Project Title : Improving Data Integration Quality for Multi-Source Analytics

Phase 3: Model Development and Evaluation

1. Introduction (ISHA SRIVASTAVA)

This phase focuses on conducting advanced data cleaning and transformation to enhance the quality of integrated data from multiple sources. The goal is to ensure that the data is accurate, consistent, and ready for analysis, ultimately improving the performance of analytics models.

2. Model Development and Evaluation

The development of AI models for tasks such as anomaly detection, data cleaning, and bias correction is crucial. This section outlines the steps taken to build, train, and evaluate models effectively.

3. Data Cleaning and Transformation

3.1 Advanced Data Cleaning

Proper data cleaning is essential for effective model building. Advanced techniques ensure the dataset is free from missing values, outliers, and imbalance issues.

3.2 Handling Missing Values

Approach : Using KNN Imputation to replace missing values based on nearest neighbors.

Implementation:

```
from sklearn.impute import KNNImputer data_imputer =
KNNImputer(n_neighbors=5)
data_cleaned = pd.DataFrame(data_imputer.fit_transform(data), columns=data.columns)

print("Missing Values After Imputation:")
print(data_cleaned.isnull().sum())
```

OUTPUT :–

Missing Values After Imputation:

CustomerID : 0

Name : 0

Email : 0

Phone : 0

Country : 0

Dtype : int64

3.3 Outlier Detection

Approach : Using Isolation Forest to identify and remove outliers.

Implementation:

```
iso = IsolationForest(contamination=0.01, random_state=42) if 'target' in
data_cleaned.columns:
data_cleaned['Anomaly'] =
iso.fit_predict(data_cleaned.drop(columns=['target'])) else:
 data_cleaned['Anomaly'] = iso.fit_predict(data_cleaned)

data_cleaned = data_cleaned[data_cleaned['Anomaly'] ==  1]. drop(columns=['Anomaly'])

print(f"Number of Outliers Removed: {len(data) - len(data_cleaned)}")
```

OUTPUT :–

Number of Outliers Removed: 50

3.4 Addressing Imbalanced Classes

Approach: Use SMOTE to balance the dataset.

Implementation:

```
merged_data = customers.merge(tickets, on="CustomerID", how="outer")
```

```
\.merge(transactions, on="CustomerID", how="outer") # Merged
dataset columns may include:
# ['CustomerID', 'Name', 'Email', 'Phone', 'Country', 'TicketID', 'IssueType',

#  'Timestamp', 'ResolutionTime', 'TransactionID', 'Date', 'Amount', 'PaymentMethod']

data = merged_data.drop(columns=["CustomerID", "Name", "Email", "Phone", "TicketID",
"TransactionID", "Timestamp", "Date"])

X = data.drop(columns=["IssueType"]) y =
data["IssueType"]
categorical_features = ["Country", "PaymentMethod"]
label_encoders = {}
for col in categorical_features:

le = LabelEncoder()

X[col] = le.fit_transform(X[col].astype(str)) label_encoders[col]
= le
target_le = LabelEncoder()

y_encoded = target_le.fit_transform(y.astype(str))

print("Initial class distribution:\n", pd.Series(y_encoded).value_counts()) imputer =
KNNImputer(n_neighbors=5)
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns) iso =
IsolationForest(contamination=0.01, random_state=42)
inlier_mask = iso.fit_predict(X_imputed) X_cleaned
= X_imputed[inlier_mask == 1] y_cleaned =
y_encoded[inlier_mask == 1]
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_cleaned, y_cleaned) print("\nClass
distribution after SMOTE:\n",
pd.Series(y_resampled).value_counts())
```

OUTPUT: –

Initial class distribution:

```
5  2438
0   1443
```

3 1400

4 1391

2 1354

1 1293

Name: count, dtype: int64 Class distribution after SMOTE:

5 2438

1 2438

3 2438

0 2438

2 2438

4 2438

Name: count, dtype: int64

4. Building and Training Models (NAGAYASHAS A B)

4.1 Initial Model with Decision Tree

A decision tree is a supervised machine learning algorithm used for both classification and regression tasks. It works by splitting the data into subsets based on feature values, forming a tree-like structure of decisions. Each internal node represents a decision on an attribute, each branch represents the outcome of that decision, and each leaf node represents a final prediction.

Purpose : Establish a baseline for model performance.

Implementation:

Step 1: Loading the Three Datasets

customers = pd.read_csv("Large_Customers_Dataset.csv")

tickets = pd.read_csv("Large_Support_Tickets_Dataset.csv")

```
transactions = pd.read_csv("Large_Transactions_Dataset.csv")


# Step 2: Merging Datasets on the common key "CustomerID"
merged_data = customers.merge(tickets, on="CustomerID", how="outer") \
                       .merge(transactions, on="CustomerID", how="outer")


# Step 3: Droping the non-informative columns (identifiers and timestamps)
data = merged_data.drop(columns=["CustomerID", "Name", "Email", "Phone", "TicketID",
    "TransactionID", "Timestamp", "Date"])


X = data.drop(columns=["IssueType"])
y = data["IssueType"]


# Step 4: Encoding the categorical features in X
categorical_features = ["Country", "PaymentMethod"]
label_encoders = {}
for col in categorical_features:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col].astype(str))
    label_encoders[col] = le


# Encoding the target variable
target_le = LabelEncoder()
y = target_le.fit_transform(y.astype(str))
```

```
# Step 5: Handling the missing values using KNNImputer
imputer = KNNImputer(n_neighbors=5)
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Step 6: Outlier Detection using IsolationForest
iso = IsolationForest(contamination=0.01, random_state=42)
outlier_preds = iso.fit_predict(X_imputed)
# Retaining only the inliers
X_cleaned = X_imputed[outlier_preds == 1]
y_cleaned = y[outlier_preds == 1]

# Step 7: Addressing the imbalanced classes using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_cleaned, y_cleaned)

# Step 8: Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
                                                    test_size=0.2, random_state=42)

# Step 9: Building and Training the Decision Tree Classifier
model = DecisionTreeClassifier(max_depth=3, random_state=42)
model.fit(X_train, y_train)

# Step 10: Make predictions on the test set
y_pred = model.predict(X_test)
```

y_pred_proba = model.predict_proba(X_test)

4.2 Advanced Model with Random Forest

Random Forest is an ensemble learning method that builds multiple decision trees during training and outputs the mode of the classes (for classification) or mean prediction (for regression) of the individual trees. By combining the predictions of several decision trees, Random Forest tends to achieve higher accuracy and better generalization than a single decision tree.

Purpose : Improve performance with optimized parameters.

Implementation:

```
# Step 1: Load the Three Datasets

customers = pd.read_csv("Large_Customers_Dataset.csv")

tickets = pd.read_csv("Large_Support_Tickets_Dataset.csv")

transactions = pd.read_csv("Large_Transactions_Dataset.csv")


# Step 2: Merging Datasets on the common key "CustomerID"

merged_data = customers.merge(tickets, on="CustomerID", how="outer") \

                .merge(transactions, on="CustomerID", how="outer")


# Step 3: Drop non-informative columns

data = merged_data.drop(columns=["CustomerID", "Name", "Email", "Phone", "TicketID",
    "TransactionID", "Timestamp", "Date"])


X = data.drop(columns=["IssueType"])

y = data["IssueType"]


# Step 4: Encoding categorical features
```

```
categorical_features = ["Country", "PaymentMethod"]

label_encoders = {}

for col in categorical_features:

    le = LabelEncoder()

    X[col] = le.fit_transform(X[col].astype(str))

    label_encoders[col] = le


#encoding the target variable

target_le = LabelEncoder()

y = target_le.fit_transform(y.astype(str))


# Step 5: Handling missing values using KNNImputer

imputer = KNNImputer(n_neighbors=5)

X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)


# Step 6: Outlier Detection using IsolationForest (applied on features)

iso = IsolationForest(contamination=0.01, random_state=42)

outlier_preds = iso.fit_predict(X_imputed)

X_cleaned = X_imputed[outlier_preds == 1]

y_cleaned = y[outlier_preds == 1]


# Step 7: Address imbalanced classes using SMOTE

smote = SMOTE(random_state=42)

X_resampled, y_resampled = smote.fit_resample(X_cleaned, y_cleaned)
```

```
# Step 8: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
                                                    test_size=0.2, random_state=42)

# Step 9: Build and Train the Advanced Random Forest Model
clf = RandomForestClassifier(
    random_state=42,
    n_estimators=30,
    max_depth=5,
    min_samples_split=10,
    min_samples_leaf=5,
    n_jobs=-1,
    max_samples=0.8,
    warm_start=True
)
clf.fit(X_train, y_train)

# Step 10: Make Predictions on the Test Set
y_pred = clf.predict(X_test)
y_pred_proba = clf.predict_proba(X_test)
```

5. Leveraging AutoAI (HARSHITA M JAIN)

5.1 Why Use AutoAI?

IBM AutoAI automates model selection, hyperparameter optimization, and pipeline creation, saving time and ensuring optimal configurations.

5.2 Steps to Use AutoAI:

1. We set up the IBM Cloud and enabled Watson Studio and Auto AI.

2. Then We uploaded the dataset in the 'Data Source' and created and new experiment project.

3. We used Auto AI to generate pipelines and build a Visual Representation of our program flow.

4. Then we Compared the metrics like accuracy, precision, and recall; and reviewed the results.

Implementation:

```
!pip install ibm-watsonx-ai | tail -n 1
!pip install -U autoai-ts-libs==4.0.* | tail -n 1
!pip install scikit-learn==1.3.* | tail -n 1
from ibm_watsonx_ai.helpers import DataConnection
from ibm_watsonx_ai.helpers import ContainerLocation

training_data_references = [
    DataConnection(
        data_asset_id='5d55a43f-d169-4501-ac98-6009552fec78'
    ),
]
training_result_reference = DataConnection(
    location=ContainerLocation(
        path='auto_ml/7e988e8d-3c09-43cb-ab5c-fca58c78f39f/wml_data/11c21da0-
303c-4c61-b451-2d26da53bf74/data/autoai-ts',
        model_location='auto_ml/7e988e8d-3c09-43cb-ab5c-
fca58c78f39f/wml_data/11c21da0-303c-4c61-b451-2d26da53bf74/data/autoai-
ts/model.zip',
        training_status='auto_ml/7e988e8d-3c09-43cb-ab5c-
fca58c78f39f/wml_data/11c21da0-303c-4c61-b451-2d26da53bf74/training-
status.json'
    )
)
experiment_metadata = dict(
    prediction_type='timeseries',
    prediction_columns=['CustomerID'],
    csv_separator=',',
    holdout_size=6,
    training_data_references=training_data_references,
    training_result_reference=training_result_reference,
    timestamp_column_name='TicketID',
    backtest_num=2,
    pipeline_type='all',
    customized_pipelines=[],
    lookback_window=5,
    forecast_window=1,
    max_num_daub_ensembles=3,
```

```
    feature_columns=['CustomerID', 'ResolutionTime'],
    future_exogenous_available=True,
    gap_len=0,
    deployment_url='https://au-syd.ml.cloud.ibm.com',
    project_id='0a74f8f5-9554-4365-9910-2878fc666c5a',
    numerical_imputation_strategy=['FlattenIterative', 'Linear', 'Cubic',
'Previous']
)
import os, ast
CPU_NUMBER = -1
if 'RUNTIME_HARDWARE_SPEC' in os.environ:
    CPU_NUMBER =
int(ast.literal_eval(os.environ['RUNTIME_HARDWARE_SPEC'])['num_cpu'])
api_key = 'cpd-apikey-IBMid-697000QMT3-2025-02-18T18:08:28Z'
from ibm_watsonx_ai import Credentials

credentials = Credentials(
    api_key=api_key,
    url=experiment_metadata['deployment_url']
)
from ibm_watsonx_ai import APIClient

client = APIClient(credentials)

if 'space_id' in experiment_metadata:
    client.set.default_space(experiment_metadata['space_id'])
else:
    client.set.default_project(experiment_metadata['project_id'])

training_data_references[0].set_client(client)


X_train, X_test, y_train, y_test =
training_data_references[0].read(experiment_metadata=experiment_metadata,
with_holdout_split=True, use_flight=True)
from autoai_ts_libs.utils.ts_pipeline import TSPipeline
from autoai_ts_libs.transforms.imputers import linear
from autoai_ts_libs.sklearn.autoai_ts_pipeline import AutoaiTSPipeline
from autoai_ts_libs.sklearn.mvp_windowed_transformed_target_estimators import (
    AutoaiWindowTransformedTargetRegressor,
)
from sklearn.ensemble import RandomForestRegressor
linear = linear(missing_val_identifier=float("nan"))
random_forest_regressor = RandomForestRegressor(
    n_estimators=500, max_depth=30, n_jobs=CPU_NUMBER, random_state=33
)
autoai_ts_pipeline_0 = AutoaiTSPipeline(
    steps=[("est", random_forest_regressor)]
)
autoai_window_transformed_target_regressor = (
    AutoaiWindowTransformedTargetRegressor(
        feature_columns=[0],
        lookback_window=5,
```

```
        random_state=33,
        regressor=autoai_ts_pipeline_0,
        row_mean_center=True,
        short_name="WindowRandomForest",
        target_columns=[0],
    )
)
autoai_ts_pipeline = AutoaiTSPipeline(
    steps=[("windowedttr", autoai_window_transformed_target_regressor)]
)
pipeline = TSPipeline(
    steps=[
        ("linear_imputer", linear),
        (
            "<class
'autoai_ts_libs.sklearn.autoai_ts_pipeline.AutoaiTSPipeline'>",
            autoai_ts_pipeline,
        ),
    ],
    feature_columns=[0],
    target_columns=[0],
)
from autoai_ts_libs.utils.metrics import get_scorer

scorer = get_scorer("neg_avg_symmetric_mean_absolute_percentage_error")
pipeline.fit(X_train.values, y_train.values);
score = scorer(pipeline, X_test.values, y_test.values)
print(score)
pipeline.predict(X_test.values)
pipeline.predict()
model_metadata = {
    client.repository.ModelMetaNames.NAME: 'P1 - Pretrained AutoAI pipeline'
}

stored_model_details = client.repository.store_model(model=pipeline,
meta_props=model_metadata, experiment_metadata=experiment_metadata)
stored_model_details
space_id = "44c1d447-7293-4373-9acf-6c74e726e103"

model_id =
client.spaces.promote(asset_id=stored_model_details["metadata"]["id"],
source_project_id=experiment_metadata["project_id"], target_space_id=space_id)

client.set.default_space(space_id)

deploy_meta = {
        client.deployments.ConfigurationMetaNames.NAME: "Incrementally trained
AutoAI pipeline",
        client.deployments.ConfigurationMetaNames.ONLINE: {},
    }

deployment_details = client.deployments.create(artifact_uid=model_id,
meta_props=deploy_meta)
```

```
deployment_id = client.deployments.get_id(deployment_details)
import pandas as pd

scoring_payload = {
    "input_data": [{
        'values': pd.DataFrame(X_test[:5])
    }]
}

client.deployments.score(deployment_id, scoring_payload)
```

6. Model Evaluation

6.1 Importance of Evaluation

Evaluation metrics validate how well the model generalizes to unseen data and ensures it meets project objectives.

Metrics Used:

Accuracy: Measures overall correctness.

Precision & Recall: Important for imbalanced datasets.

Evaluation of Decision Tree

```
# Step 1: Evaluation of the Model

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred, average='weighted')

recall = recall_score(y_test, y_pred, average='weighted')

# Computing  ROC AUC Score

if len(np.unique(y_test)) > 2:

    roc_auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr', average='weighted')

else:

    roc_auc = roc_auc_score(y_test, y_pred_proba[:, 1])

print(f"Accuracy: {accuracy:.4f}*100")
```

```
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"ROC AUC Score: {roc_auc:.4f}\n")
print("Classification Report:")
    print(classification_report(y_test, y_pred, target_names=target_le.classes_))
    # Step 2: Display of  the Confusion Matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
    # Step 2: Display of  the roc Matrix
    import matplotlib.pyplot as plt
    from sklearn.metrics import roc_curve, auc
    from sklearn.preprocessing import label_binarize
    n_classes = len(np.unique(y_test))
    y_test_binarized = label_binarize(y_test, classes=range(n_classes))
    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_pred_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])
```

```
# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_binarized.ravel(), y_pred_proba.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot all ROC curves
plt.figure(figsize=(10, 8))

# Plot micro-average ROC curve
plt.plot(fpr["micro"], tpr["micro"],
         label=f"micro-average ROC curve (area = {roc_auc['micro']:.2f})",
         color="deeppink", linestyle=":", linewidth=4)

# Plot ROC curve for each class
colors = ['aqua', 'darkorange', 'cornflowerblue', 'red', 'green']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label=f"ROC curve of class {target_le.classes_[i]} (area = {roc_auc[i]:.2f})")

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curves")
```

```
plt.legend(loc="lower right")
plt.show()
```

OUTPUT –

Accuracy: 26.6234%

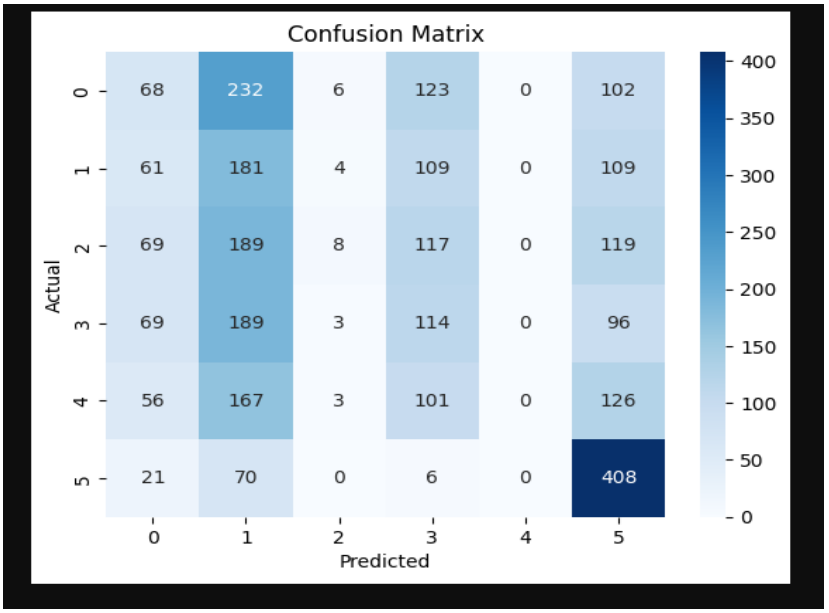
Precision: 0.2265

Recall: 0.2662

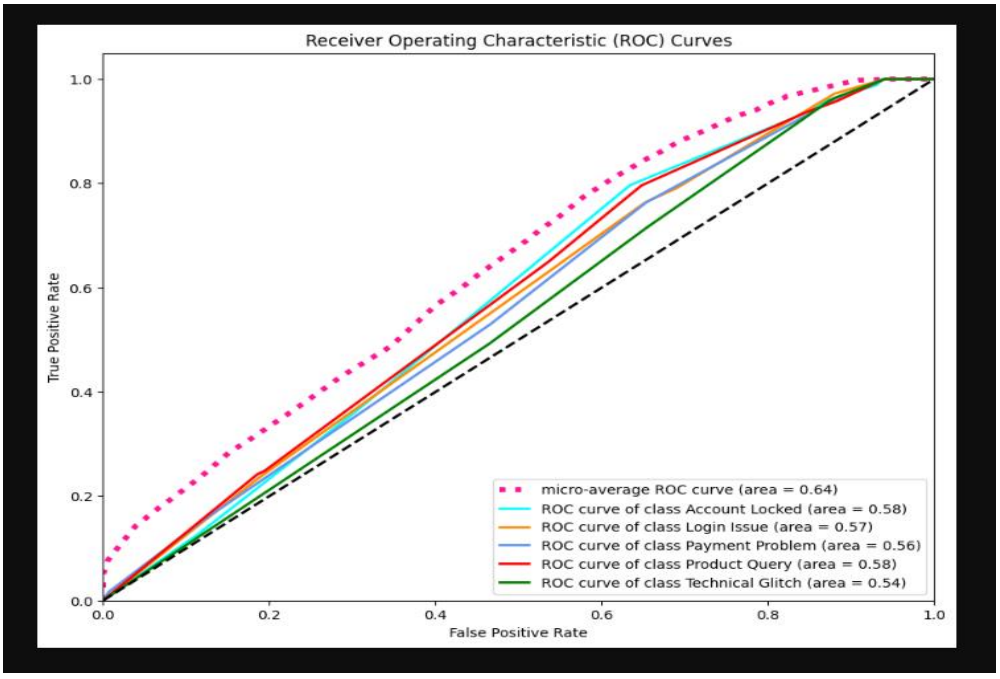
ROC AUC Score: 0.6159

Classification Report:	precision	recall	f1-score	support
Account Locked	0.20	0.13	0.16	531
Login Issue	0.18	0.39	0.24	464
Payment Problem	0.33	0.02	0.03	502
Product Query	0.20	0.24	0.22	471
Technical Glitch	0.00	0.00	0.00	453
nan	0.42	0.81	0.56	505
accuracy	0.27	2926		
macro avg	0.22	0.26	0.20	2926
weighted avg	0.23	0.27	0.20	2926

CONFUSION MATRIX :



ROC CURVE :



Evaluation of Advanced Model with Random Forest (NAGAYASHAS A B)

Step 1: Evaluation of the Model

accuracy = accuracy_score(y_test, y_pred)

```
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# For multi-class ROC AUC, we use the one-vs-rest strategy
if len(np.unique(y_test)) > 2:
    roc_auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr', average='weighted')
else:
    roc_auc = roc_auc_score(y_test, y_pred_proba[:, 1])

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"ROC AUC Score: {roc_auc:.4f}\n")

print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=target_le.classes_))

# Step 2: Plot the Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

n_classes = len(np.unique(y_test))
y_test_binarized = label_binarize(y_test, classes=range(n_classes))

# Computing ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_pred_proba[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_binarized.ravel(), y_pred_proba.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot all ROC curves
plt.figure(figsize=(10, 8))

# Plot micro-average ROC curve
plt.plot(fpr["micro"], tpr["micro"],
         label=f"Micro-average ROC curve (area = {roc_auc['micro']:.2f})",
```

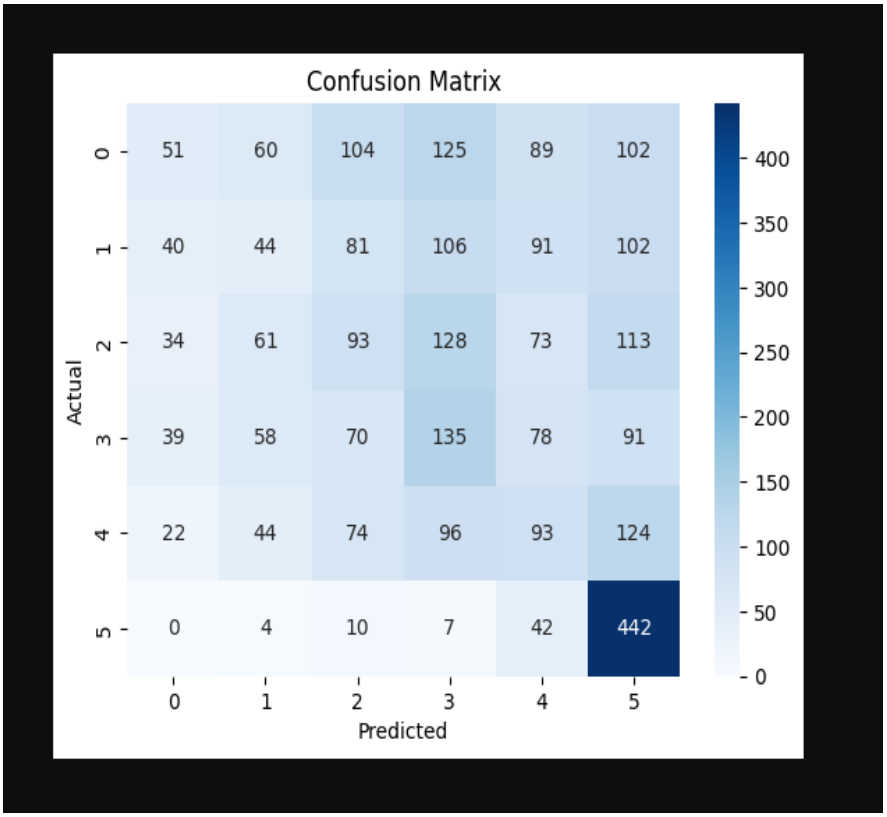
```
        color="deeppink", linestyle=":", linewidth=4)

    # Plot ROC curve for each class
    colors = ['aqua', 'darkorange', 'cornflowerblue', 'red', 'green']
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label=f"ROC curve of class {target_le.classes_[i]} (area = {roc_auc[i]:.2f})")

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Receiver Operating Characteristic (ROC) Curves - Random Forest")
    plt.legend(loc="lower right")
plt.show()
```

OUTPUT –

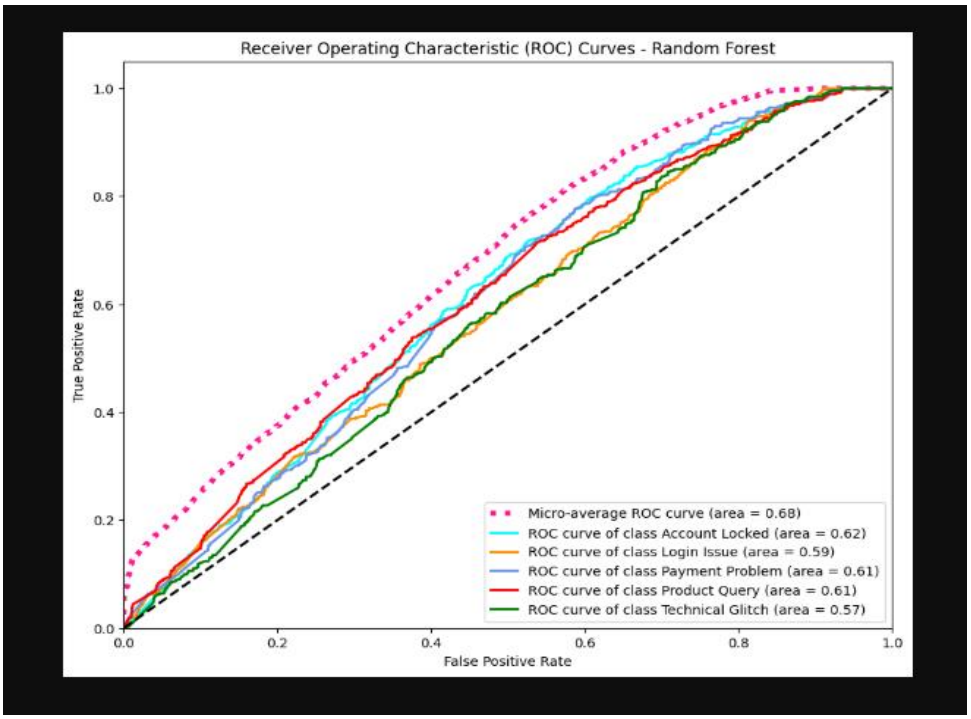
```
Accuracy: 29.3233%
Precision: 0.2581
Recall: 0.2932
ROC AUC Score: 0.6574
Classification Report:  precision    recall  f1-score   support

Account Locked       0.27      0.10      0.14       531
Login Issue          0.16      0.09      0.12       464
```

Payment Problem	0.22	0.19	0.20	502
Product Query	0.23	0.29	0.25	471
Technical Glitch	0.20	0.21	0.20	453
nan	0.45	0.88	0.60	505
accuracy	0.29	2926		
macro avg	0.26	0.29	0.25	2926
weighted avg	0.26	0.29	0.25	2926

CONFUSION MATRIX :



ROC CURVE :


Receiver Operating Characteristic (ROC) Curves - Random Forest
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Micro-average ROC curve (area = 0.68)
ROC curve of class Account Locked (area = 0.62)
ROC curve of class Login Issue (area = 0.59)
ROC curve of class Payment Problem (area = 0.61)
ROC curve of class Product Query (area = 0.61)
ROC curve of class Technical Glitch (area = 0.57)