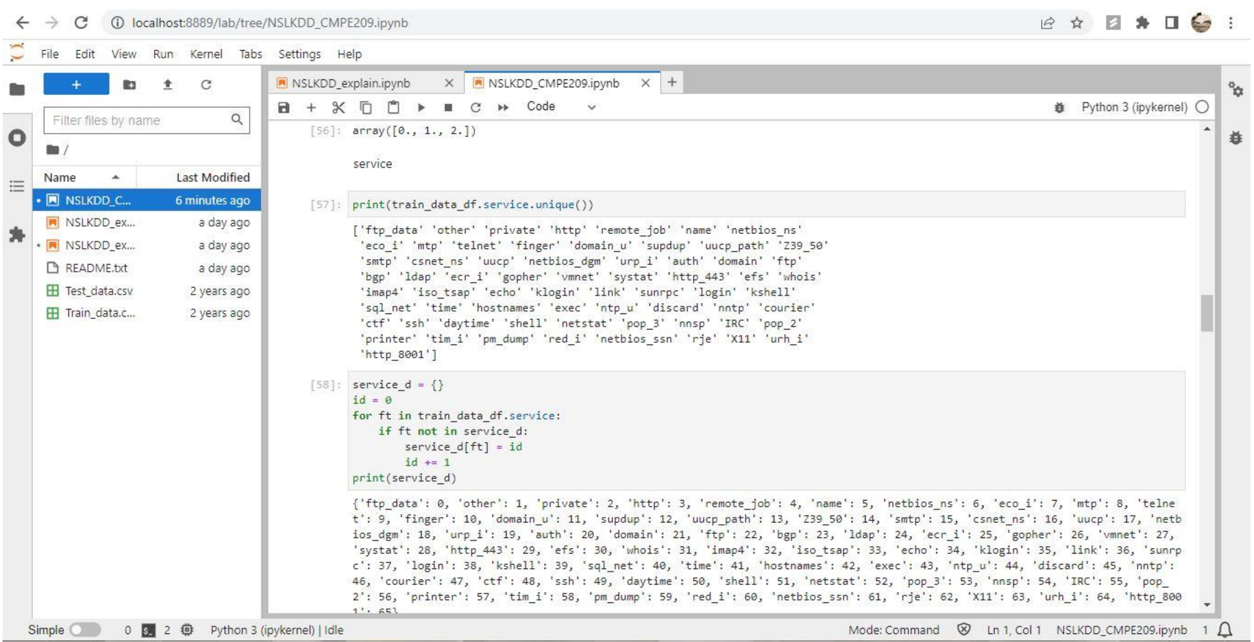

Fig. 8: Improved code 6

9. Figure 9 shows the continued code.

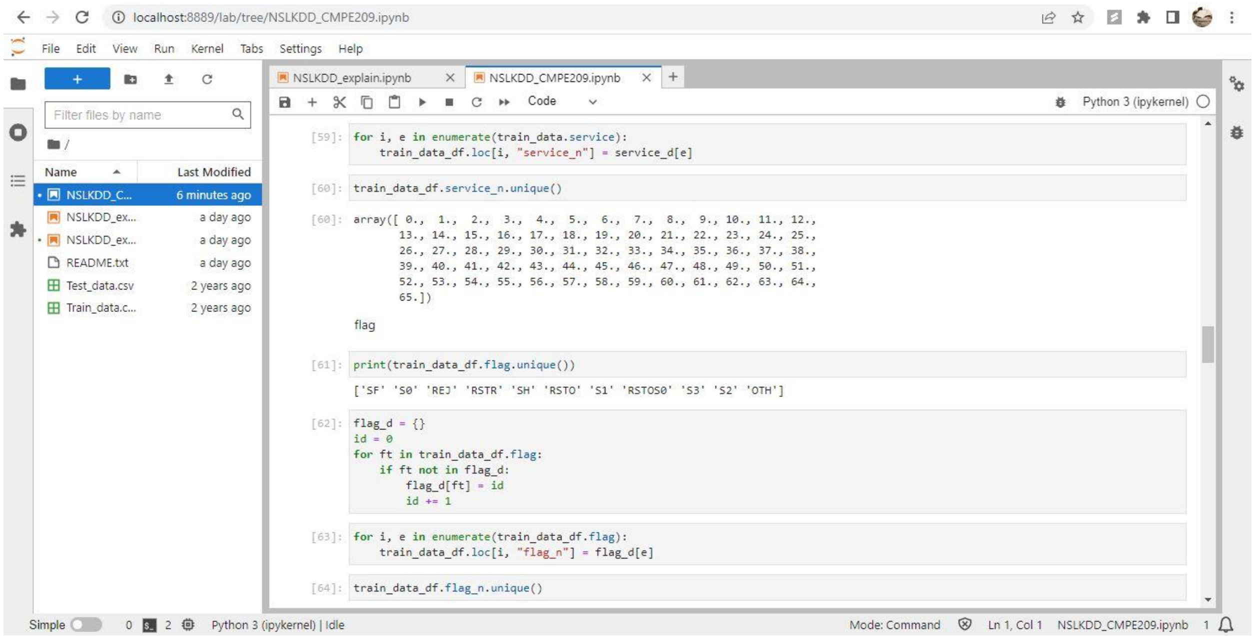

Fig. 9: Improved code 7

10. Figure 10 shows the continued code.

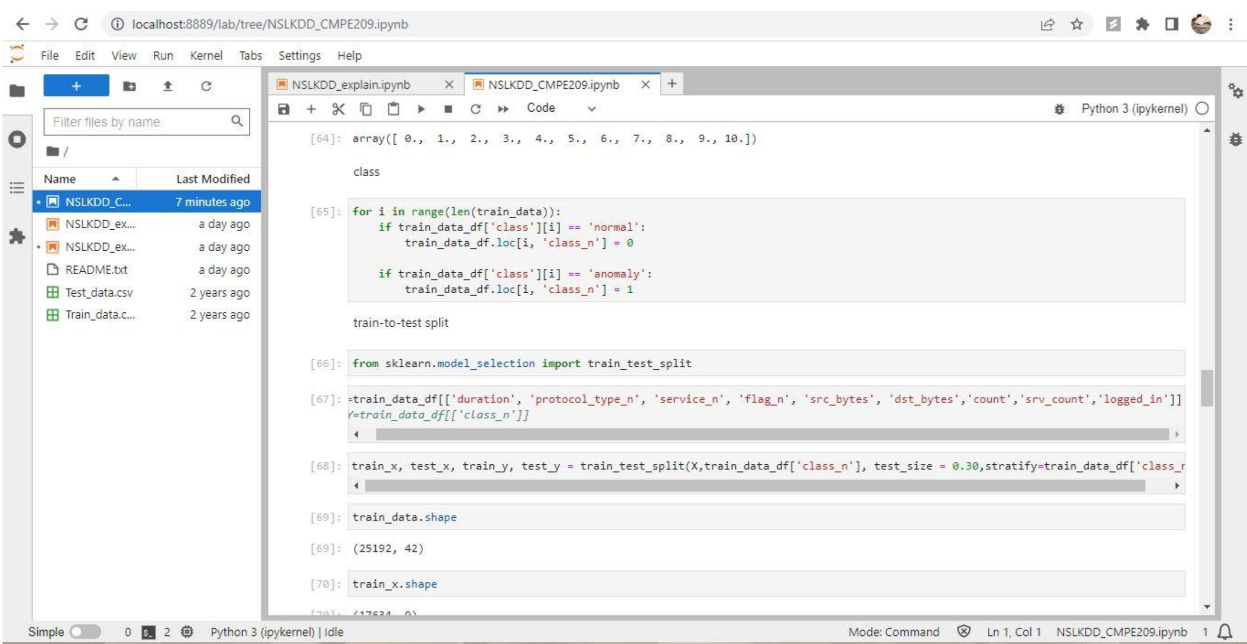

Fig. 10: Improved code 8

11. Figure 11 shows the continued code.

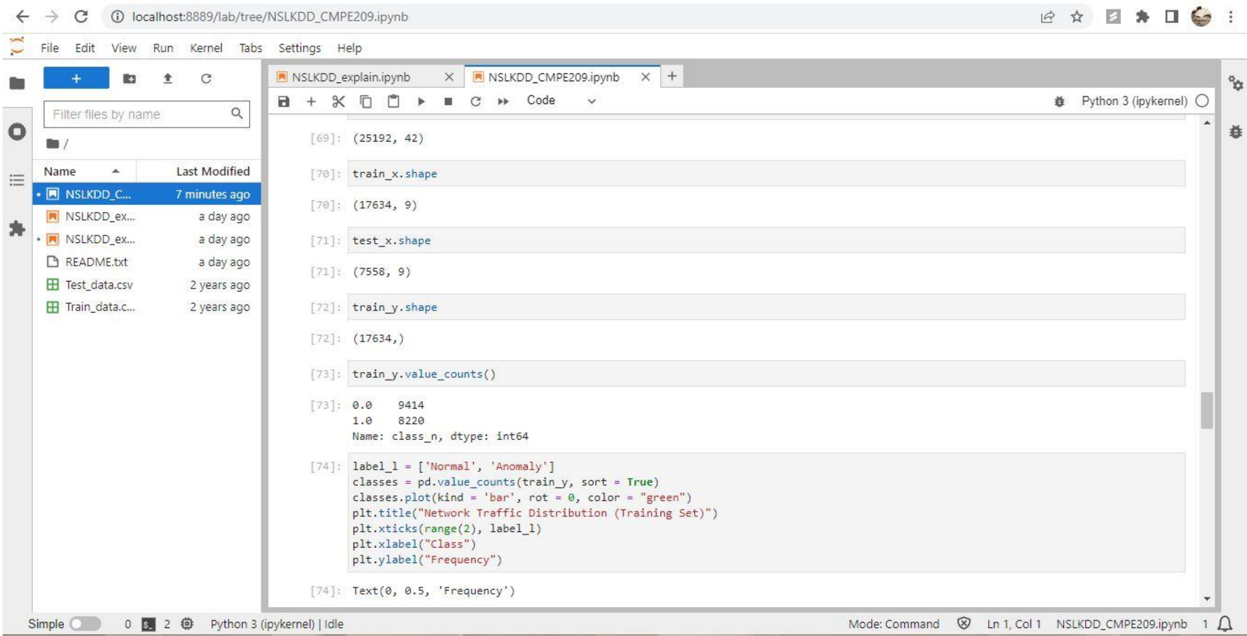

Fig. 11: Improved code 9

12. Figure 12 shows the continued code.

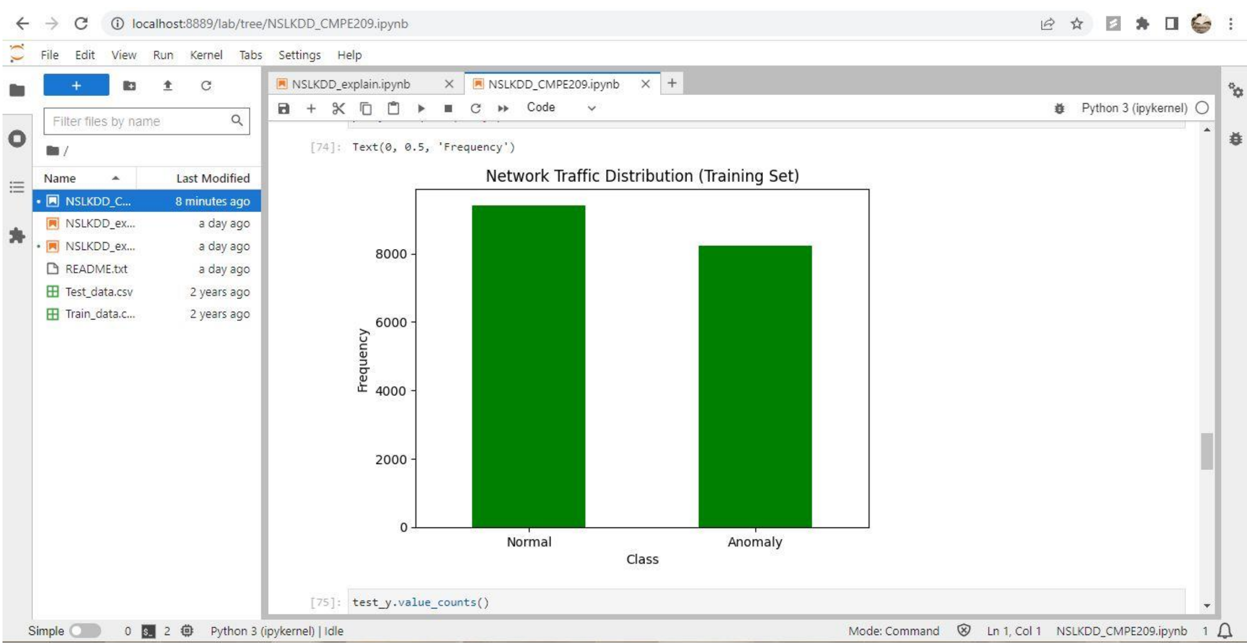

Fig. 12: Improved code 10

13. Figure 13 shows the continued code.

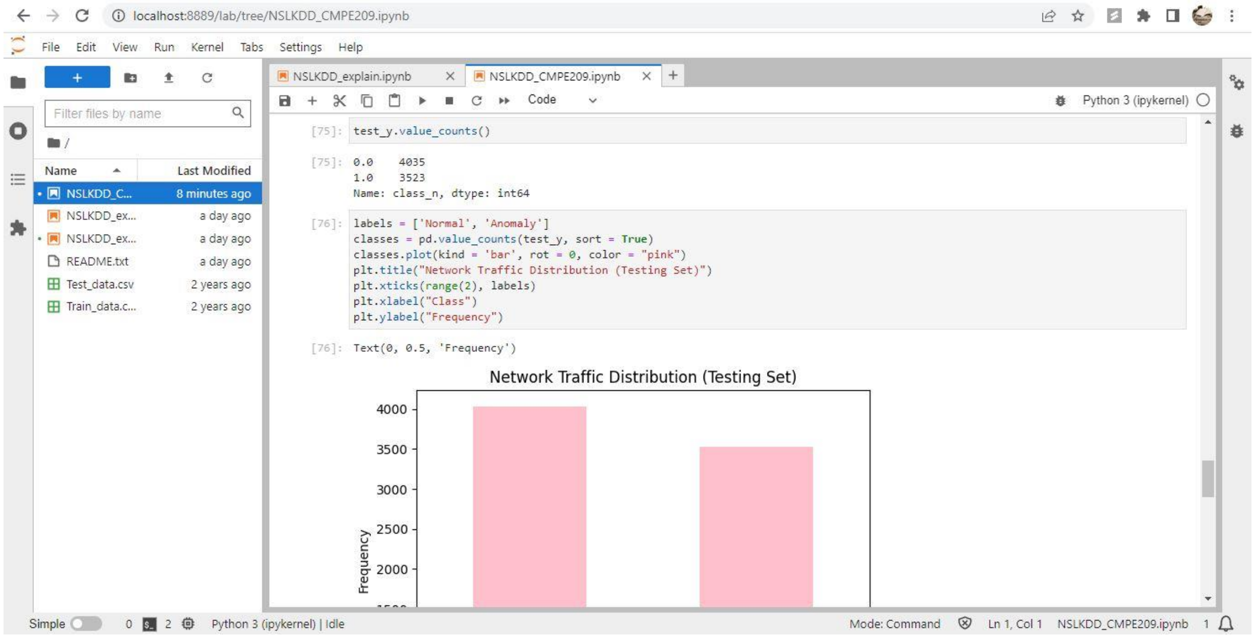

Fig. 13: Improved code 11

14. Figure 14 shows the continued code.

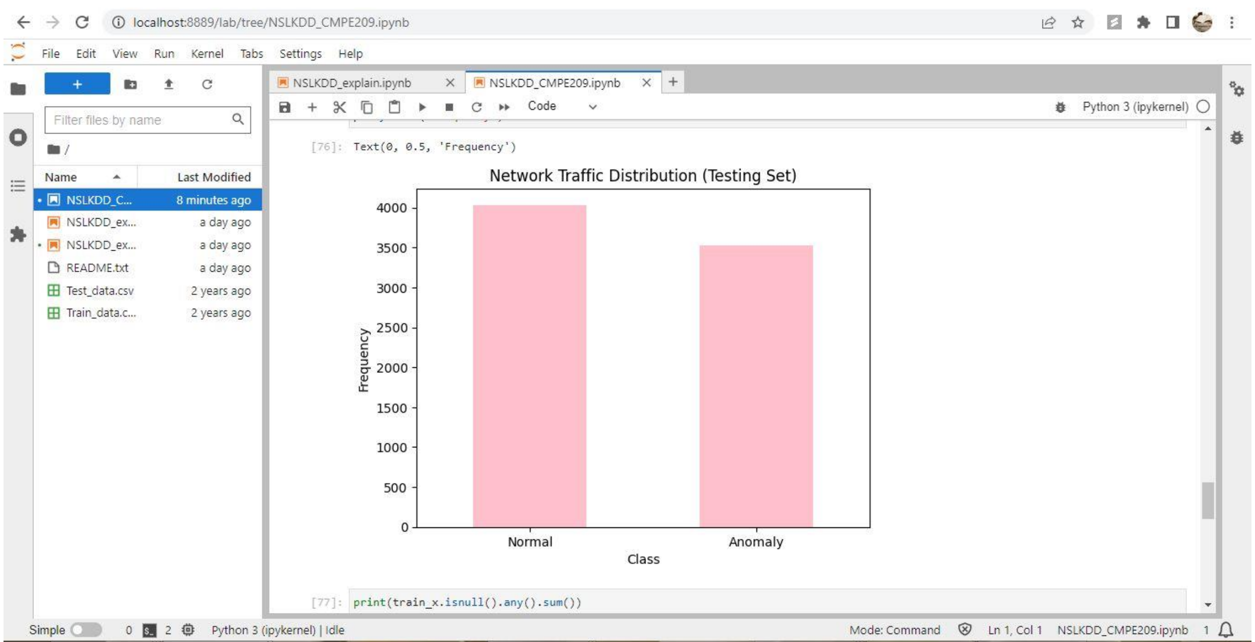

Fig. 14: Improved code 12

15. Figure 15 shows the achieved 99% output.

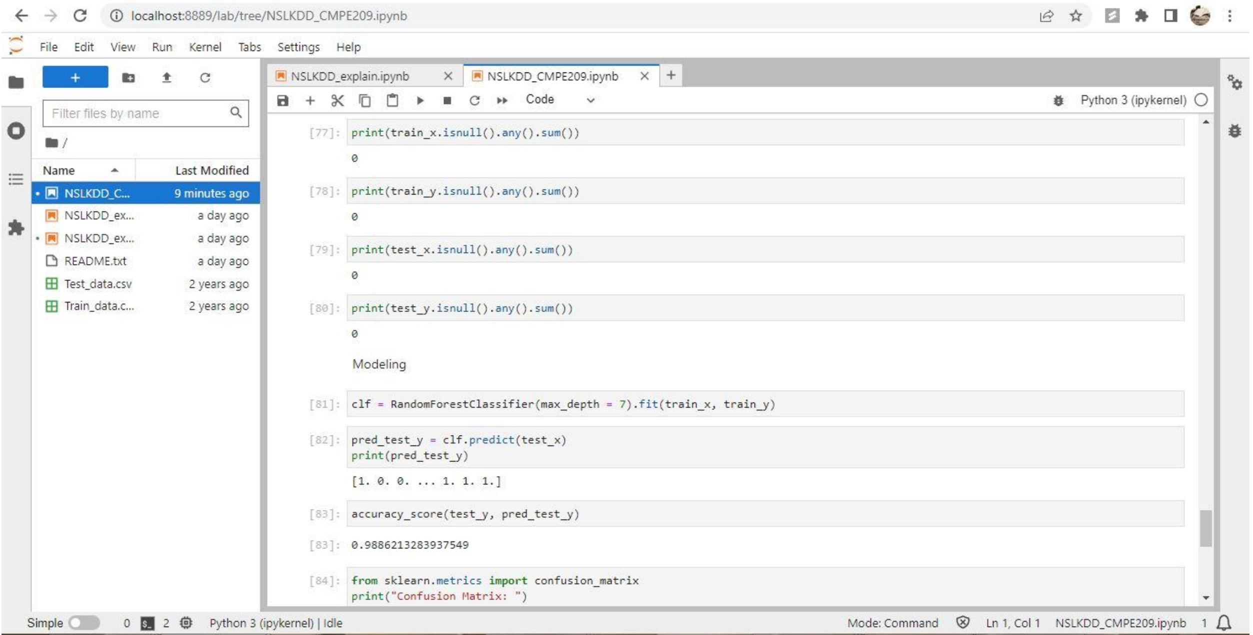

Fig. 15: Output

16. Figure 16 shows the confusion matrix.

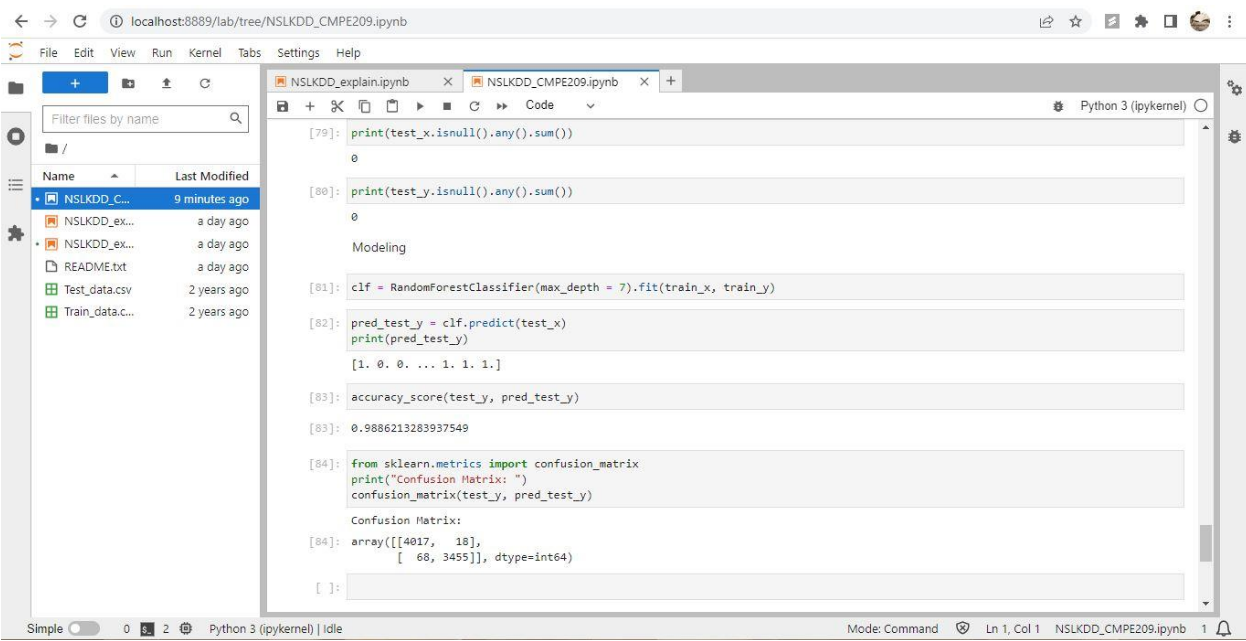

Fig. 16: Confusion matrix

CONCLUSION:

From this project, I learned about data sets and how to improve the accuracy of Random Forest by changing a few parameters. Further, I learned how Machine Learning is used for network intrusion detection.

APPENDIX

Code:
Name: NSLKDD_CMPE209.ipynb

```
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import sklearn.preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
train_data = pd.read_csv('Train_data.csv')
test_data = pd.read_csv('Test_data.csv')
```

```
train_data_df = pd.DataFrame(train_data)
print(train_data_df.shape)
```

```
test_data_df = pd.DataFrame(test_data)
print(test_data_df.shape)
```

```
set(train_data_df).difference(set(test_data_df))
```

```
train_data_df.columns
```

```
train_data_df
```

```
train_data_df = train_data_df[['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
'dst_bytes','count','srv_count','logged_in', 'class']]
```

```
train_data_df.describe()
```

```
train_data_df['class'].value_counts()
```

Feature Engineering

protocol_type

```
# convert protocol_type column to numerical values
for i in range(len(train_data_df.protocol_type)):
    if train_data_df.protocol_type[i] == 'tcp':
        train_data_df.loc[i, "protocol_type_n"] = 0
        continue
    if train_data_df.protocol_type[i] == 'udp':
        train_data_df.loc[i, "protocol_type_n"] = 1
        continue
```

```
    if train_data_df.protocol_type[i] == 'icmp':
        train_data_df.loc[i, "protocol_type_n"] = 2
        continue
```

```
train_data_df.protocol_type_n.unique()
```

service

```
print(train_data_df.service.unique())
```

```
service_d = {}
id = 0
for ft in train_data_df.service:
    if ft not in service_d:
        service_d[ft] = id
        id += 1
print(service_d)
```

```
for i, e in enumerate(train_data.service):
    train_data_df.loc[i, "service_n"] = service_d[e]
```

```
train_data_df.service_n.unique()
```

flag

```
print(train_data_df.flag.unique())
```

```
flag_d = {}
id = 0
for ft in train_data_df.flag:
    if ft not in flag_d:
        flag_d[ft] = id
        id += 1
```

```
for i, e in enumerate(train_data_df.flag):
    train_data_df.loc[i, "flag_n"] = flag_d[e]
```

```
train_data_df.flag_n.unique()
```

class

```
for i in range(len(train_data)):
    if train_data_df['class'][i] == 'normal':
        train_data_df.loc[i, 'class_n'] = 0

    if train_data_df['class'][i] == 'anomaly':
        train_data_df.loc[i, 'class_n'] = 1
```

train-to-test split

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

```
X=train_data_df[['duration', 'protocol_type_n', 'service_n', 'flag_n', 'src_bytes',
'dst_bytes','count','srv_count','logged_in']]
#Y=train_data_df[['class_n']]
```

```
train_x, test_x, train_y, test_y = train_test_split(X,train_data_df['class_n'], test_size =
0.30,stratify=train_data_df['class_n'])
```

```
train_data.shape
```

```
train_x.shape
```

```
test_x.shape
```

```
train_y.shape
```

```
train_y.value_counts()
```

```
label_l = ['Normal', 'Anomaly']
classes = pd.value_counts(train_y, sort = True)
classes.plot(kind = 'bar', rot = 0, color = "green")
plt.title("Network Traffic Distribution (Training Set)")
plt.xticks(range(2), label_l)
plt.xlabel("Class")
plt.ylabel("Frequency")
```

```
test_y.value_counts()
```

```
labels = ['Normal', 'Anomaly']
classes = pd.value_counts(test_y, sort = True)
classes.plot(kind = 'bar', rot = 0, color = "pink")
plt.title("Network Traffic Distribution (Testing Set)")
plt.xticks(range(2), labels)
plt.xlabel("Class")
plt.ylabel("Frequency")
```

```
print(train_x.isnull().any().sum())
```

```
print(train_y.isnull().any().sum())
```

```
print(test_x.isnull().any().sum())
```

```
print(test_y.isnull().any().sum())
```

Modeling

```
clf = RandomForestClassifier(max_depth = 7).fit(train_x, train_y)
```

```
pred_test_y = clf.predict(test_x)
print(pred_test_y)
```

```
accuracy_score(test_y, pred_test_y)
```

```
from sklearn.metrics import confusion_matrix
print("Confusion Matrix: ")
confusion_matrix(test_y, pred_test_y)
```