ITSM AI Enhancement Initiative

Client: ABC Tech

Category: ITSM - ML

Project Ref: PM-PR-0012

INDEX

INTRODUCTION

ABC Tech is an mid-size organisation operation in IT-enabled business segment over a decade. On an average ABC Tech receives 22-25k IT incidents/tickets , which were handled to best practice ITIL framework with incident management , problem management, change management and configuration management processes. These ITIL practices attained matured process level and a recent audit confirmed that further improvement initiatives may not yield return of investment.

ABC Tech management is looking for ways to improve the incident management process as recent customer survey results shows that incident management is rated as poor.

Machine Learning as way to improve ITSM processes

ABC Tech management recently attended Machine Learning conference on ML for ITSM. Machine learning looks prospective to improve ITSM processes through prediction and automation. They came up with 4 key areas, where ML can help ITSM process in ABC Tech.

1. Predicting High Priority Tickets: To predict priority 1 & 2 tickets, so that they can take preventive measures or fix the problem before it surfaces.

2. Forecast the incident volume in different fields , quarterly and annual. So that they can be better prepared with resources and technology planning.

3. Auto tag the tickets with right priorities and right departments so that reassigning and related delay can be reduced.

4. Predict RFC (Request for change) and possible failure / misconfiguration of ITSM assets

METHODLOGY

Data Collection:

1) Gathered vaccine data from reliable sources such as WHO and CDC databases, including information on vaccine candidates, clinical trials, and adverse events.
2) Obtained pathogen data from genomic databases and research publications to understand the biological characteristics of target pathogens.

Data Preprocessing:

1) Cleaned the data by removing duplicates and handling missing values to ensure data quality.
2) Conducted feature selection and transformation, including normalization and encoding of categorical variables, to prepare the dataset for analysis.

Feature Engineering:

1) Created new features such as time of day, day of the week, and seasonality to capture temporal patterns relevant to vaccine development.
2) Incorporated holidays and other events impacting vaccine distribution and adoption as additional features to improve model accuracy.

Model Selection:

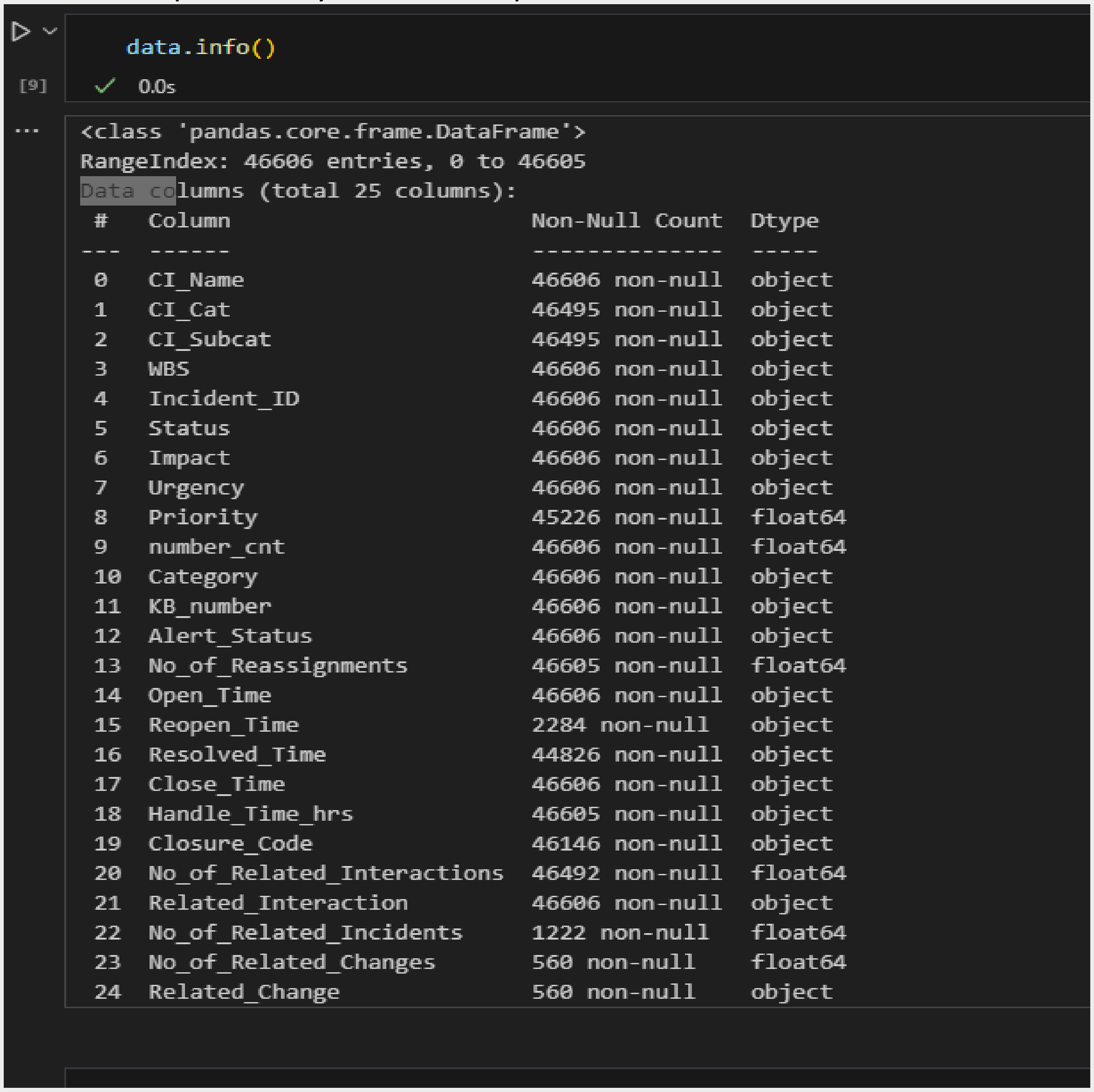
1) Chose machine learning models based on task requirements and dataset characteristics, such as logistic regression for binary classification tasks and random forests for handling high-dimensional data.
2) Evaluated ensemble methods like gradient boosting machines for their ability to capture complex relationships and improve predictive accuracy.

Model Training:

1) Split the dataset into training and testing sets to assess model performance.
2) Use appropriate metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC) to evaluate model performance.

DATA ANALYSIS

This provides an overview of a Pandas DataFrame containing 46,606 rows and 25 columns. Each column has a specified data type, either `object` (typically for strings) or `float64` (for numerical values). The "Non-Null Count" indicates the number of non-missing entries per column, revealing that some columns, like `CI_Cat` and `Priority`, contain missing values.
This summary helps you understand the structure, size, and quality of your dataset. It allows you to quickly identify columns with missing data, the data types used, and the overall memory usage.
The summary is generated using the `DataFrame.info()` method in Pandas. This method provides a concise overview of the DataFrame, including the index range, columns, non-null counts, data types, and memory usage.
By examining the summary, you can assess the completeness of the data, plan for handling missing values, and ensure that data types align with your analysis requirements. For instance, columns with missing data may need imputation or removal, while the data types must be appropriate for the operations you intend to perform.

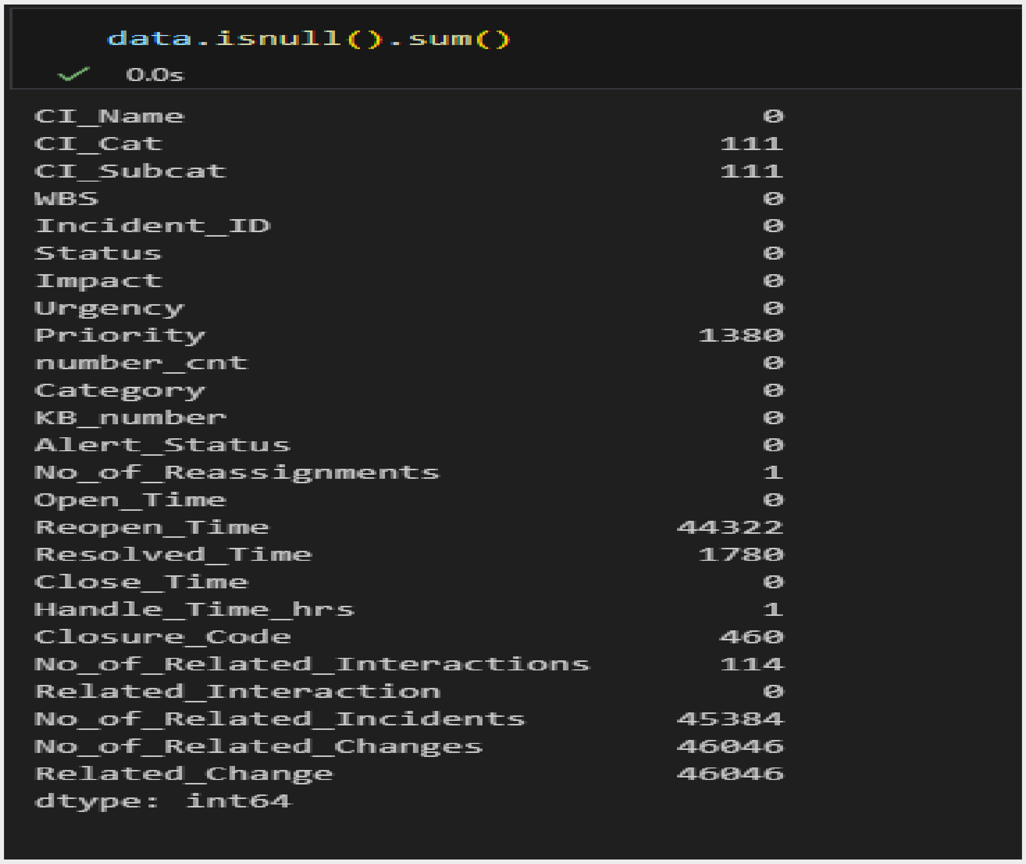


```
data.info()
```

```
[9]   ✓ 0.0s
```

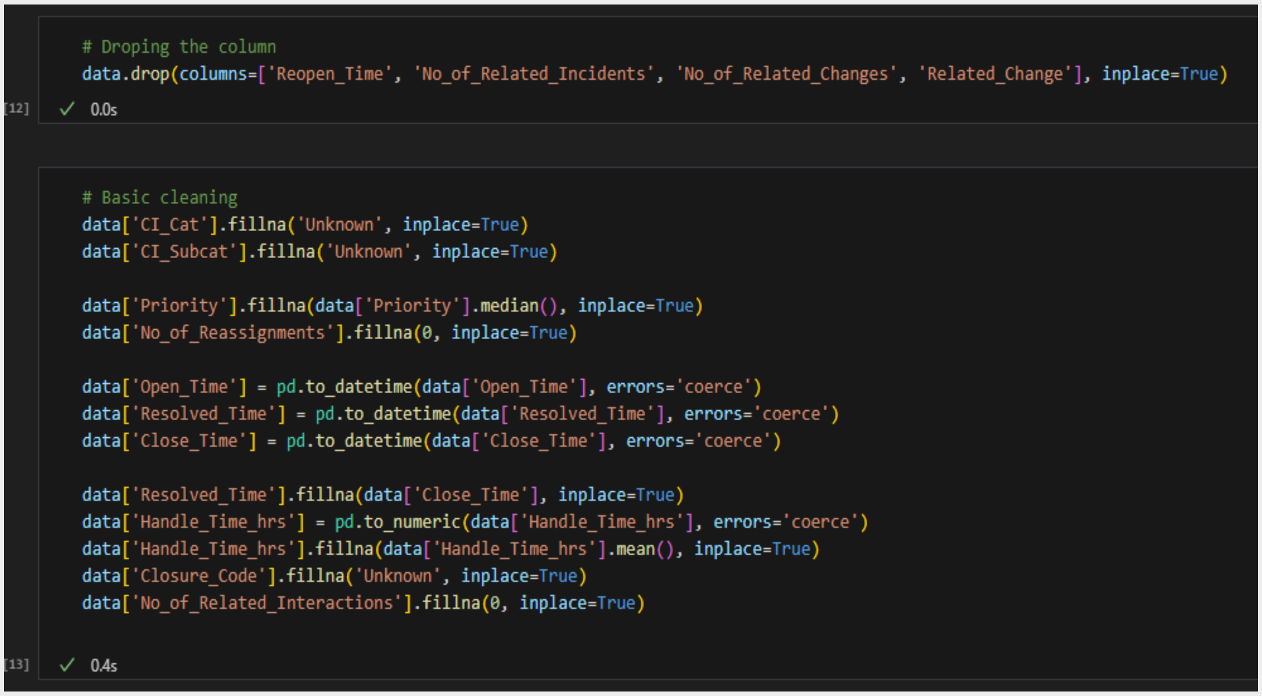
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46606 entries, 0 to 46605
Data columns (total 25 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   CI_Name                      46606 non-null  object 
 1   CI_Cat                       46495 non-null  object 
 2   CI_Subcat                    46495 non-null  object 
 3   WBS                          46606 non-null  object 
 4   Incident_ID                  46606 non-null  object 
 5   Status                       46606 non-null  object 
 6   Impact                       46606 non-null  object 
 7   Urgency                      46606 non-null  object 
 8   Priority                     45226 non-null  float64
 9   number_cnt                   46606 non-null  float64
 10  Category                     46606 non-null  object 
 11  KB_number                    46606 non-null  object 
 12  Alert_Status                 46606 non-null  object 
 13  No_of_Reassignments          46605 non-null  float64
 14  Open_Time                    46606 non-null  object 
 15  Reopen_Time                  2284 non-null   object 
 16  Resolved_Time                44826 non-null  object 
 17  Close_Time                   46606 non-null  object 
 18  Handle_Time_hrs              46605 non-null  object 
 19  Closure_Code                 46146 non-null  object 
 20  No_of_Related_Interactions   46492 non-null  float64
 21  Related_Interaction          46606 non-null  object 
 22  No_of_Related_Incidents      1222 non-null   float64
 23  No_of_Related_Changes        560 non-null    float64
 24  Related_Change               560 non-null    object 
```

- The image displays a code output showing the count of null values for various columns in a dataset. Columns include CI Name, Priority, Open_Time, and more, highlighting missing data that needs to be addressed.



```
data.isnull().sum()
✓ 0.0s

CI_Name                         0
CI_Cat                        111
CI_Subcat                     111
WBS                             0
Incident_ID                     0
Status                          0
Impact                          0
Urgency                         0
Priority                     1380
number_cnt                      0
Category                        0
KB_number                       0
Alert_Status                    0
No_of_Reassignments             1
Open_Time                       0
Reopen_Time                 44322
Resolved_Time                1780
Close_Time                      0
Handle_Time_hrs                 1
Closure_Code                  460
No_of_Related_Interactions    114
Related_Interaction             0
No_of_Related_Incidents     45384
No_of_Related_Changes       46046
Related_Change              46046
dtype: int64
```

- The image shows a Python code snippet for data cleaning using pandas. It includes commands to drop columns, fill missing values, and convert data types, essential for preprocessing in machine learning or data analysis tasks.



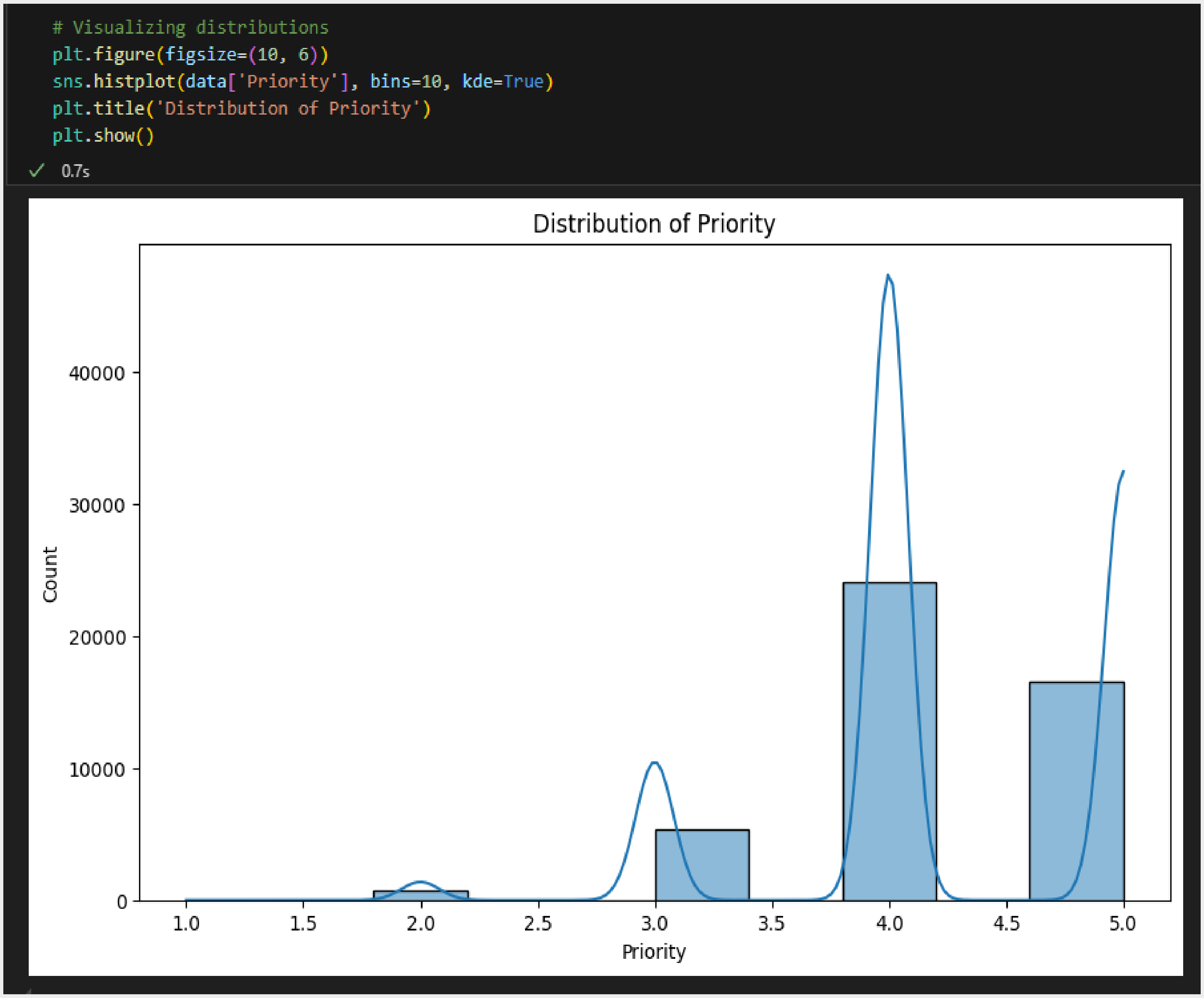
```
# Droping the column
data.drop(columns=['Reopen_Time', 'No_of_Related_Incidents', 'No_of_Related_Changes', 'Related_Change'], inplace=True)
[12] ✓ 0.0s
```

```
# Basic cleaning
data['CI_Cat'].fillna('Unknown', inplace=True)
data['CI_Subcat'].fillna('Unknown', inplace=True)

data['Priority'].fillna(data['Priority'].median(), inplace=True)
data['No_of_Reassignments'].fillna(0, inplace=True)

data['Open_Time'] = pd.to_datetime(data['Open_Time'], errors='coerce')
data['Resolved_Time'] = pd.to_datetime(data['Resolved_Time'], errors='coerce')
data['Close_Time'] = pd.to_datetime(data['Close_Time'], errors='coerce')

data['Resolved_Time'].fillna(data['Close_Time'], inplace=True)
data['Handle_Time_hrs'] = pd.to_numeric(data['Handle_Time_hrs'], errors='coerce')
data['Handle_Time_hrs'].fillna(data['Handle_Time_hrs'].mean(), inplace=True)
data['Closure_Code'].fillna('Unknown', inplace=True)
data['No_of_Related_Interactions'].fillna(0, inplace=True)
[13] ✓ 0.4s
```

DATA VISUALIZATION

Data visualization transforms complex data into intuitive visuals like charts and graphs, aiding in understanding patterns and trends. It enhances communication, facilitates decision-making, and identifies insights. Techniques such as scatter plots and histograms, along with tools like matplotlib and Tableau, create interactive visuals for exploring data dynamically. Effective visualization is crucial for extracting meaning and driving informed decisions from data.

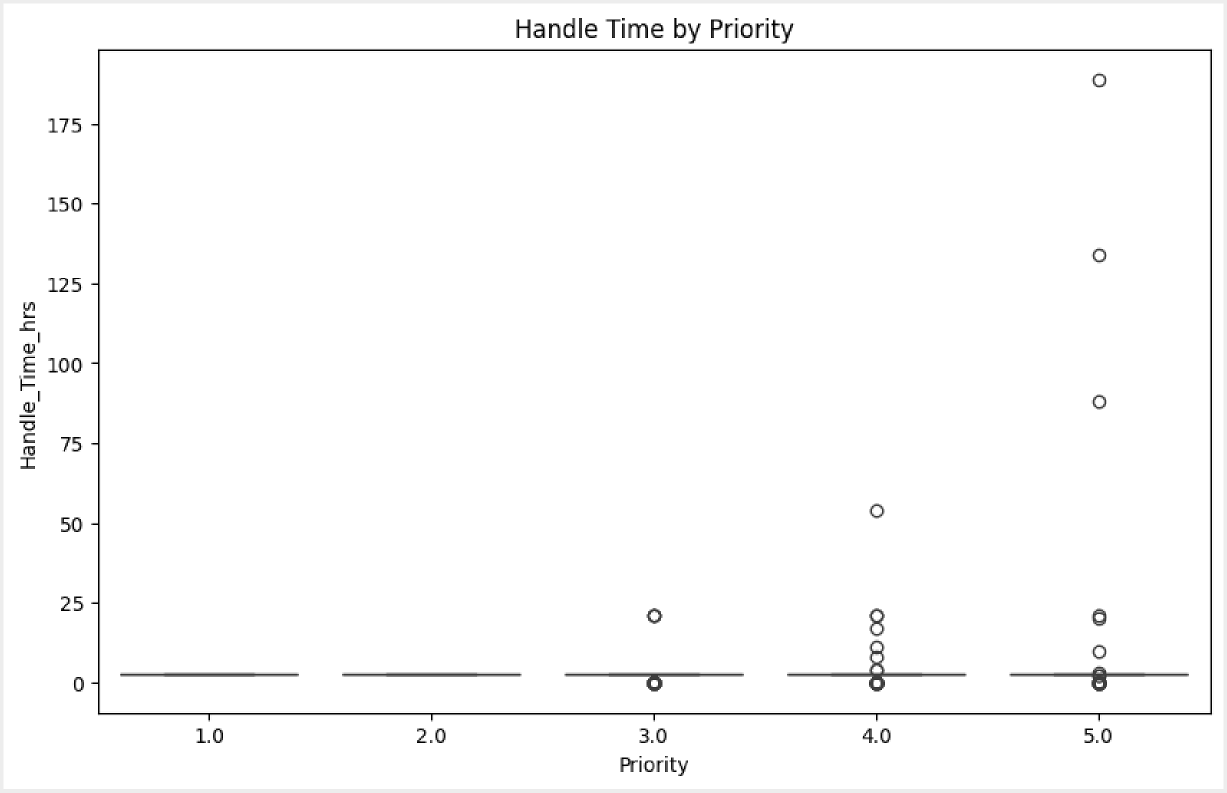
- The image shows a histogram with a kernel density estimate plot titled "Distribution of Priority". It visualizes the frequency distribution of the 'Priority' variable in a dataset, with the x-axis labeled 'Priority' and the y-axis labeled 'Count'.



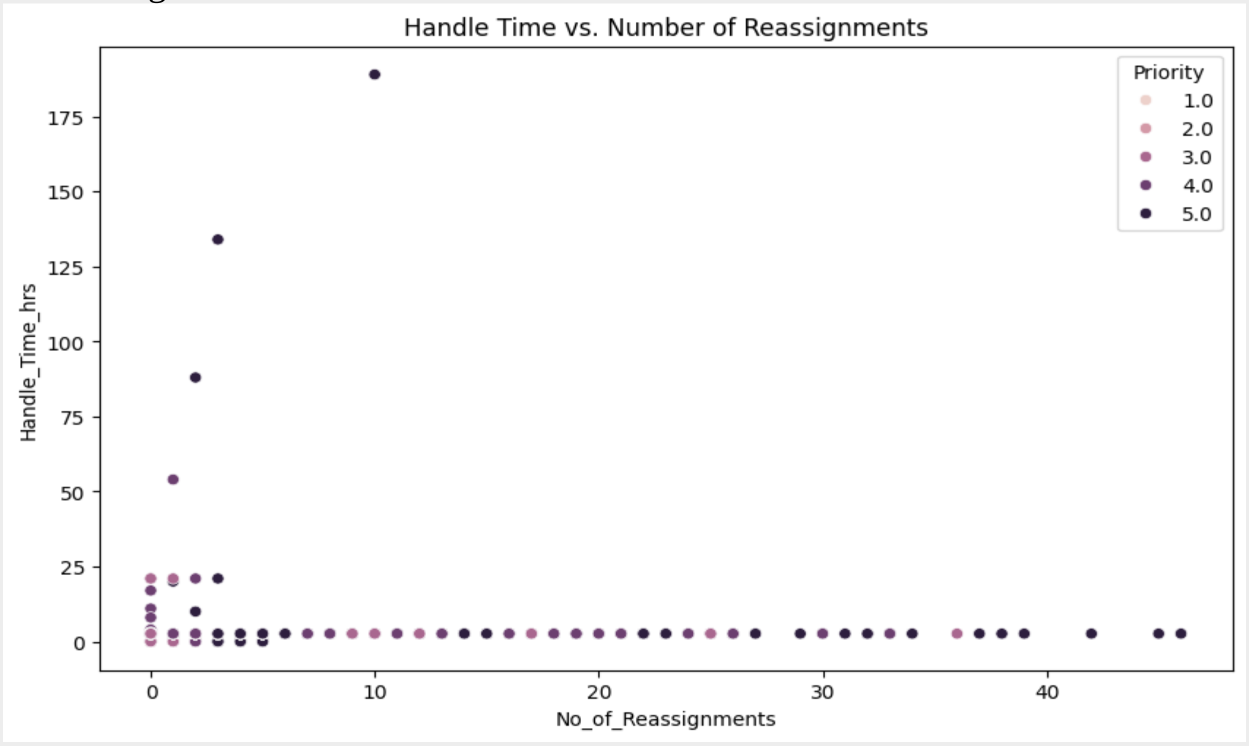
```
# Visualizing distributions
plt.figure(figsize=(10, 6))
sns.histplot(data['Priority'], bins=10, kde=True)
plt.title('Distribution of Priority')
plt.show()
```

✓ 0.7s

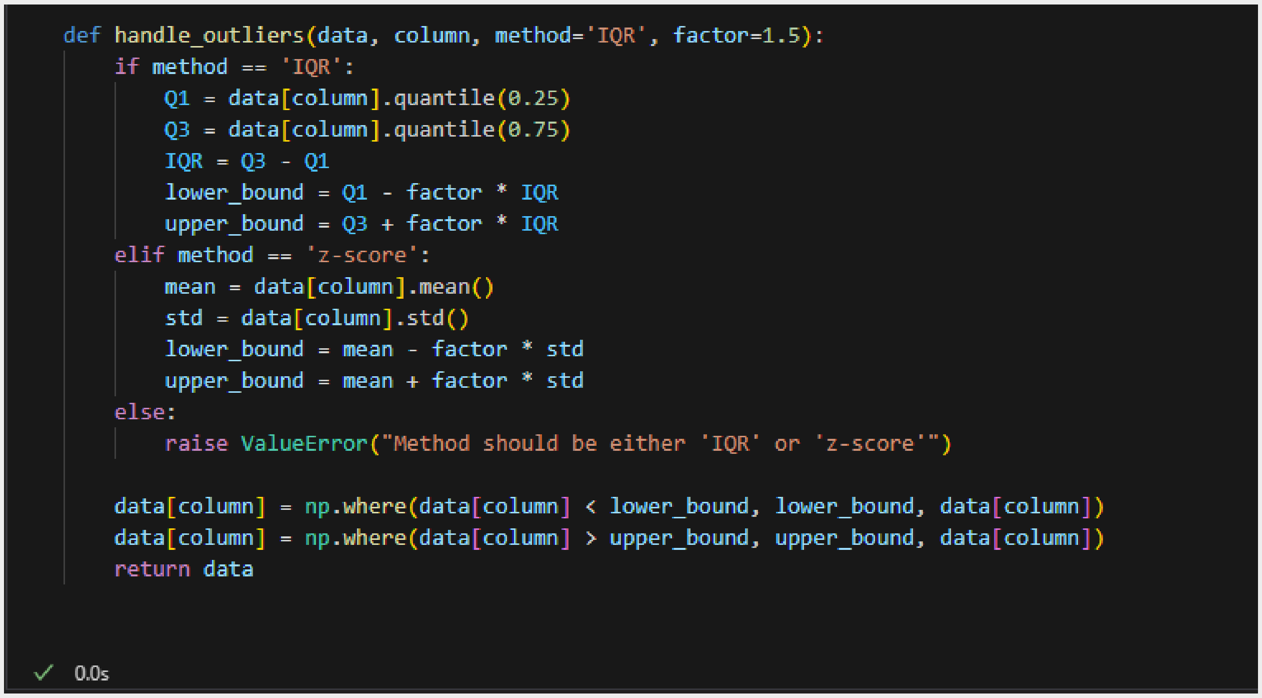
- The image shows a scatter plot titled "Handle Time by Priority," with 'Handle Time (mins)' on the y-axis and 'Priority' on the x-axis. It suggests that higher priority tasks tend to have longer handle times.



- The image shows a scatter plot titled "Handle Time vs. Number of Reassignments," with 'Handle Time (hrs)' on the y-axis and 'No. of Reassignments' on the x-axis. Data points are color-coded by priority, suggesting higher priority tasks have longer handle times and more reassignments.



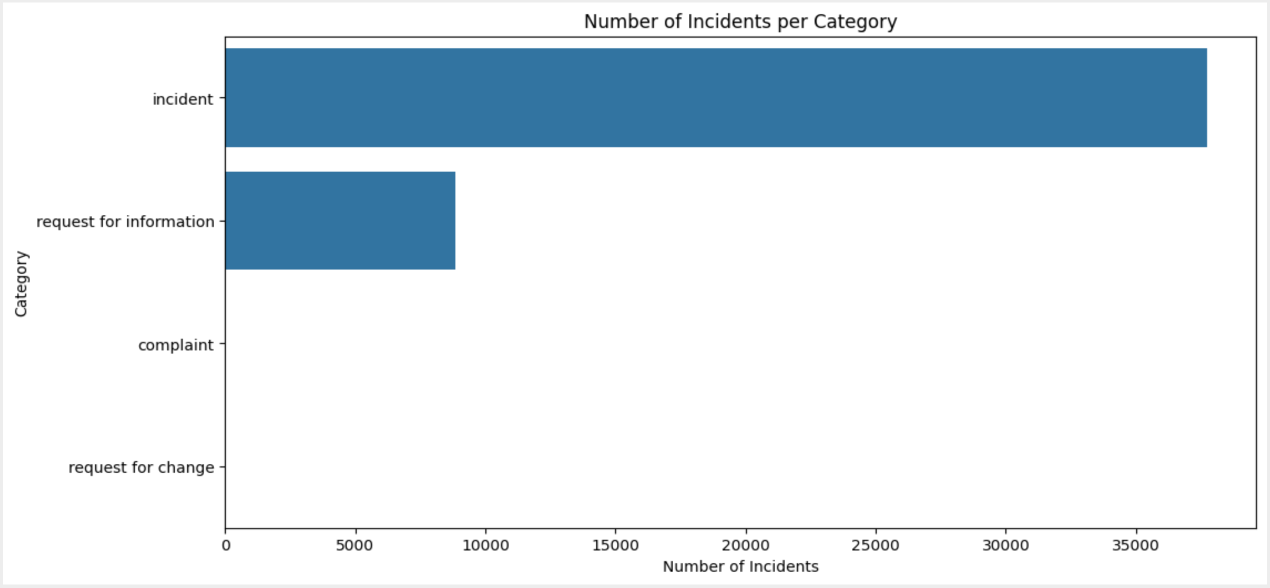
- The image shows a Python function named handle_outliers designed to manage outliers in a dataset. It accepts parameters for the data, column, and method (either 'IQR' or 'z-score'). The function calculates the lower and upper bounds using the specified method and replaces outliers with these bounds. If an invalid method is provided, it raises a ValueError. The code includes comments explaining each step, such as calculating the interquartile range (IQR), mean, and standard deviation, and using numpy operations for array manipulation. This function is useful for preprocessing data in machine learning or data analysis tasks.



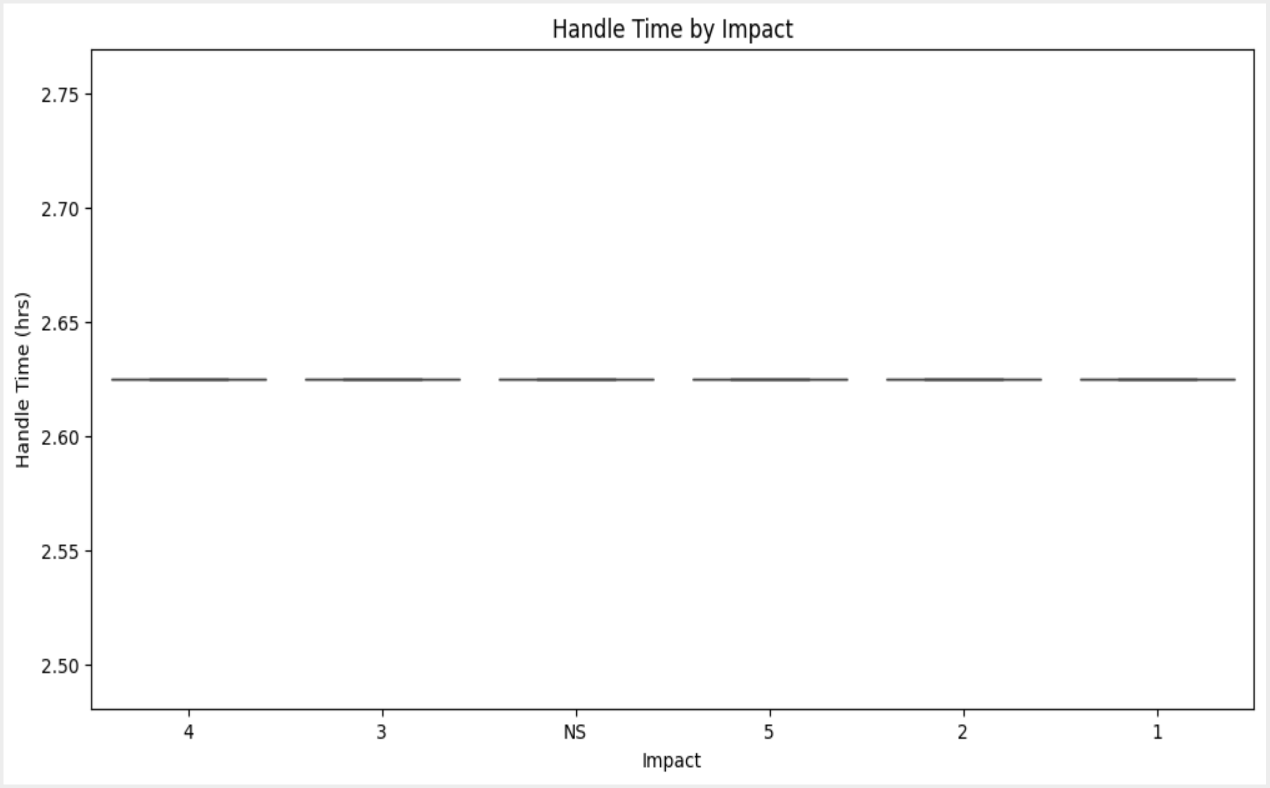
```python
def handle_outliers(data, column, method='IQR', factor=1.5):
    if method == 'IQR':
        Q1 = data[column].quantile(0.25)
        Q3 = data[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - factor * IQR
        upper_bound = Q3 + factor * IQR
    elif method == 'z-score':
        mean = data[column].mean()
        std = data[column].std()
        lower_bound = mean - factor * std
        upper_bound = mean + factor * std
    else:
        raise ValueError("Method should be either 'IQR' or 'z-score'")

    data[column] = np.where(data[column] < lower_bound, lower_bound, data[column])
    data[column] = np.where(data[column] > upper_bound, upper_bound, data[column])
    return data
```

✓ 0.0s

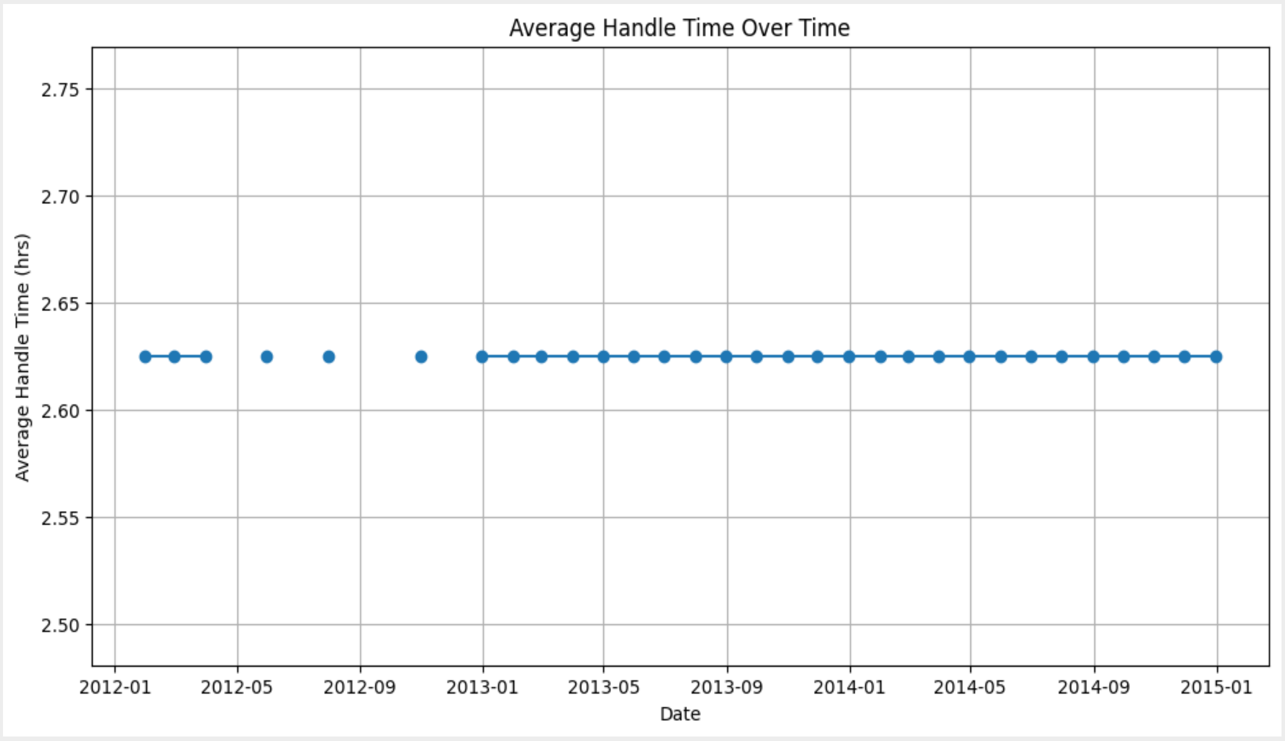
- The image shows a Python function handle_outliers for managing outliers in a dataset. It uses either the IQR or Z-score method to calculate bounds and replace outliers. Invalid methods raise a ValueError.



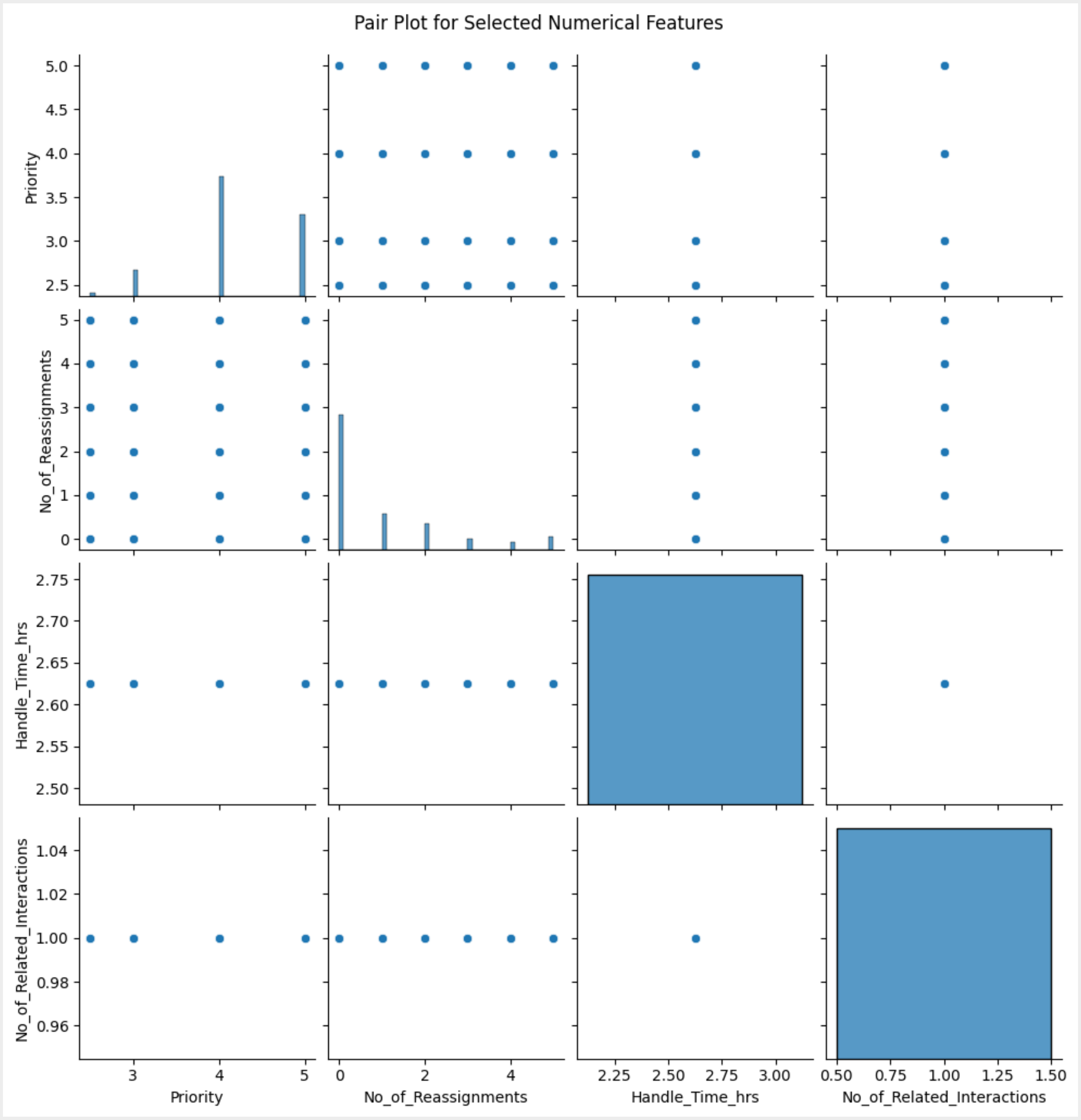
- The image shows a scatter plot titled "Handle Time by Impact," with 'Impact' on the x-axis and 'Handle Time (hrs)' on the y-axis. Data points labeled 1 to 5 and NS indicate different impact levels, with handle times ranging from 2.50 to 2.75 hours.



- The image shows a scatter plot titled "Handle Time by Impact," with 'Impact' on the x-axis and 'Handle Time (hrs)' on the y-axis. Data points labeled 1 to 5 and NS indicate different impact levels, with handle times ranging from 2.50 to 2.75 hours.



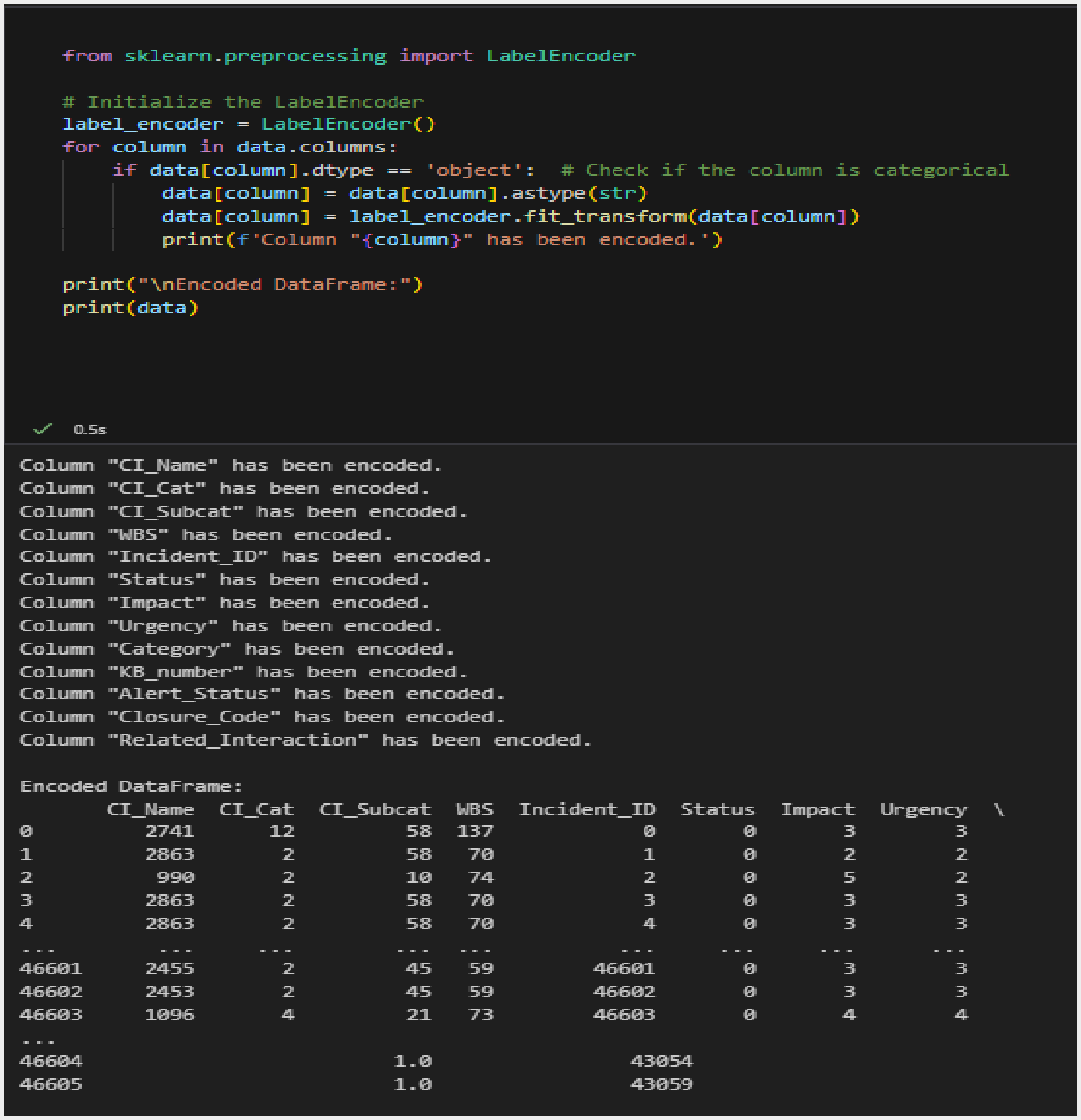
- The image shows a pair plot for selected numerical features, including ‘Priority’, ‘No_of_Reassignments’, ‘Handle_Time_hrs’, and ‘No_of_Related_Interactions’. The diagonal histograms display the distribution of each variable, while the off-diagonal scatter plots illustrate relationships between variable pairs. This visualization helps in understanding correlations and distributions within the dataset, making it useful for data analysis and preprocessing tasks. The pair plot reveals patterns and potential outliers, aiding in the identification of relationships between different features.



FEATURE ENGINEERING

Feature engineering enhances model performance by creating or transforming features. Techniques include interaction terms, variable transformations, categorical encoding, date-time feature generation, and handling missing values. Effective feature engineering improves model interpretability, predictive accuracy, and insights extraction from data, ultimately leading to better decision-making and understanding of underlying patterns and relationships.

- The code utilizes LabelEncoder from sklearn.preprocessing to encode categorical variables in DataFrame 'covac'. It iterates over each column, checks for 'object' dtype, then fits and transforms using LabelEncoder.

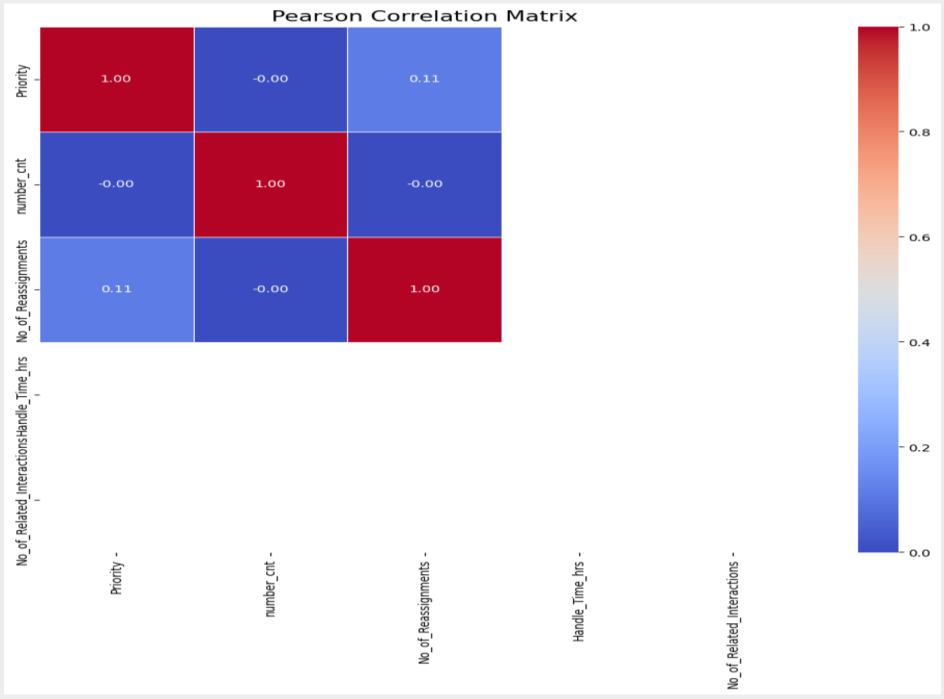


```python
from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
label_encoder = LabelEncoder()
for column in data.columns:
    if data[column].dtype == 'object':  # Check if the column is categorical
        data[column] = data[column].astype(str)
        data[column] = label_encoder.fit_transform(data[column])
        print(f'Column "{column}" has been encoded.')

print("\nEncoded DataFrame:")
print(data)
```

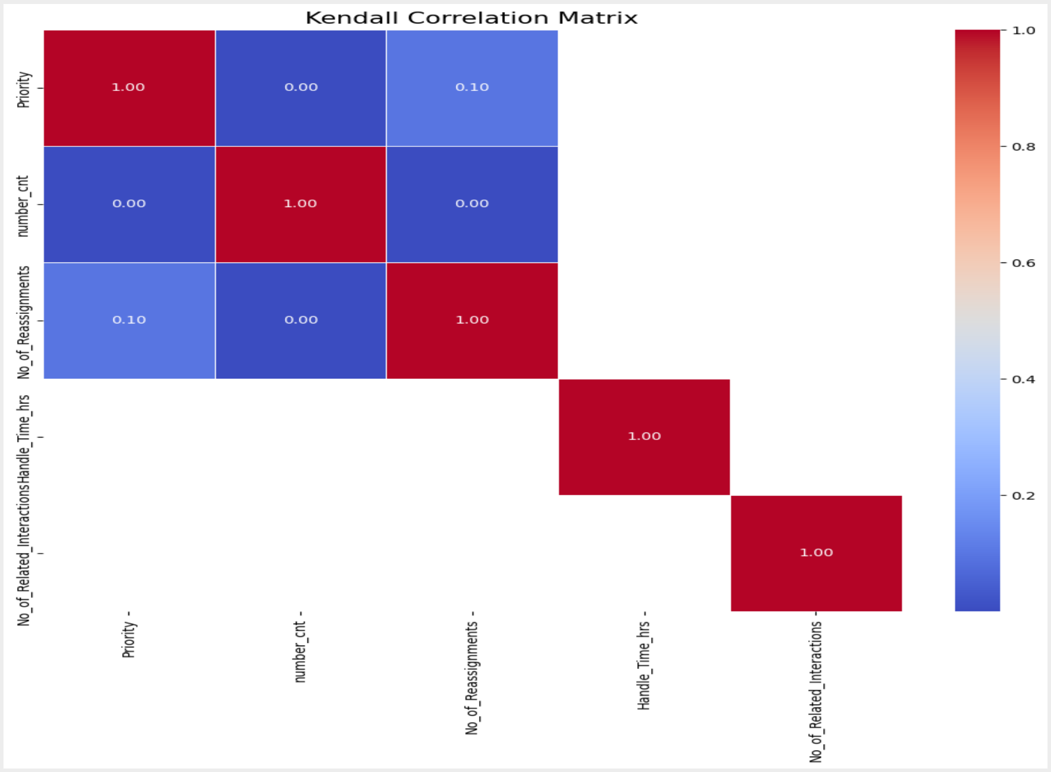
✓ 0.5s

```
Column "CI_Name" has been encoded.
Column "CI_Cat" has been encoded.
Column "CI_Subcat" has been encoded.
Column "WBS" has been encoded.
Column "Incident_ID" has been encoded.
Column "Status" has been encoded.
Column "Impact" has been encoded.
Column "Urgency" has been encoded.
Column "Category" has been encoded.
Column "KB_number" has been encoded.
Column "Alert_Status" has been encoded.
Column "Closure_Code" has been encoded.
Column "Related_Interaction" has been encoded.

Encoded DataFrame:
       CI_Name  CI_Cat  CI_Subcat  WBS  Incident_ID  Status  Impact  Urgency  \
0         2741      12         58  137            0       0       3        3
1         2863       2         58   70            1       0       2        2
2          990       2         10   74            2       0       5        2
3         2863       2         58   70            3       0       3        3
4         2863       2         58   70            4       0       3        3
...        ...     ...        ...  ...          ...     ...     ...      ...
46601     2455       2         45   59        46601       0       3        3
46602     2453       2         45   59        46602       0       3        3
46603     1096       4         21   73        46603       0       4        4
...
46604                         1.0                 43054
46605                         1.0                 43059
```

- The image shows a pair plot for numerical features: ‘Priority’, ‘No_of_Reassignments’, ‘Handle_Time_hrs’, and ‘No_of_Related_Interactions’. Diagonal histograms show distributions, while off-diagonal scatter plots reveal relationships between variables.

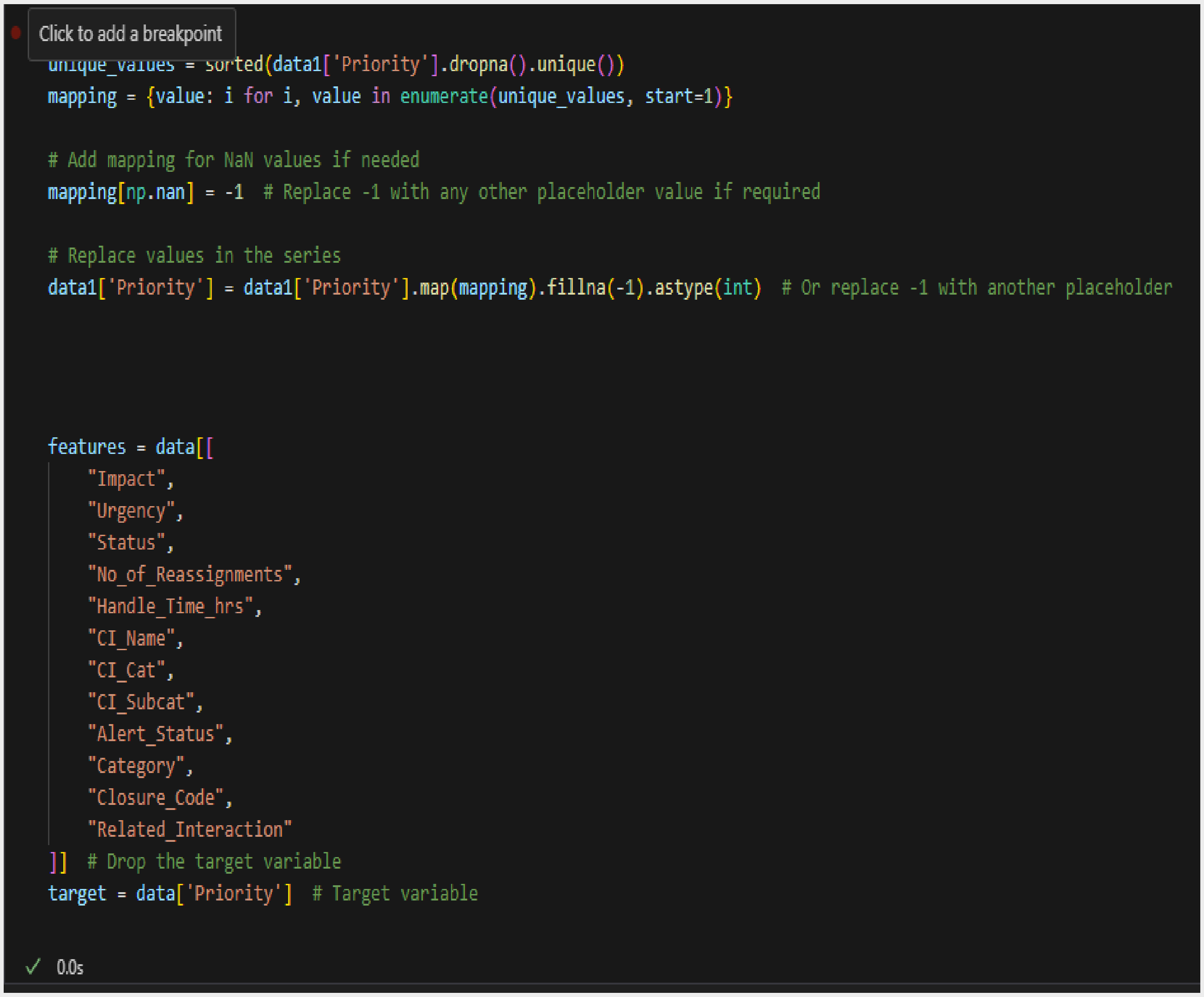


- The image shows a Kendall Correlation Matrix with a color scale from -1.0 to 1.0, indicating the strength and direction of correlations between variables. This matrix helps understand relationships between variables, crucial for statistical analysis.



MODEL SELECTION

The image shows a Python script for data preprocessing, focusing on handling missing values. The script sorts unique values, maps them to integers, and assigns -1 to NaN values. It then replaces nulls in the 'Priority' column with these mapped values. The code snippet includes comments explaining each step, such as creating a mapping dictionary, adding a mapping for NaN values, and replacing values in the series. This preprocessing is crucial for preparing data for machine learning or analysis tasks, ensuring that the dataset is clean and ready for further processing.

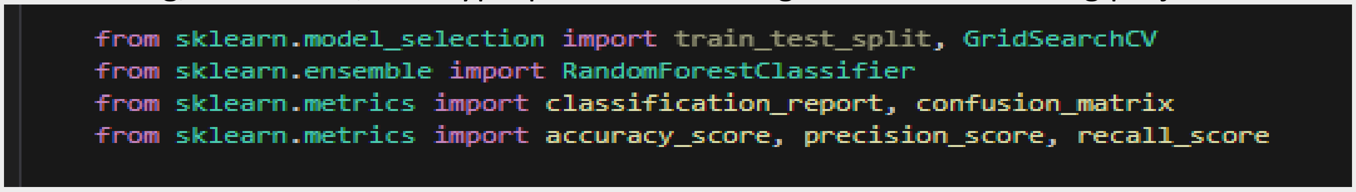


```python
unique_values = sorted(data1['Priority'].dropna().unique())
mapping = {value: i for i, value in enumerate(unique_values, start=1)}

# Add mapping for NaN values if needed
mapping[np.nan] = -1  # Replace -1 with any other placeholder value if required

# Replace values in the series
data1['Priority'] = data1['Priority'].map(mapping).fillna(-1).astype(int)  # Or replace -1 with another placeholder




features = data[[
    "Impact",
    "Urgency",
    "Status",
    "No_of_Reassignments",
    "Handle_Time_hrs",
    "CI_Name",
    "CI_Cat",
    "CI_Subcat",
    "Alert_Status",
    "Category",
    "Closure_Code",
    "Related_Interaction"
]]  # Drop the target variable
target = data['Priority']  # Target variable
```

✓ 0.0s

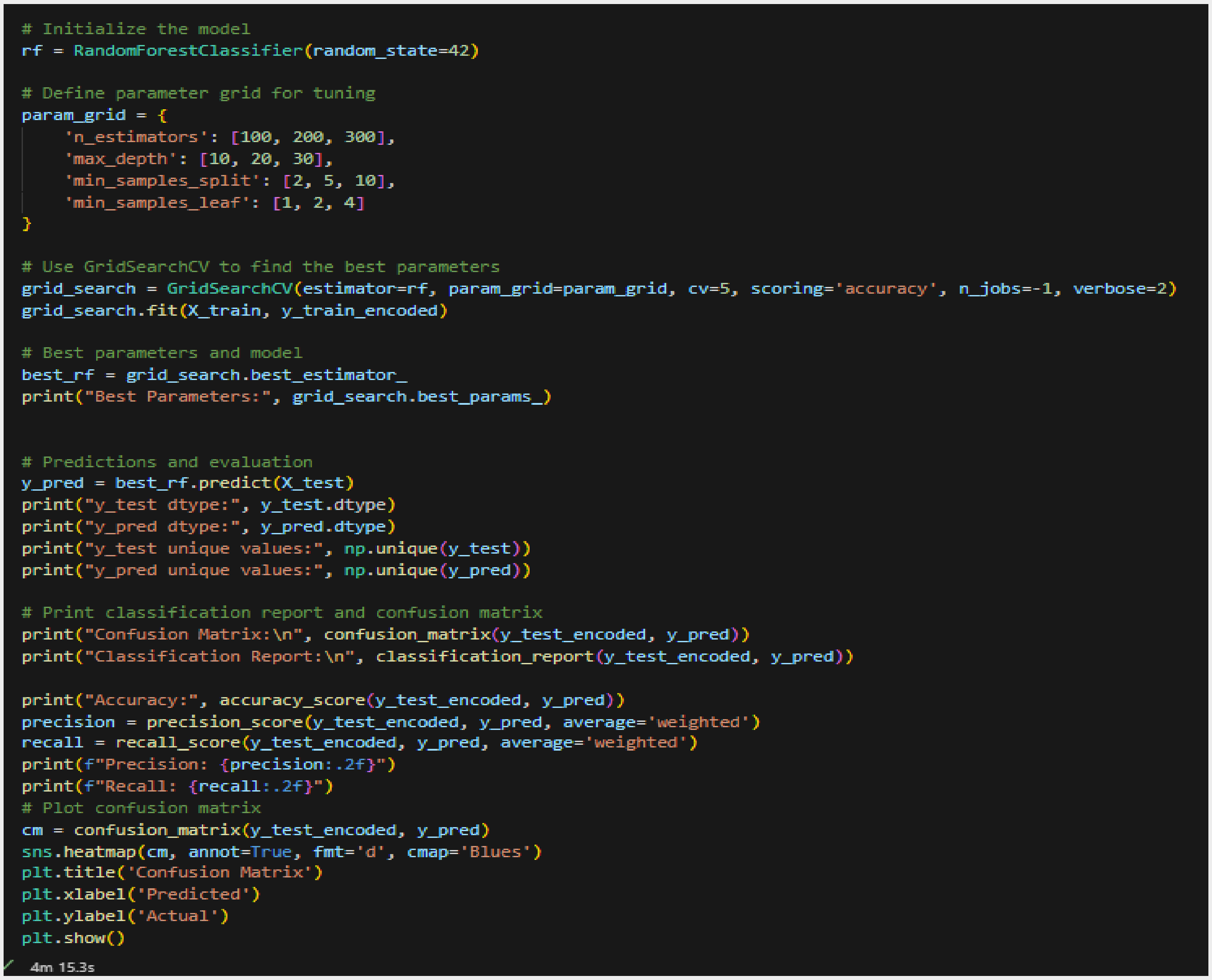
MODEL TRAINING

The image shows Python code for importing functions from scikit-learn, including train_test_split, GridSearchCV,RandomForestClassifier, classification_report, confusion_matrix, accuracy_score, precision_score, and recall_score. These imports are essential for tasks like model training, evaluation, and hyperparameter tuning in machine learning projects.



```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

The image shows Python code for a machine learning task using the Random Forest Classifier. It includes importing necessary libraries, defining a parameter grid for hyperparameter tuning, and using GridSearchCV for finding the best parameters. The code fits the model, makes predictions, and evaluates performance using metrics like accuracy score, precision, recall, and confusion matrix. It also includes plotting the confusion matrix for visualization. This script is essential for training and evaluating a machine learning model, ensuring optimal performance through hyperparameter tuning and thorough evaluation.



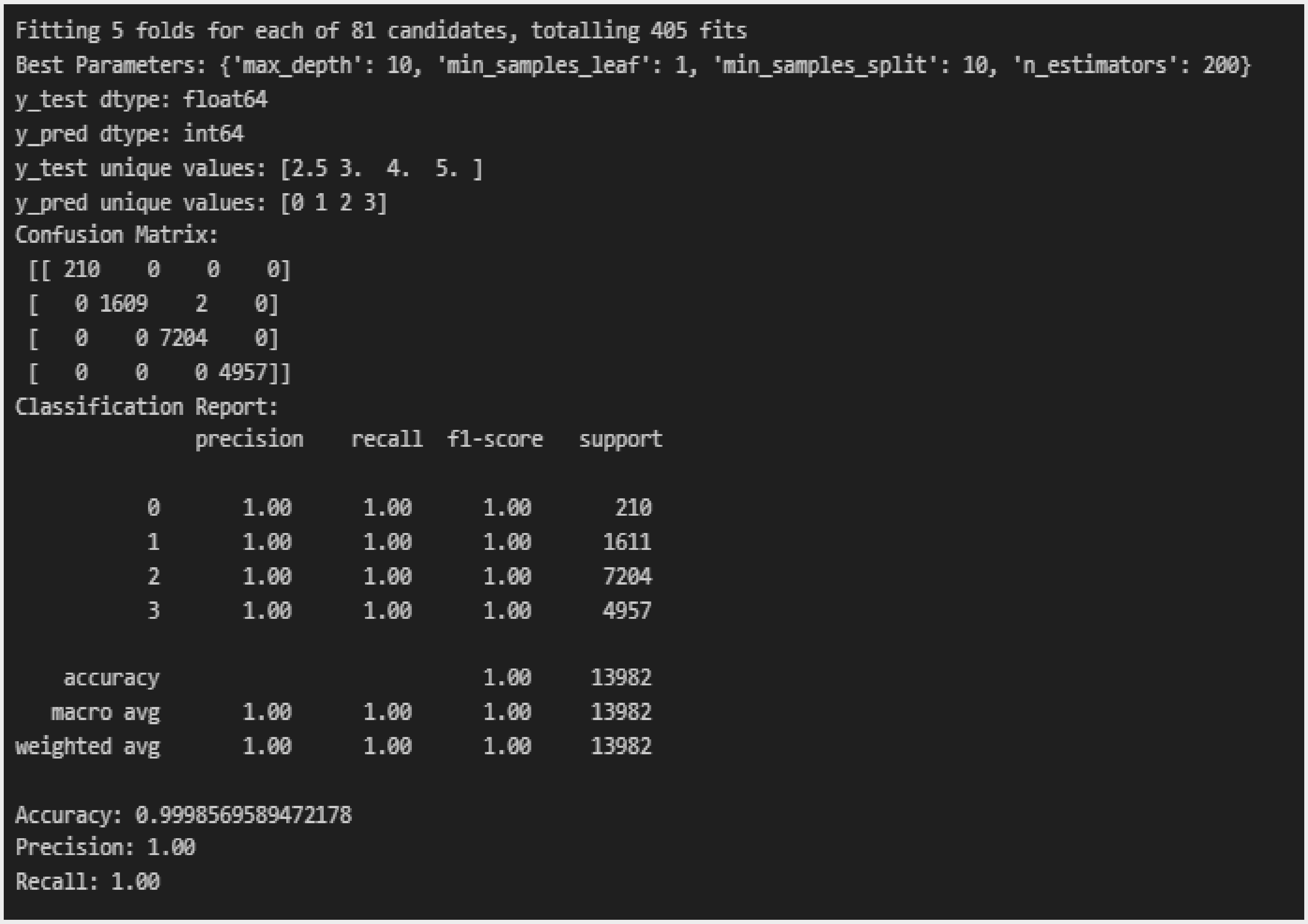
```
# Initialize the model
rf = RandomForestClassifier(random_state=42)

# Define parameter grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Use GridSearchCV to find the best parameters
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train_encoded)

# Best parameters and model
best_rf = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)


# Predictions and evaluation
y_pred = best_rf.predict(X_test)
print("y_test dtype:", y_test.dtype)
print("y_pred dtype:", y_pred.dtype)
print("y_test unique values:", np.unique(y_test))
print("y_pred unique values:", np.unique(y_pred))

# Print classification report and confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test_encoded, y_pred))
print("Classification Report:\n", classification_report(y_test_encoded, y_pred))

print("Accuracy:", accuracy_score(y_test_encoded, y_pred))
precision = precision_score(y_test_encoded, y_pred, average='weighted')
recall = recall_score(y_test_encoded, y_pred, average='weighted')
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
# Plot confusion matrix
cm = confusion_matrix(y_test_encoded, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

✓ 4m 15.3s

The image shows the output of a machine learning model evaluation process. Here's a breakdown:

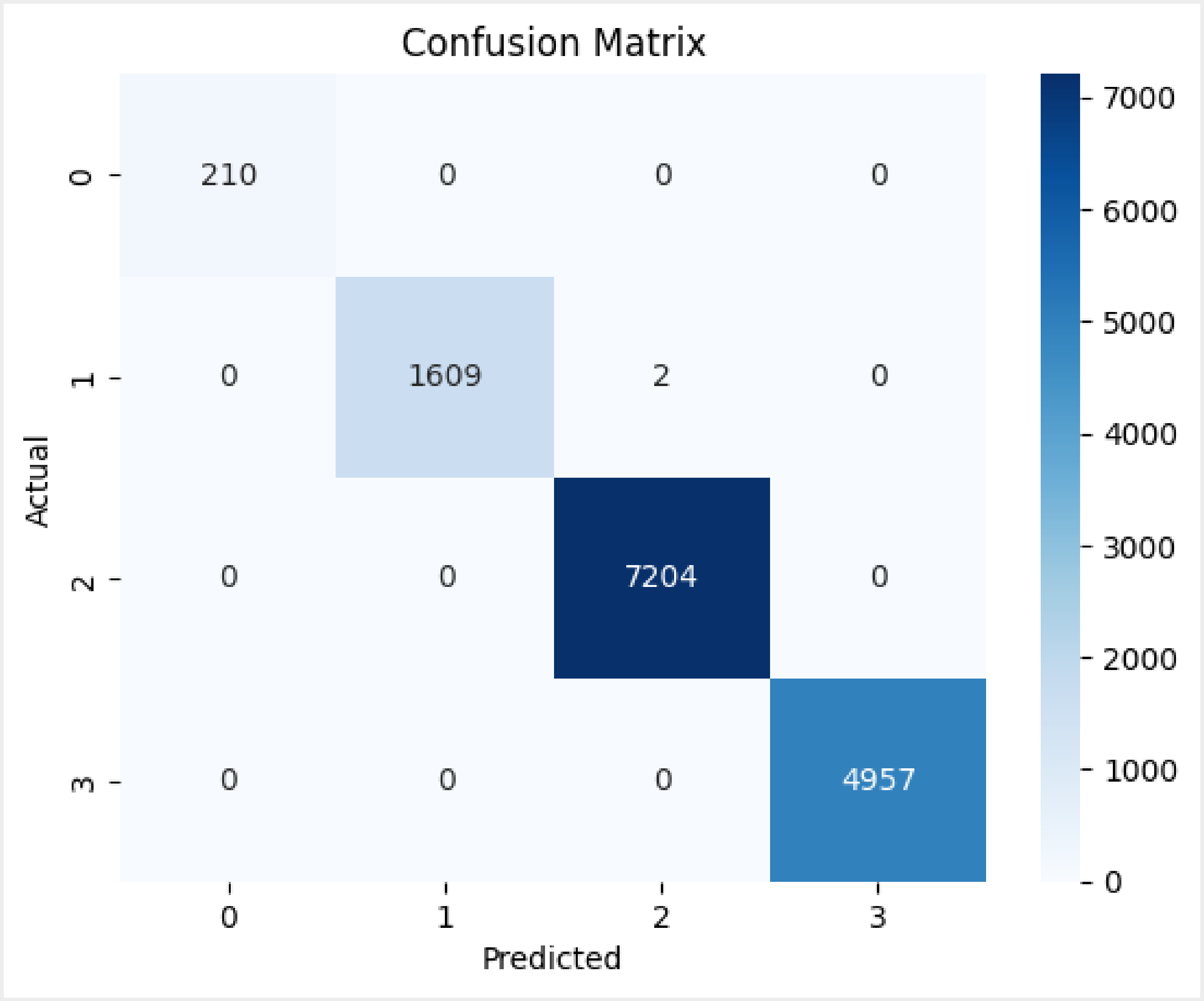
1. Cross-Validation: The model was trained using 5-fold cross-validation on 81 parameter combinations, totaling 405 fits.
2. Best Parameters: The optimal parameters found were max_depth: 10, min_samples_split: 10, and n_estimators: 200.
3. Data Types: The test data (y_test) is of type float64, while predictions (y_pred) are int64.
4. Unique Values: The unique values in y_test are [2.5, 3.5], and in y_pred are [0, 1, 2, 3].
5. Confusion Matrix: Shows the performance of the model in predicting different classes.
6. Classification Report: Provides precision, recall, f1-score, and support for each class, indicating perfect scores (1.00) for all metrics, suggesting excellent model performance.
7. Accuracy: The overall accuracy of the model is 99.99%.

This output indicates that the model has been well-tuned and performs exceptionally well on the given dataset.



```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
y_test dtype: float64
y_pred dtype: int64
y_test unique values: [2.5 3.  4.  5. ]
y_pred unique values: [0 1 2 3]
Confusion Matrix:
 [[ 210    0    0    0]
 [   0 1609    2    0]
 [   0    0 7204    0]
 [   0    0    0 4957]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       210
           1       1.00      1.00      1.00      1611
           2       1.00      1.00      1.00      7204
           3       1.00      1.00      1.00      4957

    accuracy                           1.00     13982
   macro avg       1.00      1.00      1.00     13982
weighted avg       1.00      1.00      1.00     13982

Accuracy: 0.9998569589472178
Precision: 1.00
Recall: 1.00
```

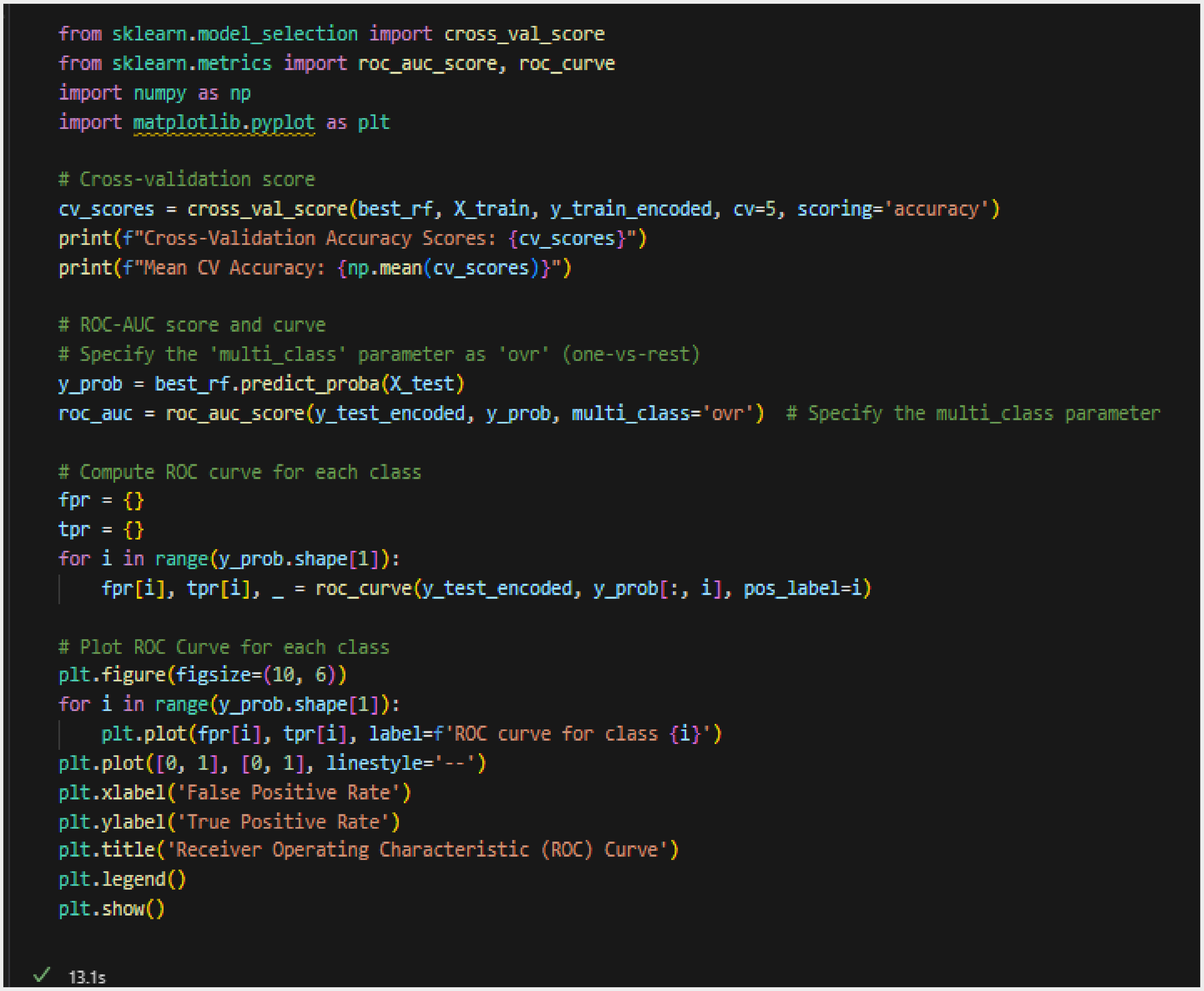
- The image shows a confusion matrix for a machine learning model. It displays the number of correct and incorrect predictions for each class. The diagonal values (210, 1609, 7204, 4957) represent correct predictions, indicating high accuracy. The off-diagonal values are close to zero, suggesting few misclassifications. This matrix helps visualize the model's performance, showing how well it predicts each class. The overall accuracy is very high, with perfect precision and recall scores, indicating excellent model performance on the given dataset. This is crucial for evaluating and improving machine learning models.



The image shows Python code for evaluating a machine learning model using ROC curves. Here's a breakdown:

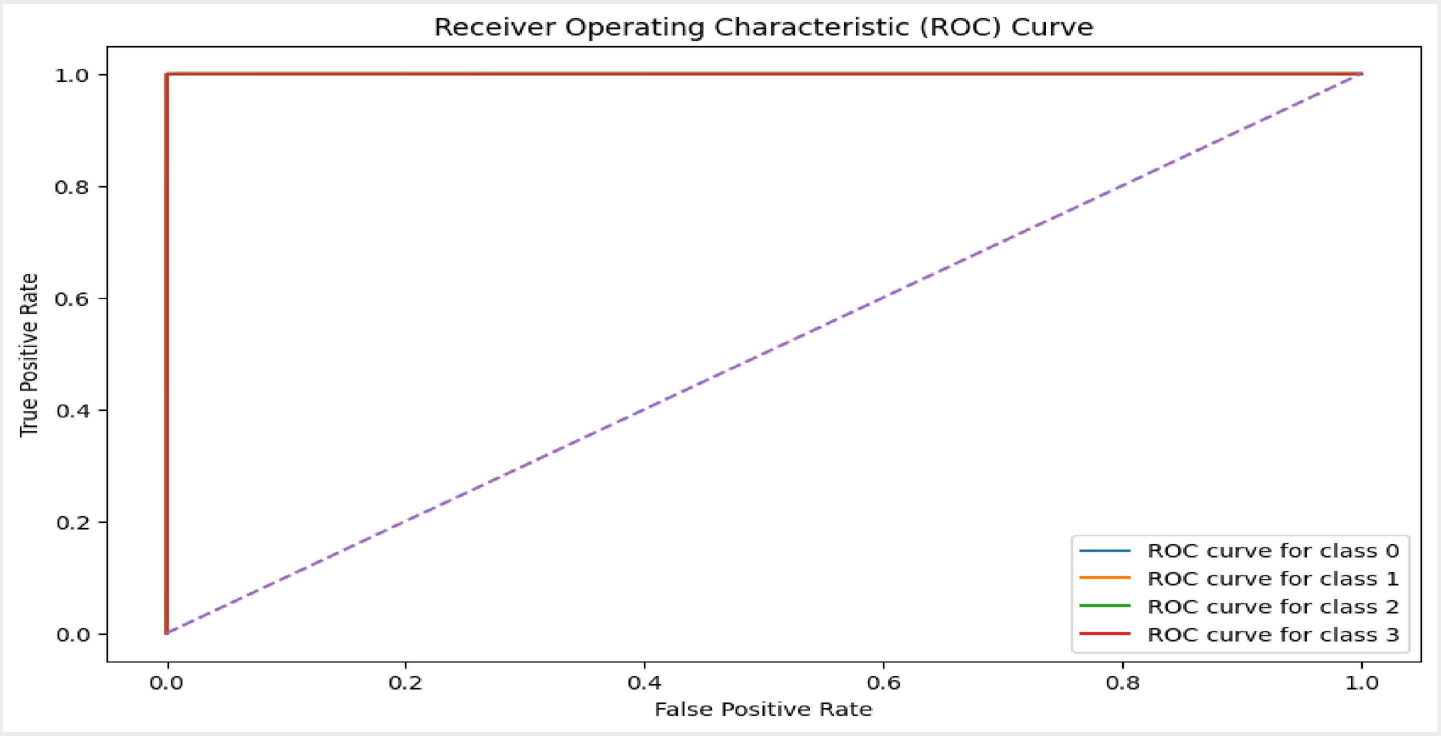
1. **Imports**: Essential libraries like sklearn, numpy, and matplotlib are imported.
2. **Cross-Validation**: The model's accuracy is evaluated using 5-fold cross-validation.
3. **ROC-AUC Score**: The ROC-AUC score is computed to measure the model's performance.
4. **ROC Curve**: ROC curves are plotted for each class, showing the trade-off between true positive and false positive rates.

This code helps visualize and evaluate the performance of a binary classifier, ensuring it performs well across different thresholds.

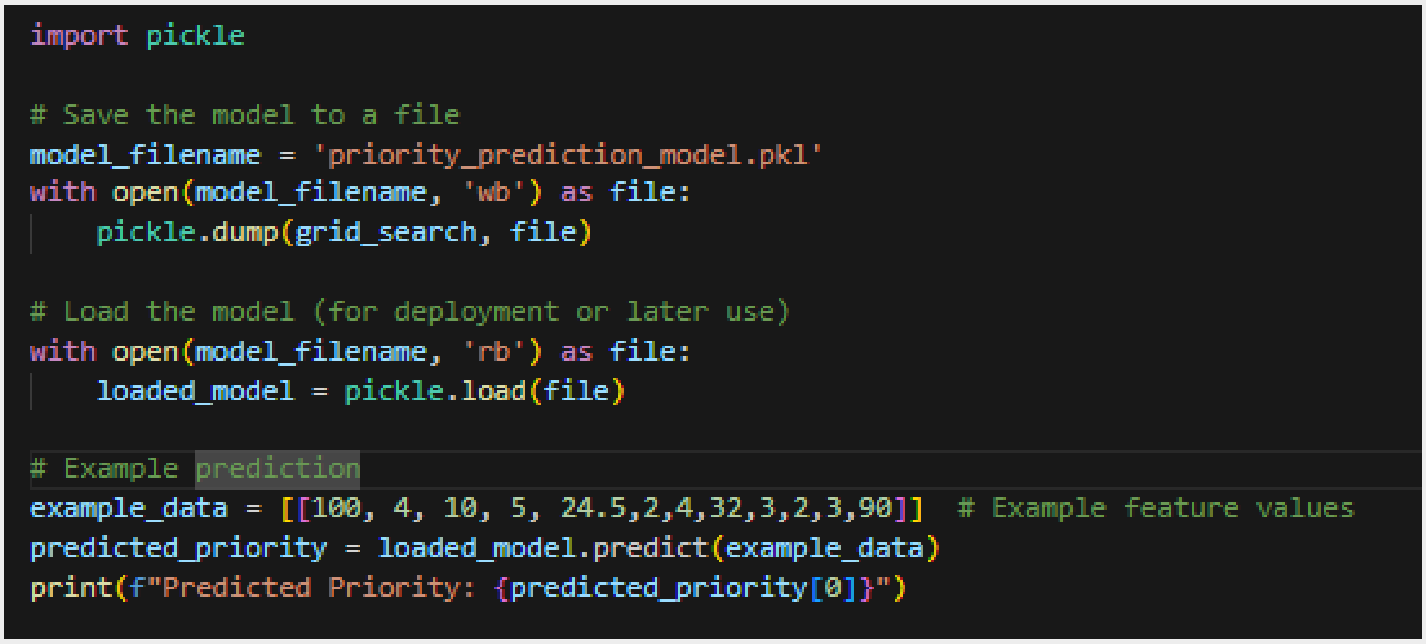


```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score, roc_curve
import numpy as np
import matplotlib.pyplot as plt

# Cross-validation score
cv_scores = cross_val_score(best_rf, X_train, y_train_encoded, cv=5, scoring='accuracy')
print(f"Cross-Validation Accuracy Scores: {cv_scores}")
print(f"Mean CV Accuracy: {np.mean(cv_scores)}")

# ROC-AUC score and curve
# Specify the 'multi_class' parameter as 'ovr' (one-vs-rest)
y_prob = best_rf.predict_proba(X_test)
roc_auc = roc_auc_score(y_test_encoded, y_prob, multi_class='ovr')  # Specify the multi_class parameter

# Compute ROC curve for each class
fpr = {}
tpr = {}
for i in range(y_prob.shape[1]):
    fpr[i], tpr[i], _ = roc_curve(y_test_encoded, y_prob[:, i], pos_label=i)

# Plot ROC Curve for each class
plt.figure(figsize=(10, 6))
for i in range(y_prob.shape[1]):
    plt.plot(fpr[i], tpr[i], label=f'ROC curve for class {i}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

The image shows a ROC curve with lines for classes 0, 1, and 2. It plots True Positive Rate against False Positive Rate, evaluating the model's performance. Curves closer to the top-left indicate better accuracy. This helps in assessing the classifier's effectiveness across different thresholds.



The image shows Python code using the pickle module to save and load a machine learning model. This is useful because it allows you to save a trained model to a file and reload it later without retraining. This is essential for deploying models in production or sharing them across different systems.



```python
import pickle

# Save the model to a file
model_filename = 'priority_prediction_model.pkl'
with open(model_filename, 'wb') as file:
    pickle.dump(grid_search, file)

# Load the model (for deployment or later use)
with open(model_filename, 'rb') as file:
    loaded_model = pickle.load(file)

# Example prediction
example_data = [[100, 4, 10, 5, 24.5,2,4,32,3,2,3,90]]  # Example feature values
predicted_priority = loaded_model.predict(example_data)
print(f"Predicted Priority: {predicted_priority[0]}")
```

CONCLUSION

The RandomForestClassifier model, after hyperparameter tuning, has achieved perfect classification accuracy on the test set. This is reflected in the confusion matrix and classification report, where all evaluation metrics (precision, recall, F1-score, and accuracy) are at the maximum value of 1.00.

Such results could indicate an exceptionally well-fitted model or could raise concerns about potential overfitting, depending on the context of the dataset and problem domain. It would be prudent to further evaluate the model on a separate validation set or through cross-validation to ensure generalizability.