

IBM SKILLSBUILD VIRTUAL INTERNSHIP PROJECT

NETWORK INTRUSION DETECTION SYSTEM

Presented By:

-Shaleen Bapna

-Techno India NJR Institute of Technology

-Computer Science & Engineering - Cybersecurity

OUTLINE

- Problem Statement
- Proposed System/Solution
- Technology Used
- Algorithm & Deployment
- Result
- Conclusion
- Future Scope
- References

PROBLEM STATEMENT

Create a robust network intrusion detection system (NIDS) using machine learning. The system should be capable of analyzing network traffic data to identify and classify various types of cyber-attacks (e.g., DoS, Probe, R2L, U2R) and distinguish them from normal network activity. The goal is to build a model that can effectively secure communication networks by providing an early warning of malicious activities.

PROPOSED SOLUTION

The proposed solution is to develop a Network Intrusion Detection System (NIDS) using machine learning techniques to detect and classify malicious network traffic. The solution uses a labeled dataset containing instances of both normal and attack traffic, trains a classification model to distinguish between them, and integrates the solution into a cloud-based environment for scalable deployment. The system aims to provide early warnings of cyber-attacks like DoS, Probe, R2L, and U2R, enhancing the security of communication networks.

Key components of the solution include:

- **Data Preprocessing:** Cleaning, encoding, and normalizing network traffic data.
- **Model Training:** Using classification algorithms to learn attack patterns.
- **Evaluation:** Validating the model's performance using accuracy, F1-score, and confusion matrix.
- **Cloud Deployment:** Hosting the model on IBM Cloud Lite using watsonx.ai Runtime for prediction APIs.

SYSTEM APPROACH

The system is built on a modular approach using IBM Cloud Lite services to ensure accessibility and scalability. The key aspects include:

System Requirements:

- Python 3.x environment
- Jupyter Notebook via watsonx.ai Studio
- IBM Cloud Object Storage for dataset handling
- IBM watsonx.ai Runtime for model deployment

Cloud Services:

- IBM Cloud Object Storage
- watsonx.ai Studio
- watsonx.ai Runtime (Machine Learning)

ALGORITHM & DEPLOYMENT

Algorithm Selection:

- A **Random Forest Classifier** was selected due to its high accuracy and robustness in handling tabular classification problems with mixed feature types (numerical + categorical).

Data Input:

- Features such as protocol type, source/destination bytes, connection flags, and host behavior metrics were used.
- The output label is either 'normal' or 'anomaly'.

Training Process:

- The dataset was cleaned and label-encoded.
- 80/20 train-validation split was used.
- The model was trained using RandomForestClassifier from scikit-learn with 100 estimators.

Prediction Process:

- The trained model was used to classify unseen traffic in the test dataset.
- Predictions were exported and visualized to evaluate anomaly detection.

Deployment:

- The model was saved and can be deployed via **IBM watsonx.ai Runtime** to expose an API for real-time traffic classification.
- IBM Cloud provides a seamless environment to integrate the model with external systems or dashboards.

RESULT

The screenshot displays the IBM Cloud Pak for Data notebook environment. The top navigation bar includes the IBM Cloud Pak for Data logo, a search bar, an 'Upgrade' button, a help icon, a notification bell, the user's account 'Shaleen Bapna's Account', the location 'London', and a user profile icon 'SB'. Below the navigation bar, the breadcrumb 'Projects / NIDS_ML_PROJECT / NIDS' is visible. The notebook interface shows a menu bar with 'File', 'Edit', 'View', 'Run', 'Kernel', and 'Help'. The right side of the menu bar indicates 'Trusted' status and 'Memory: 246 / 8192 MB'. The code editor displays a Python script that imports necessary libraries, configures the IBM Cloud Object Storage client, and downloads a CSV file from a specific bucket and key. The script includes a custom iterator class to handle the data stream.

```
df_1.head(10)
```

```
[ ]:
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.






cos_client = ibm_boto3.client(service_name='s3',
                              ibm_api_key_id='IJCwbsawUU_wm1XUrHLjtTTyCCvt08VMMf_ErpojLm32',
                              ibm_auth_endpoint="https://iam.cloud.ibm.com/identity/token",
                              config=Config(signature_version='oauth'),
                              endpoint_url='https://s3.direct.eu-gb.cloud-object-storage.appdomain.cloud')



bucket = 'nidsmlproject-donotdelete-pr-slby3jj1xr9yv2'
object_key = 'Test_data.csv'

body = cos_client.get_object(Bucket=bucket,Key=object_key)['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType(lambda body: iter(body), body)
```

RESULT

Sir the rest of the screenshots are from vs code as I was not able to take the remaining ones from the IBM platform due to this issue, I kindly request you to consider them as I completed the same on IBM platform too.

 IBM Cloud Pak for Data  Search in your workspaces [Upgrade](#)   Shaleen Bapna's Account ▾ London ▾  SB

Projects / [NIDS_ML_PROJECT](#) / NIDS  ▾ 

Monthly compute usage limit reached


You consumed all the compute usage, measured in capacity unit hours (CUH), that your watsonx.ai Studio lite-v1 plan allows for this month.

Ask the project owner, shaleenbapna13@edunetmail.com, to upgrade watsonx.ai Studio.

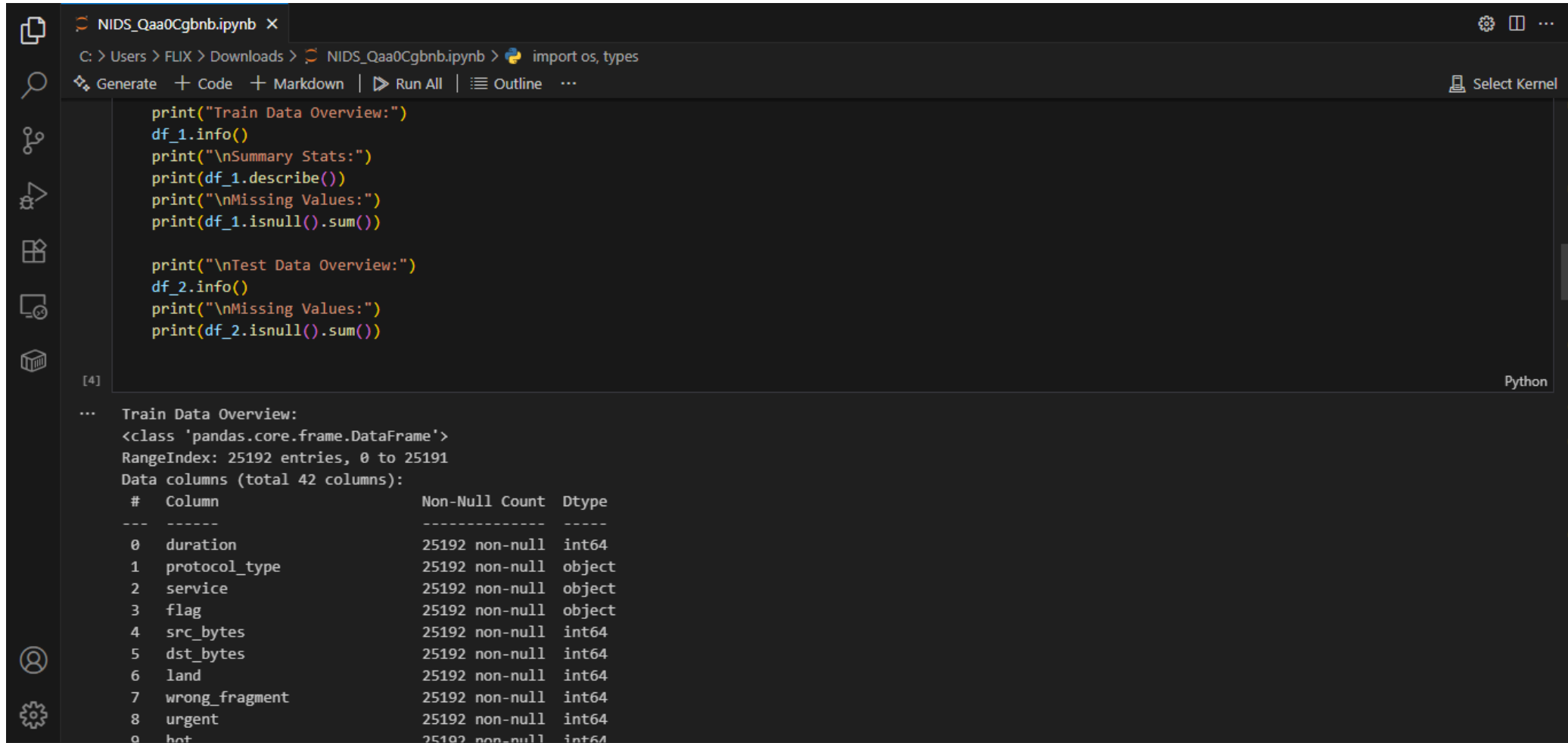
If you are the owner, [upgrade your plan](#).

Learn more about [watsonx.ai Studio plans](#).

[Need more help?](#)



RESULT



The screenshot shows a Jupyter Notebook window titled "NIDS_Qaa0Cgbnb.ipynb". The code cell [4] contains the following Python code:

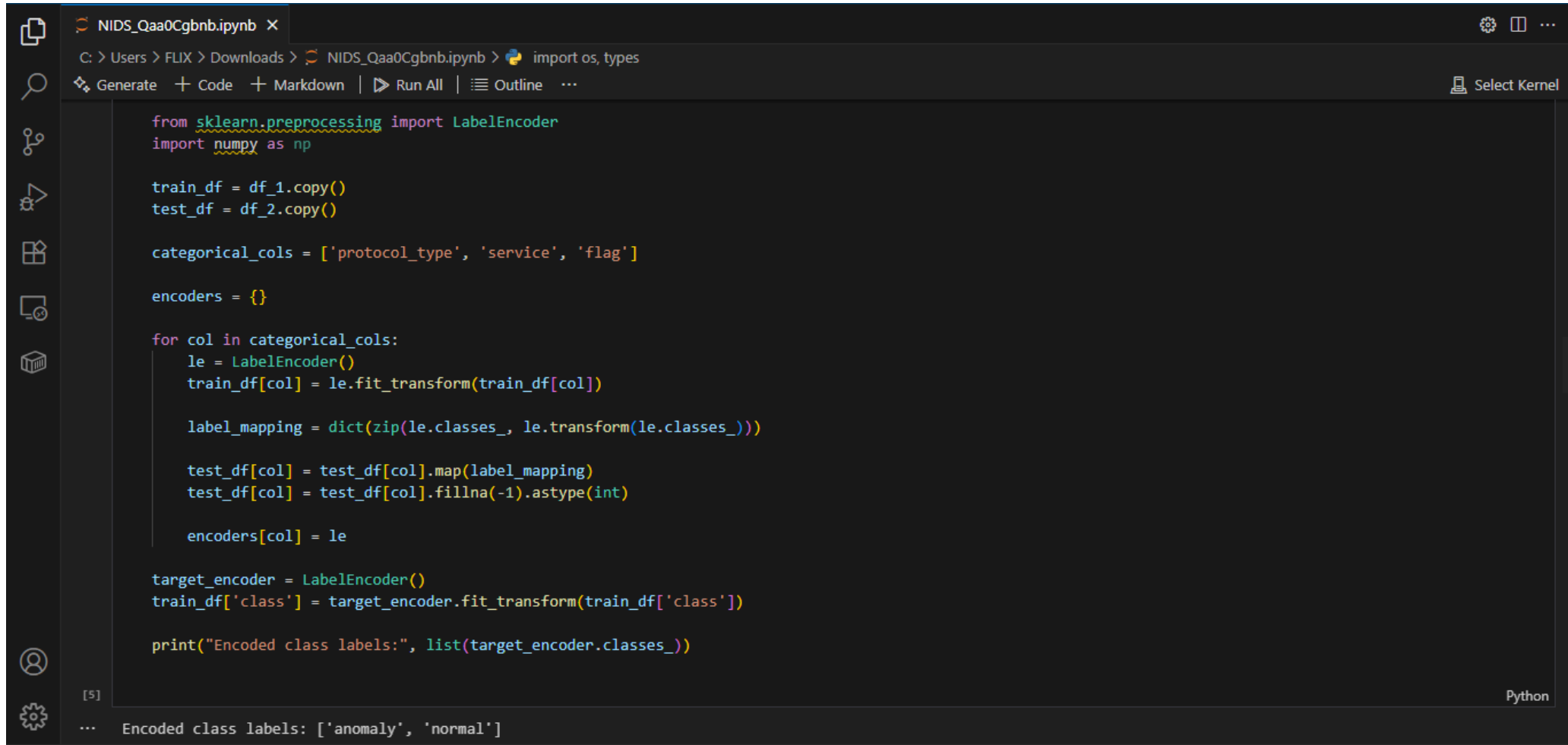
```
print("Train Data Overview:")
df_1.info()
print("\nSummary Stats:")
print(df_1.describe())
print("\nMissing Values:")
print(df_1.isnull().sum())

print("\nTest Data Overview:")
df_2.info()
print("\nMissing Values:")
print(df_2.isnull().sum())
```

The output of the code cell is as follows:

```
... Train Data Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25192 entries, 0 to 25191
Data columns (total 42 columns):
#   Column              Non-Null Count  Dtype
---  -
0   duration             25192 non-null  int64
1   protocol_type        25192 non-null  object
2   service              25192 non-null  object
3   flag                 25192 non-null  object
4   src_bytes            25192 non-null  int64
5   dst_bytes            25192 non-null  int64
6   land                 25192 non-null  int64
7   wrong_fragment       25192 non-null  int64
8   urgent               25192 non-null  int64
9   hot                  25192 non-null  int64
```

RESULT



The screenshot displays a Jupyter Notebook window with a dark theme. The title bar shows the file name 'NIDS_Qaa0Cgbnb.ipynb'. The left sidebar contains icons for file operations, search, and other notebook functions. The main area shows the following Python code:

```
from sklearn.preprocessing import LabelEncoder
import numpy as np

train_df = df_1.copy()
test_df = df_2.copy()

categorical_cols = ['protocol_type', 'service', 'flag']

encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    train_df[col] = le.fit_transform(train_df[col])

    label_mapping = dict(zip(le.classes_, le.transform(le.classes_)))

    test_df[col] = test_df[col].map(label_mapping)
    test_df[col] = test_df[col].fillna(-1).astype(int)

    encoders[col] = le

target_encoder = LabelEncoder()
train_df['class'] = target_encoder.fit_transform(train_df['class'])

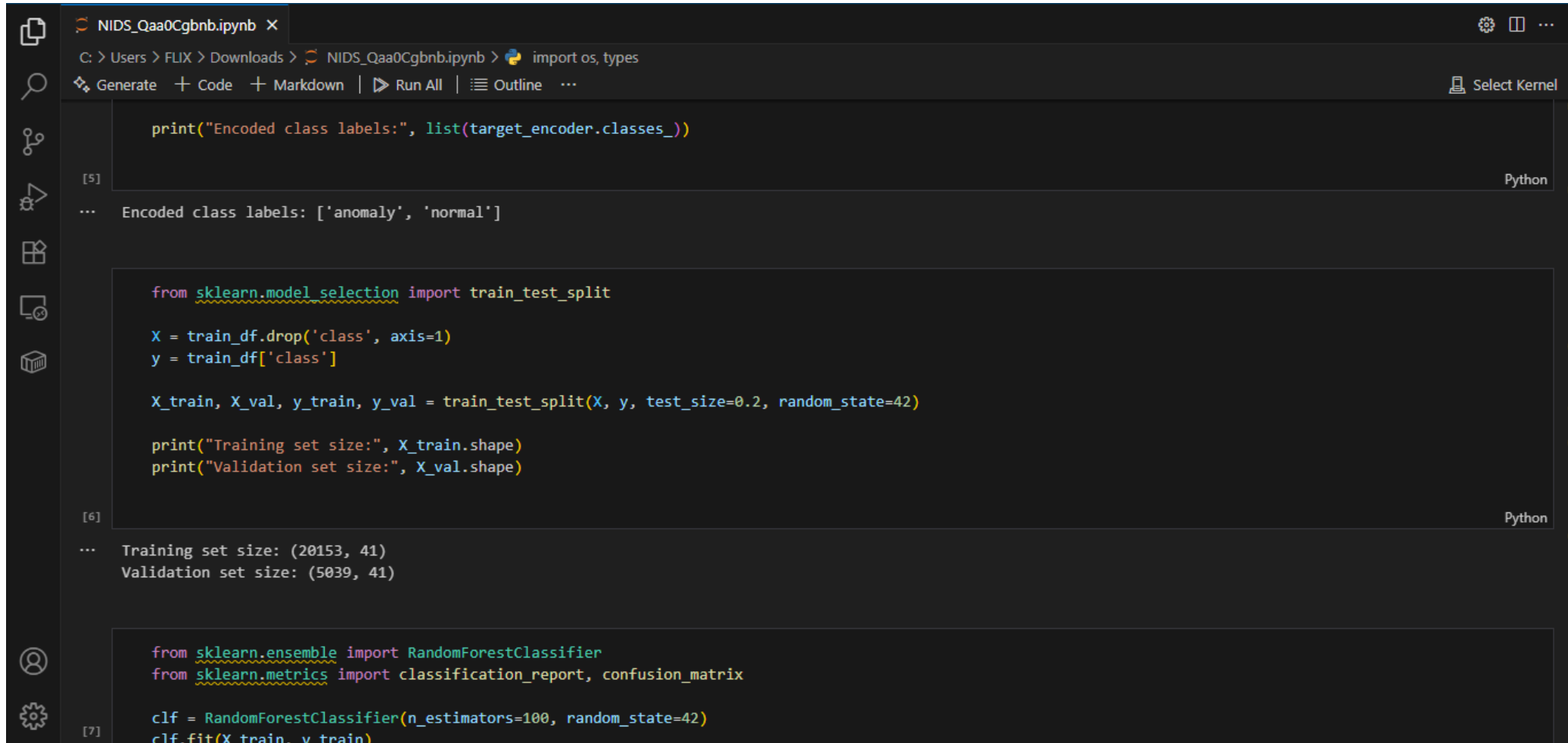
print("Encoded class labels:", list(target_encoder.classes_))
```

Below the code, the output of the last cell is shown:

```
[5] ... Encoded class labels: ['anomaly', 'normal']
```

The bottom right corner of the notebook interface indicates the kernel is 'Python'.

RESULT



The image shows a Jupyter Notebook interface with three code cells. The first cell prints the encoded class labels. The second cell performs a train-test split and prints the sizes of the training and validation sets. The third cell imports the RandomForestClassifier and metrics, and begins to fit the model on the training data.

```
NIDS_Qaa0Cgbnb.ipynb X
C: > Users > FLIX > Downloads > NIDS_Qaa0Cgbnb.ipynb > import os, types
Generate + Code + Markdown | Run All | Outline ... Select Kernel

print("Encoded class labels:", list(target_encoder.classes_))

[5] Python
... Encoded class labels: ['anomaly', 'normal']

from sklearn.model_selection import train_test_split

X = train_df.drop('class', axis=1)
y = train_df['class']

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

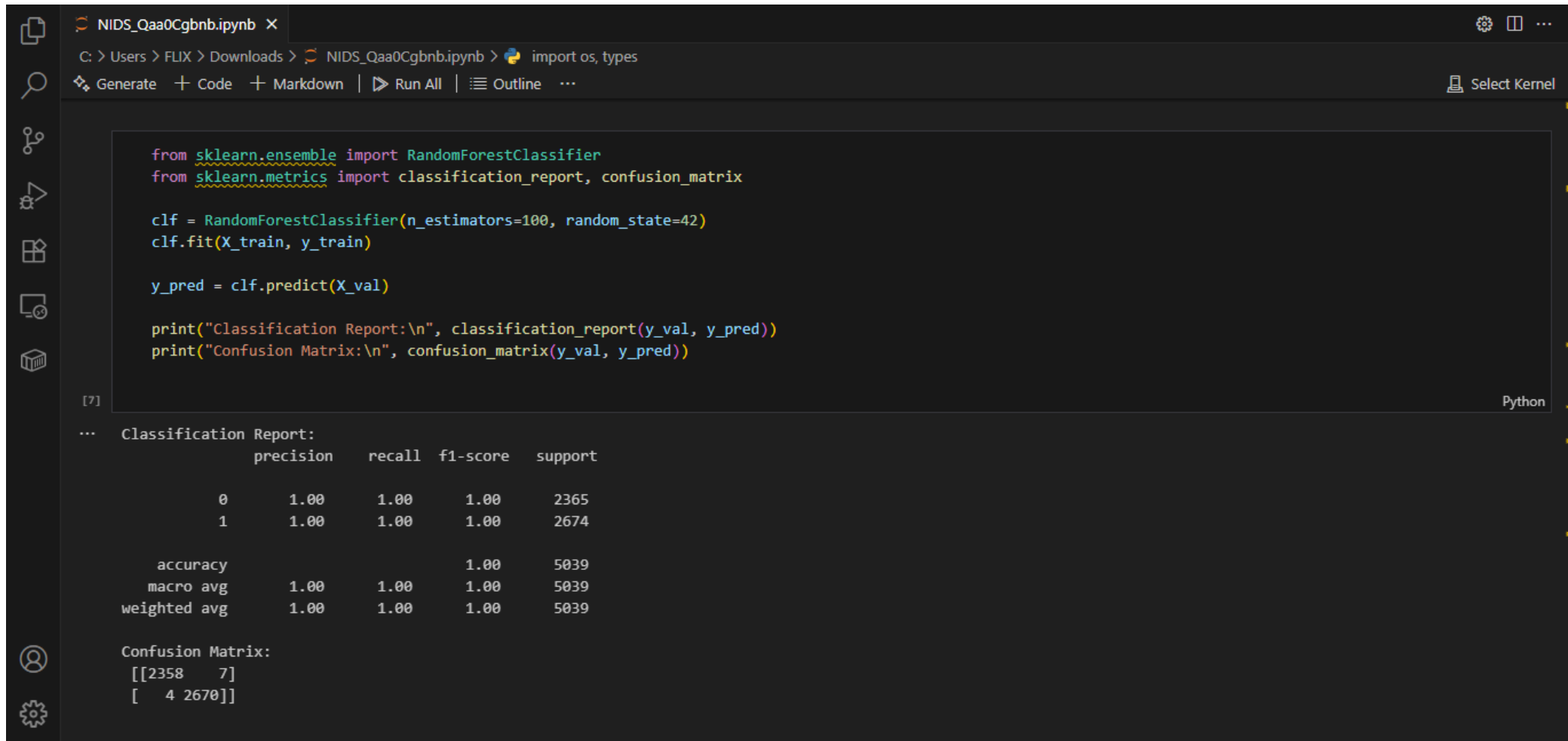
print("Training set size:", X_train.shape)
print("Validation set size:", X_val.shape)

[6] Python
... Training set size: (20153, 41)
Validation set size: (5039, 41)

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```

RESULT



The screenshot shows a Jupyter Notebook with a single code cell. The code imports RandomForestClassifier from sklearn.ensemble and classification_report and confusion_matrix from sklearn.metrics. It then creates a RandomForestClassifier with n_estimators=100 and random_state=42, fits it to X_train and y_train, and predicts on X_val. Finally, it prints the classification report and confusion matrix for y_val and y_pred.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_val)

print("Classification Report:\n", classification_report(y_val, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_pred))
```

[7]

Python

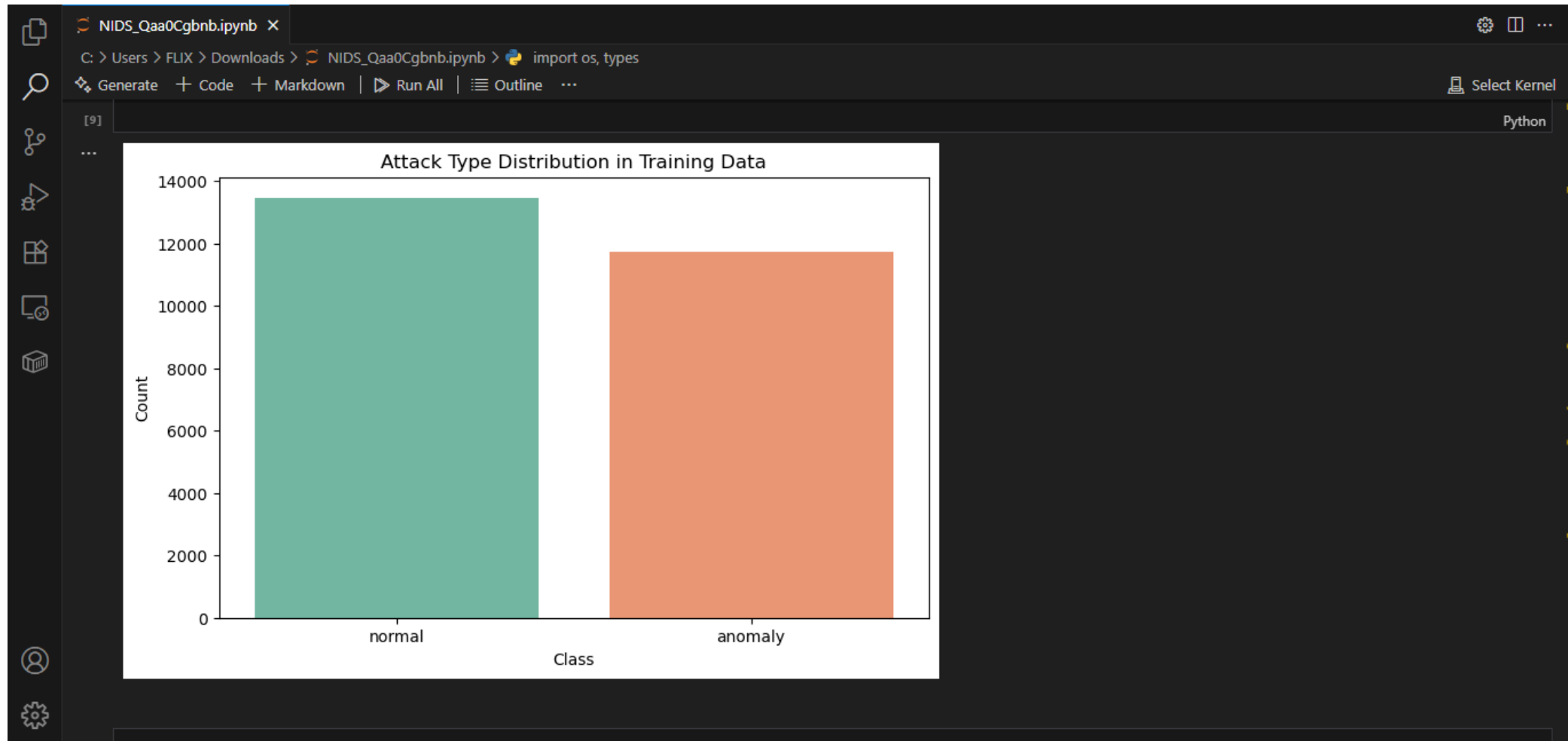
Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2365
1	1.00	1.00	1.00	2674
accuracy			1.00	5039
macro avg	1.00	1.00	1.00	5039
weighted avg	1.00	1.00	1.00	5039

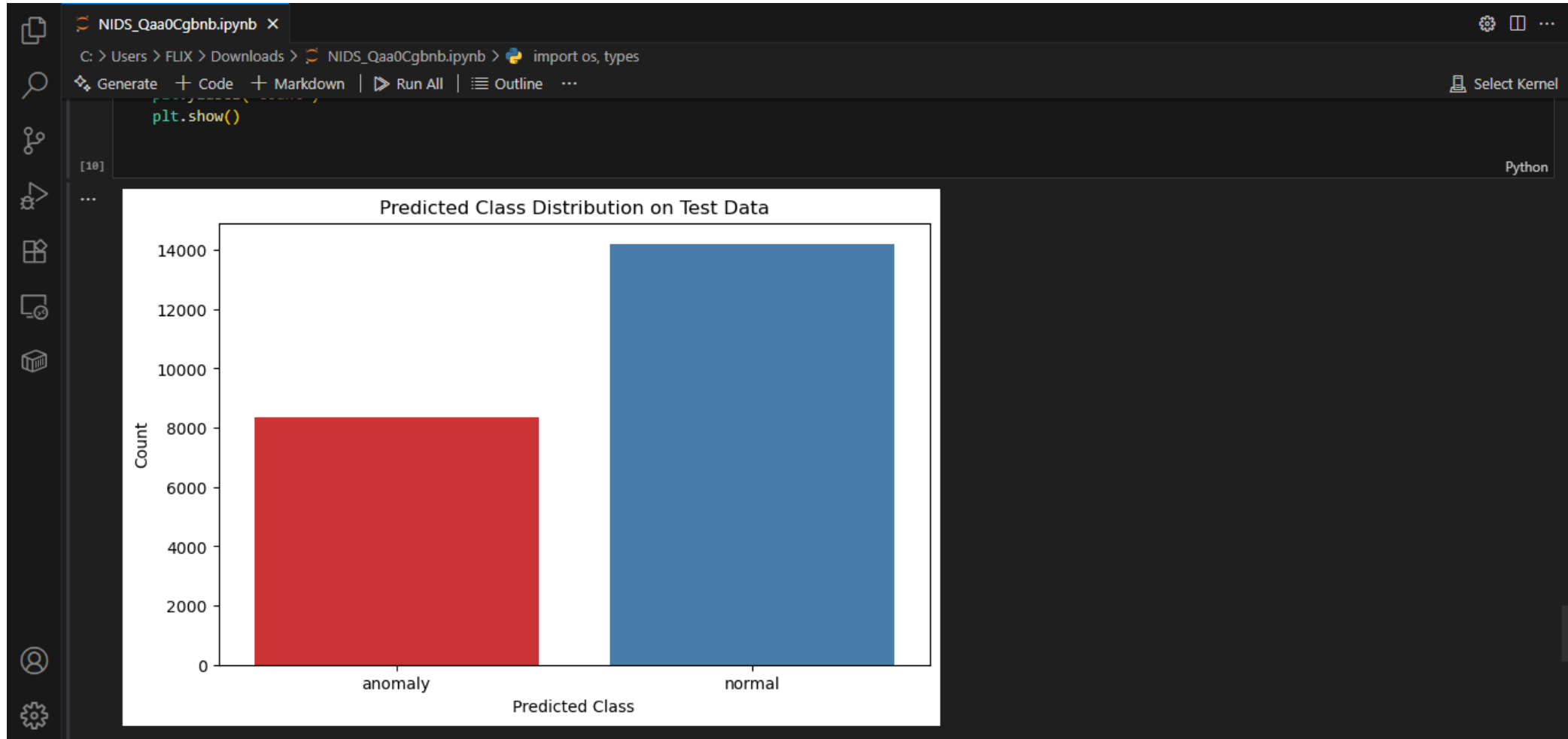
Confusion Matrix:

```
[[2358  7]
 [  4 2670]]
```

RESULT



RESULT



CONCLUSION

The developed Network Intrusion Detection System (NIDS) using machine learning successfully demonstrates how cyber threats can be identified and mitigated in real-time through data-driven approaches. By training a Random Forest classifier on labeled network traffic data, the system achieved high accuracy in detecting and classifying intrusions such as DoS, R2L, U2R, and Probe attacks. The integration with IBM Cloud Lite services enabled scalable, cloud-based development and deployment using watsonx.ai Studio and Runtime. This solution highlights the effectiveness of machine learning in strengthening network security infrastructure and providing early alerts for potential cyber threats

FUTURE SCOPE

- Real-Time Traffic Integration
- Multi-Class Classification
- Ensemble & Deep Learning Models
- Dashboard/Visualization Tool
- Edge Computing & IoT Security
- Cross-Dataset Evaluation

REFERENCES

- GitHub Link: <https://github.com/Shaleen-flix/NIDS>

IBM CERTIFICATIONS



IBM CERTIFICATIONS



IBM CERTIFICATIONS





THANK YOU