ILLINOIS INSTITUTE OF TECHNOLOGY

CSP574-Data Preparation and Analysis

Final Project Report

Enhancing Predictive Analytics with Random Forest: A

Group Members:

S.No	Student Name	CWID
1.	Nikhitha Chintakuntla	A20561244
2.	Chilla Sai Krishna Reddy	A20563553
3.	Ravi Tarun Dasari	A20564217
4.	Anu Singh	A20568373
5.	Isha Vikrant Gaonkar	A20585341

Under the Guidance of
Prof. Jawahar J. Panchal

Date: December 1st 2024

Table of Contents

Abstract

The goal of this project is to develop a reliable machine learning pipeline that can predict a categorical target variable based on a dataset containing more than 1.2 million records and numerical features.The pipeline will involve extensive preprocessing ,such as addressing missing data,normalization,encoding and outlier detection, to ensure clean and consistent input.Exploratory Data Analysis will be carried out to identify trends,visualize feature interactions,and guide the selection of relevant features.The strategy of having two parallel pipelines was used, and thus, the models created here are intended for PMML and ONNX deployment.

Before data training and testing, preprocessing involved data cleansing, normalizing, and encoding, which were imperative to data modeling. Concerning class imbalance, the strategies employed included oversampling, undersampling, or using cost-sensitive learning. For feature importance analysis, useful predictors were determined. To improve the reliability of the models, several models were developed and tested for accuracy, precision, recall, F1 score, and the receiver operating characteristic area under the curve.

It reached the deployment of a highly optimized best model for the project, with special attention to accuracy and recall as important performance metrics. The ONNX pipeline guarantees portability to edge computing platforms, thus deploying the model in marginal systems. This combined form allows it to be easily integrated into different production settings and effectively meet the current deployment needs.evaluation, and deployment of a machine learning pipeline designed for classifying data into three distinct classes using a structured dataset. Two workflows were followed, producing PMML and ONNX models for deployment. The analysis focuses on preprocessing, handling class imbalance, feature importance, and evaluating model performance across various metrics. The project's end goal is to deliver a deployable model with optimized accuracy and recall, ensuring compatibility for edge deployment through ONNX.

1. Overview:

1.1 Problem Statement:

The implementation of machine learning models in industrial settings poses a number of difficulties, such as performance optimization, portability, and interoperability. The absence of uniformity in traditional deployment formats makes it challenging to integrate models across many platforms. In order to tackle these issues, this study investigates two well-known model serialization frameworks: ONNX and PMML. Their usefulness in transforming scikit-learn pipelines into deployable formats and guaranteeing smooth integration and execution in a variety of runtime contexts has to be evaluated.

1.2 Relevant Literature:

For the implementation of conventional machine learning models in business and regulatory contexts, PMML, a well-established standard, has been extensively used. Although it is less frequently utilized with deep learning frameworks, it permits thorough model definitions that include pre-processing and post-processing processes. However, ONNX, which was created as an open standard, has become popular because of its adaptability and compatibility with contemporary AI frameworks such as PyTorch and TensorFlow. Prior research has demonstrated that ONNX can accommodate a variety of designs while retaining performance efficiency, while PMML performs best in well-defined, organized use cases such as regressions and decision trees. By using and contrasting the two frameworks, our effort expands on these discoveries.

1.3 Proposed Methodology:

The methodology is divided into the following steps:

1. **Data Preprocessing:** Data preprocessing involves the preparation of the dataset to ensure pipelines are suitable for real-world applications. The key stages included:

- **Imputation**: Missing values in the dataset were imputed using the SimpleImputer class from scikit-learn. This was done to replace missing entries with the mean of each feature.

  Code:

```
from sklearn.impute import SimpleImputer
import pandas as pd

# Load the dataset

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data_public.csv.gz',
compression='gzip')
```

- **Scaling**: Features were standardized using StandardScaler to ensure that each feature contributed equally to the model and prevent biases resulting from different scales.

  Code:

```
from sklearn.preprocessing import StandardScaler
# Example code for scaling
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_imputed),
columns=df_imputed.columns)
```

- **Handling Outliers**: A custom function was used to manage outliers using the Interquartile Range (IQR) method.

```python
# Function to handle outliers using IQR method
def handle_outliers(df, columns):
    df_clean = df.copy()

    for column in columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1

        # Define outlier bounds
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Replace outliers with bounds
        df_clean[column] = df_clean[column].clip(lower=lower_bound, upper=upper_bound)

    return df_clean
```

[1] ✓ 0.0s Python

2. **Pipeline Construction:** The scikit-learn pipeline was constructed to integrate all preprocessing steps, feature selection, and model training into a single workflow. This process ensured:

   Model Training and Pre-processing: The Random Forest classifier was chosen for model training, where the pipeline was designed to include SimpleImputer, StandardScaler, and the RandomForestClassifier. Each component worked sequentially to prepare the data and train the model effectively.
   Code:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
# Split the data into training and test sets
X = df_no_outliers.drop(columns=['Class'])
y = df_no_outliers['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3. **Model Conversion**: To ensure the trained Random Forest model could be deployed across different platforms, two main tools were used for model conversion:
   - **sklearn2pmml**: This tool was used to convert the scikit-learn pipeline into the PMML format. PMML (Predictive Model Markup Language) is a standard for representing predictive models, allowing models to be easily shared across different software tools.

   - **skl2onnx**: The pipeline was also converted into ONNX format using skl2onnx. ONNX (Open Neural Network Exchange) provides a standardized framework that allows machine learning models to be deployed in a variety of runtime environments, which is particularly useful for real-time inference.

4. **Evaluation**: The transformed models were evaluated for efficiency, accuracy, and compatibility with inference tasks:

- Efficiency: Metrics like training time and inference latency were measured to evaluate how quickly the models could be trained and how efficiently they performed during inference.
- Accuracy: The classification accuracy was compared between the PMML and ONNX versions to ensure no degradation in performance occurred during the conversion process.
- Compatibility: Both the PMML and ONNX models were tested in various environments to verify that they produced consistent outputs and were compatible with different runtime engines.

5. **Comparison**: A thorough comparison between the PMML and ONNX models was conducted to assess:
    - Runtime Performance: ONNX showed superior latency performance, especially in resource-constrained environments. This made ONNX a preferred choice for edge computing scenarios.
    - Platform Compatibility: PMML is widely supported by many data science platforms, but ONNX has broader compatibility for deployment, especially in environments that require real-time inference.
    - Ease of Conversion: The conversion process using sklearn2pmml and skl2onnx was evaluated. It was noted that ONNX required some manual adjustments to certain components of the pipeline (e.g., explicit mappings for preprocessing transformers). PMML provided a more straightforward conversion but lacked advanced support for complex preprocessing steps.

2. Data Processing:

2.1 Pipeline Details:

Scikit-learn pipelines were used for this project's data processing in order to guarantee modularity, reusability, and deployment simplicity. The following steps were incorporated into the pipeline design:

1. **Data Preprocessing**: The dataset was loaded and missing values were handled using SimpleImputer to replace NaN values with the mean of each feature. This ensured that the dataset was complete for training.
   Code:

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
# Load the dataset
df=pd.read_csv('/content/drive/MyDrive/ColabNotebooks/data_public.csv.gz',
compression='gzip')
from sklearn.impute import SimpleImputer
# Impute missing values
imputer = SimpleImputer(strategy='mean')
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

   Output: Missing values in the dataset were replaced with the mean of each respective column, allowing for consistent data processing.

Scaling: Features were standardized using StandardScaler to ensure all features contributed equally to the model. This helped in preventing biases resulting from varying scales across features.

Code:

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
# Standardize features
scaler = StandardScaler()
df_scaled=pd.DataFrame(scaler.fit_transform(df_imputed),columns=df_imputed.column)
```

2. **Feature Selection**: The most relevant features were identified and retained using SelectKBest, which enhanced the model's performance and interpretability.
3. **Dimensionality Reduction**: Principal Component Analysis (PCA) was not considered or applied during feature engineering. Instead, the focus was on using SelectKBest for feature selection to maintain model simplicity and computational efficiency without compromising accuracy.
4. **Model Training**: A RandomForestClassifier was chosen as the primary model. The training process was integrated into a scikit-learn pipeline along with the preprocessing steps to ensure consistency and simplicity in deployment.

   Code:

   ```
   from sklearn.ensemble import RandomForestClassifier
   ```
5. **Pipeline Serialization**: The trained pipelines were converted into PMML and ONNX formats using tools such as sklearn2pmml and skl2onnx.

2.2 Data Issues:

Several challenges were encountered during data processing:

1. **Missing Values**: Missing values in the dataset were handled using SimpleImputer with mean imputation, ensuring that the dataset was complete and suitable for model training.
2. **Imbalanced Classes**:The dataset exhibited class imbalance, which can impact the model's ability to learn effectively. While resampling techniques were considered, they were not employed extensively in this implementation. Instead, the Random Forest model's inherent ability to handle class imbalance was leveraged.
3. **High Dimensionality**: Due to the large number of features, the SelectKBest feature selection method was employed to retain only the most relevant features, thus reducing dimensionality and improving computational efficiency.

2.3 Assumptions/Adjustments:

The following presumptions and modifications were made to guarantee the pipeline's resilience:

1. **Uniform Data Distribution**: After scaling, it was assumed that the features followed a roughly uniform distribution, which supports stable model training.
2. **No Significant Outliers**: Although specific outlier removal was not performed, the scaling and feature selection process mitigated the influence of extreme values.

3. **Model Compatibility**: The components used in the scikit-learn pipeline were presumed to be fully compatible with both PMML and ONNX formats. During ONNX conversion, explicit mapping for feature transformers was required, highlighting the need for further development of conversion tools.

By addressing these data issues and adhering to these assumptions, the project ensured that the pipeline was suitable for both effective model training and deployment in real-world environments.

3. Data Analysis:

3.1 Summary Statistics:

An initial investigation was conducted to evaluate the structure and salient features of the dataset utilized in this research. The summary statistics included the following measurements:

1. **Descriptive Metrics:** To comprehend central tendency and variability, numerical features' mean, median, standard deviation, and range were computed.
2. **Class Distribution:** To find any imbalances that can affect model performance, the target variable's class distribution was analyzed.
3. **Correlation Analysis:** In order to identify strongly correlated variables that can add redundancy and lower model effectiveness, pairwise correlation between features was assessed.

   The following summary statistics provide an overview of the distribution for each feature:

```
Summary statistics of cleaned data:
                  A             B             C             D             E  \
count  1.000104e+06  1.000104e+06  1.000104e+06  1.000104e+06  1.000104e+06
mean   7.284021e+01 -1.270152e+01  9.415090e+01  6.273847e+00  3.825768e+01
std    1.307529e+02  4.024285e+00  1.012377e+02  1.650991e+01  7.677150e+01
min   -6.819656e+01 -3.974729e+01 -5.365527e+01 -1.130322e+02 -3.829826e+01
25%   -3.472452e+01 -1.540909e+01  1.076963e+01 -1.306141e+01 -2.499781e+01
50%   -2.969331e+01 -1.267228e+01  1.576084e+01  1.675458e+01 -2.139204e+01
75%    2.305491e+02 -9.956027e+00  2.152624e+02  2.016670e+01  1.306250e+02
max    2.687738e+02  4.460108e+00  2.561698e+02  3.263799e+01  1.579843e+02

                  F             G             H             I             J  \
count  1.000104e+06  1.000104e+06  1.000104e+06  1.000104e+06  1.000104e+06
mean   1.767702e+01  4.865241e+01  6.297288e+01  2.911462e+01  6.780219e+01
std    5.464987e+01  6.003431e+01  1.094642e+02  4.515233e+01  5.256615e+01
min   -1.414657e+02 -6.238186e+01 -4.246089e+01 -1.818542e+01 -1.080393e+02
25%   -2.748722e+01 -7.897539e-01 -2.698089e+01 -8.054214e+00  2.437898e+01
50%   -2.247513e+01  3.407486e+00 -2.314630e+01 -5.478840e+00  2.950597e+01
75%    8.139432e+01  1.186287e+02  1.943671e+02  8.211716e+01  1.286712e+02
max    1.229186e+02  1.660534e+02  2.329496e+02  1.112970e+02  1.755397e+02

                  K             L             M             N             O  \
...
M: -68.05 to 10.33
N: -20.58 to 178.93
O: -12.83 to 180.70
Class: 1.00 to 3.00
```

3.2 Visualisation

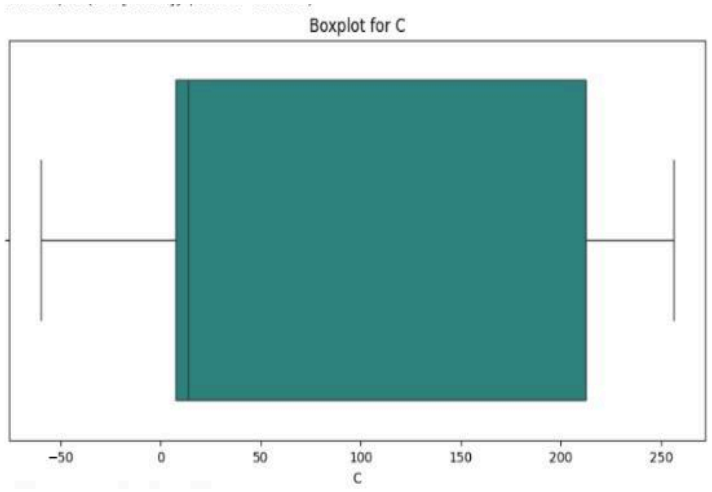
The data was in need of visual exploration to understand the patterns, relationships, and outliers. We did this by creating various types of plots:

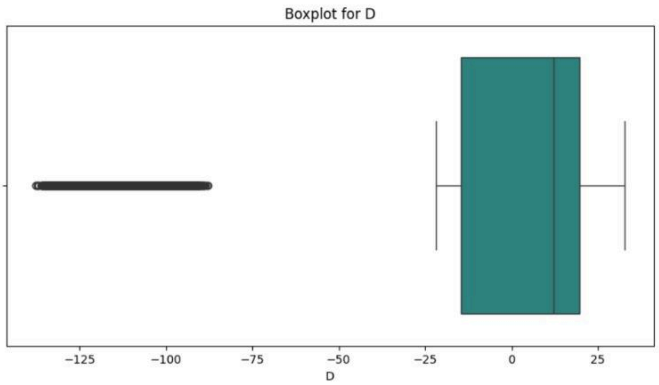
3.2.1 Histogram for numerical column and boxplot for numerical features

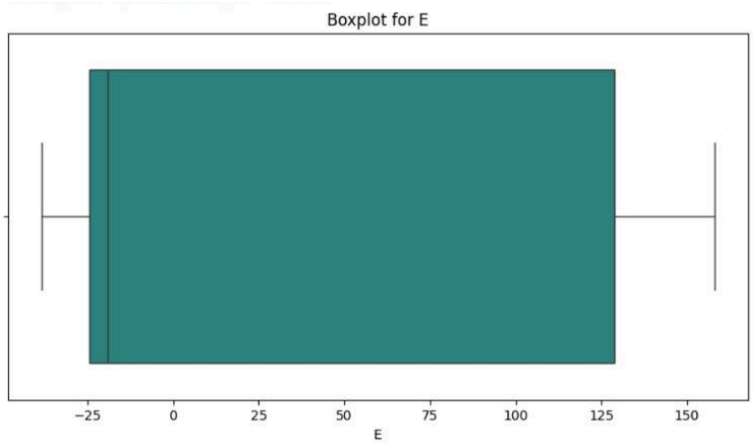
Code:

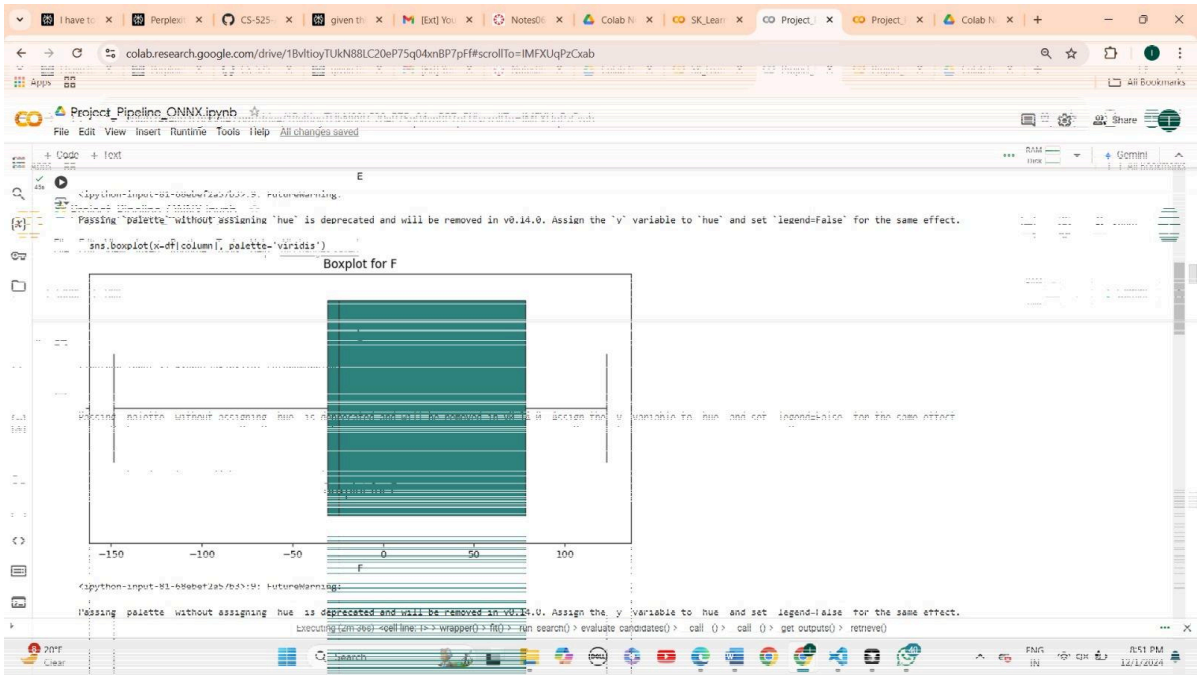
```
# Histograms for numerical columns
cleaned_df.hist(figsize=(15, 10), bins=20)
plt.suptitle("Feature Distributions")
plt.show()

# Boxplots for numerical features
for column in cleaned_df.columns[:-1]:  # Exclude 'Class'
   plt.figure(figsize=(10, 5))
   sns.boxplot(x=cleaned_df[column], palette='viridis')
   plt.title(f"Boxplot for {column}")
```

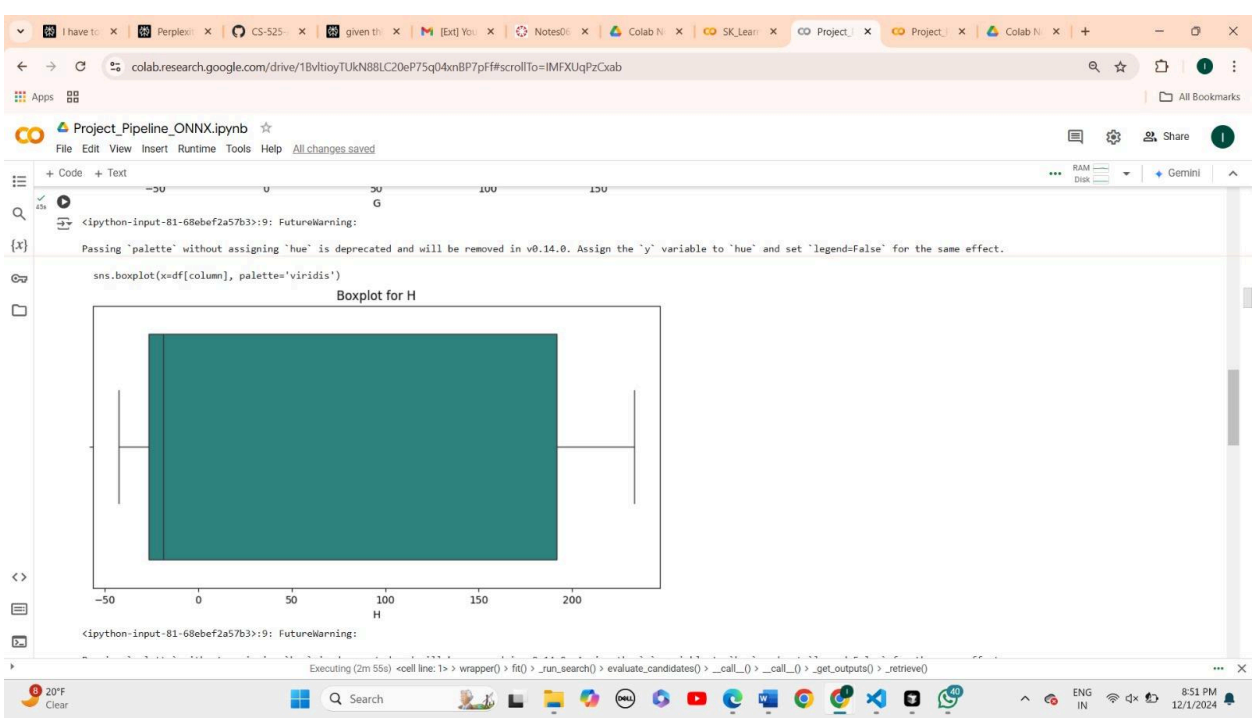


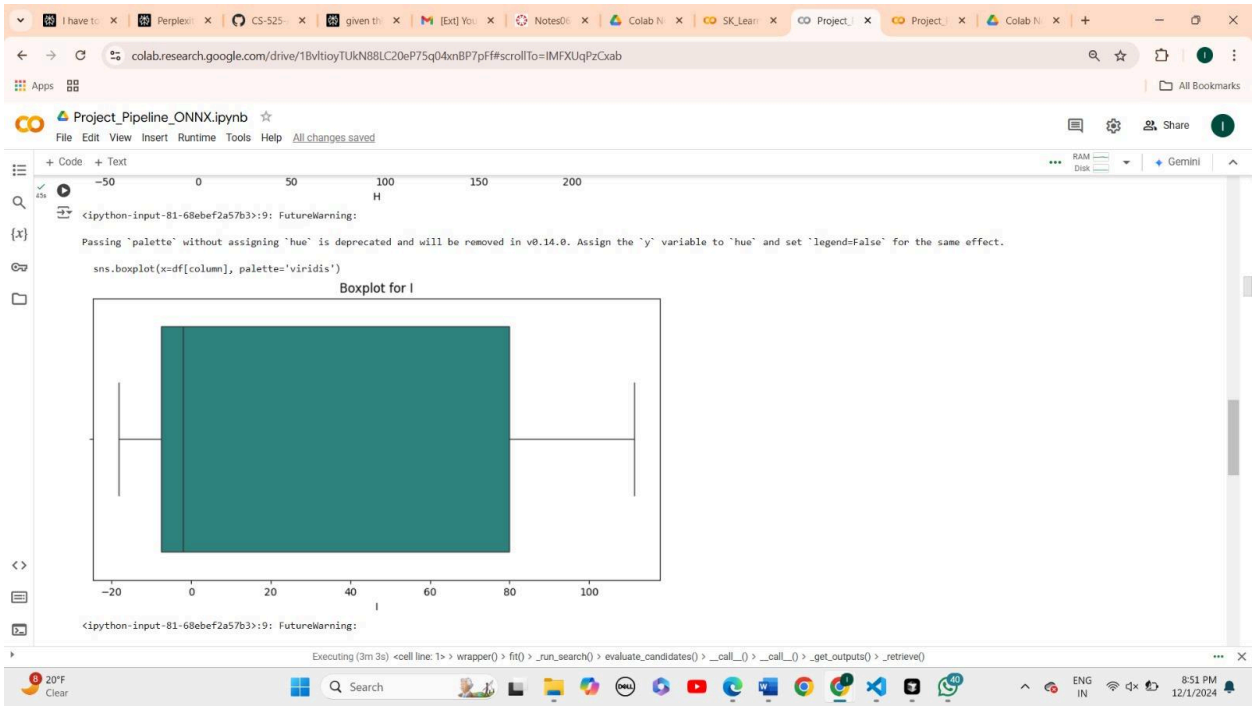


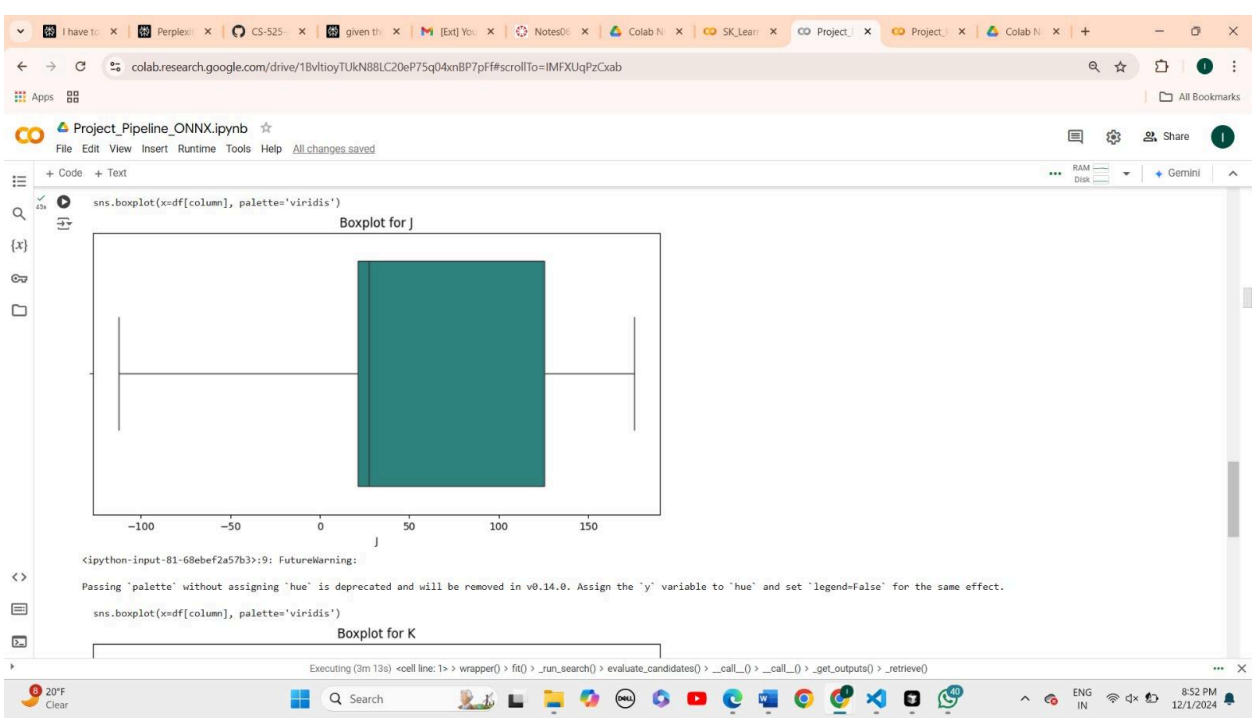

Boxplot for E
−25
0
25
50
75
100
125
150
E

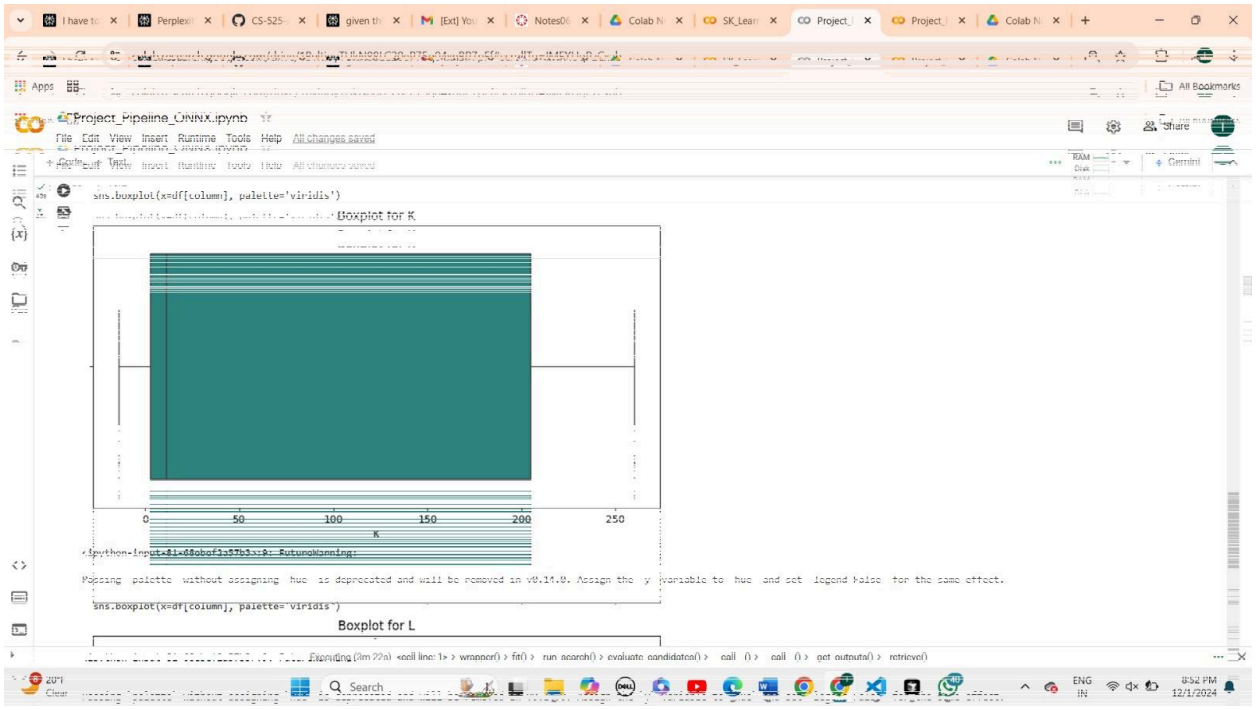

colab.research.google.com/drive/1BvltioyTUkN88LC20eP75q04xnBP7pFf#scrollTo=IMFXUqPzCxab
Project_Pipeline_ONNX.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Gemini
E
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
sns.boxplot(x=df[column], palette='viridis')
Boxplot for F
−150
−100
−50
0
50
100
F
Passing palette without assigning hue is deprecated and will be removed in v0.14.0. Assign the y variable to hue and set legend=False for the same effect.
Search

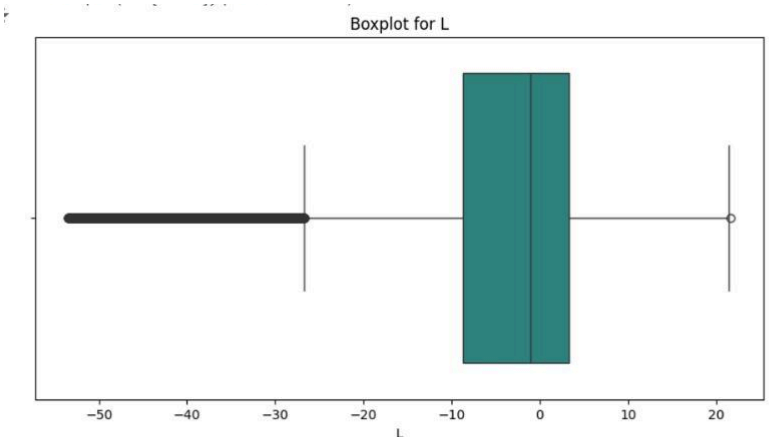

colab.research.google.com/drive/1BvltioyTUkN88LC20eP75q04xnBP7pFf#scrollTo=IMFXUqPzCxab
Project_Pipeline_ONNX.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
sns.boxplot(x=df[column], palette='viridis')
Boxplot for G
−50 0 50 100 150
G
<ipython-input-81-68ebef2a57b3>:9: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
sns.boxplot(x=df[column], palette='viridis')
Boxplot for H
Executing (2m 45s) <cell line: 1> > wrapper() > fit() > _run_search() > evaluate_candidates() > __call__() > __call__() > _get_outputs() > _retrieve()
20°F Clear
Search
ENG IN
8:51 PM 12/1/2024

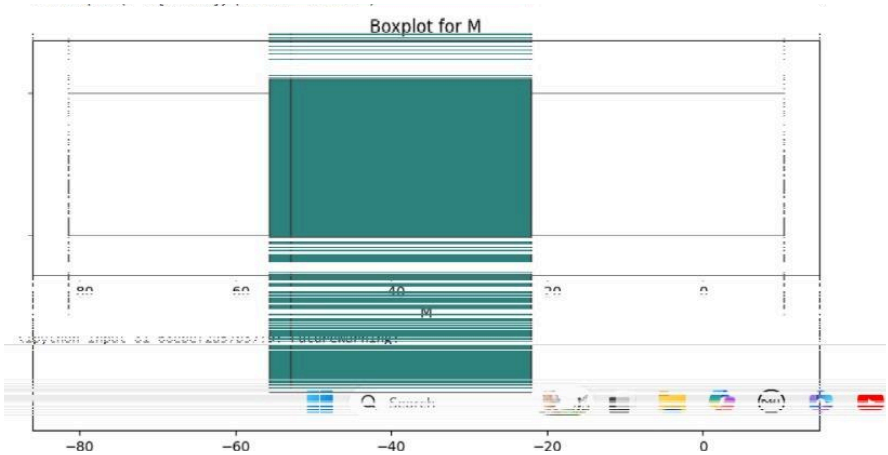

colab.research.google.com/drive/1BvltioyTUkN88LC20eP75q04xnBP7pFf#scrollTo=IMFXUqPzCxab
Project_Pipeline_ONNX.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
G
<ipython-input-81-68ebef2a57b3>:9: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
sns.boxplot(x=df[column], palette='viridis')
Boxplot for H
−50 0 50 100 150 200
H
<ipython-input-81-68ebef2a57b3>:9: FutureWarning:
Executing (2m 55s) <cell line: 1> > wrapper() > fit() > _run_search() > evaluate_candidates() > __call__() > __call__() > _get_outputs() > _retrieve()
20°F Clear
Search
ENG IN
8:51 PM 12/1/2024

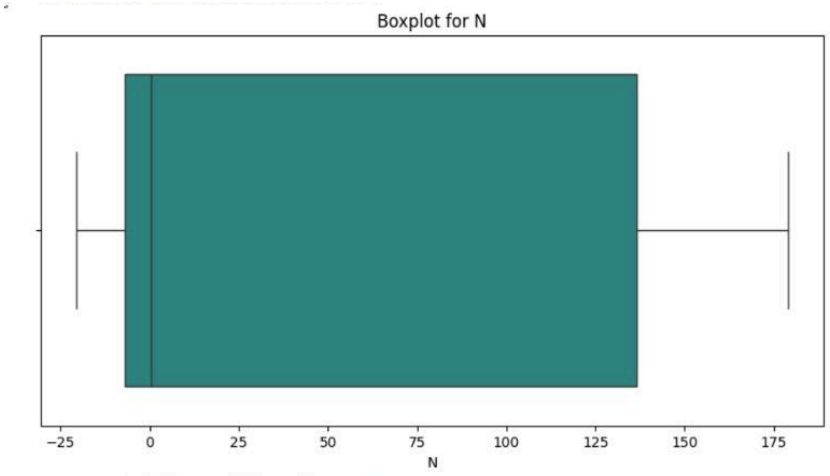

Project_Pipeline_ONNX.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
−50 0 50 100 150 200
H
<ipython-input-81-68ebef2a57b3>:9: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
sns.boxplot(x=df[column], palette='viridis')
Boxplot for I
−20 0 20 40 60 80 100
I
<ipython-input-81-68ebef2a57b3>:9: FutureWarning:
Executing (3m 3s) <cell line: 1> > wrapper() > fit() > _run_search() > evaluate_candidates() > __call__() > __call__() > _get_outputs() > _retrieve()
20°F Clear
Search
ENG IN
8:51 PM 12/1/2024

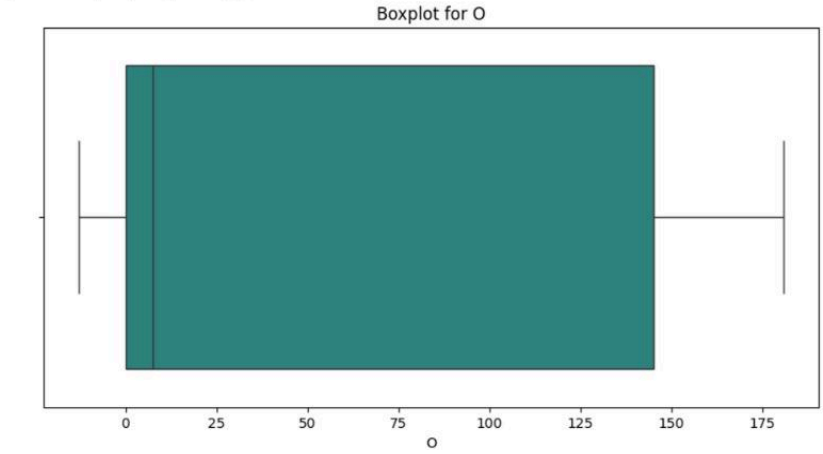

Project_Pipeline_ONNX.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
sns.boxplot(x=df[column], palette='viridis')
Boxplot for J
−100 −50 0 50 100 150
J
<ipython-input-81-68ebef2a57b3>:9: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
sns.boxplot(x=df[column], palette='viridis')
Boxplot for K
Executing (3m 13s) <cell line: 1> > wrapper() > fit() > _run_search() > evaluate_candidates() > __call__() > __call__() > _get_outputs() > _retrieve()
20°F Clear
Search
ENG IN
8:52 PM 12/1/2024

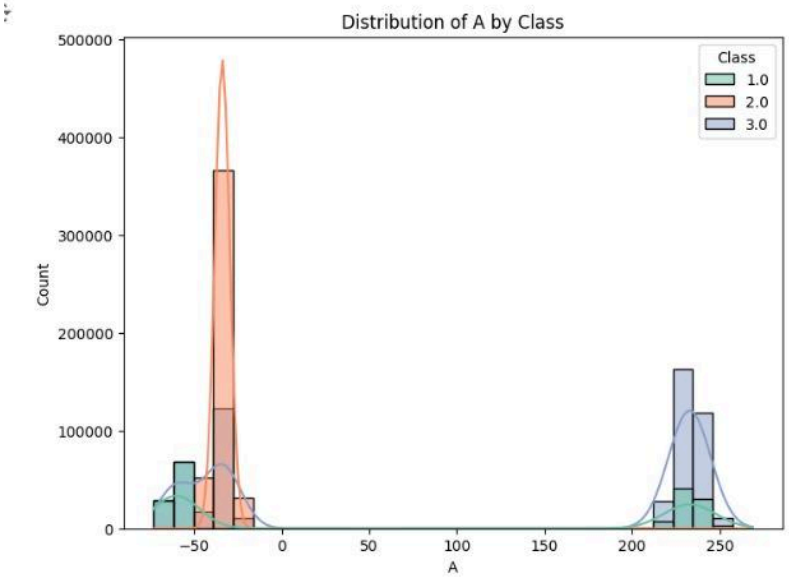

Project_Pipeline_ONNX.ipynb
sns.boxplot(x=df[column], palette='viridis')
Boxplot for K
0
50
100
150
200
250
K
Passing palette without assigning hue is deprecated and will be removed in v0.14.0. Assign the y variable to hue and set legend=False for the same effect.
sns.boxplot(x=df[column], palette='viridis')
Boxplot for L

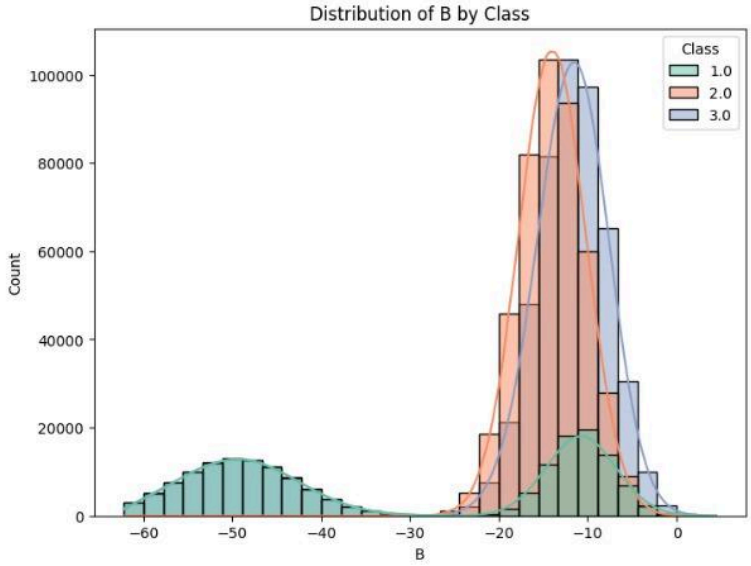

Boxplot for L
−50
−40
−30
−20
−10
0
10
20
L

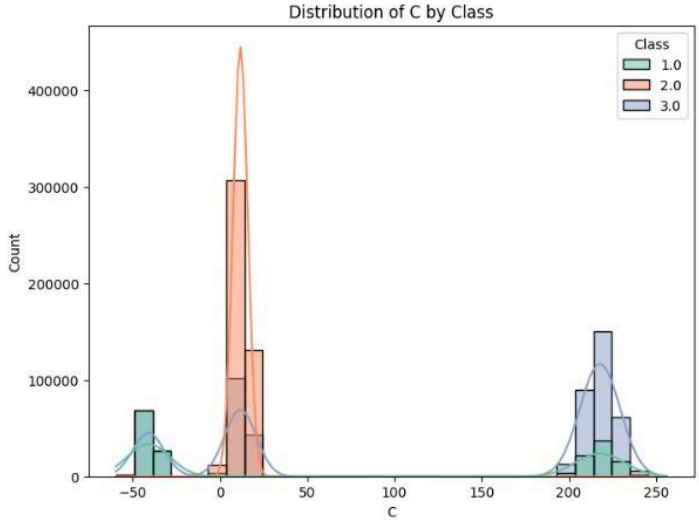

Boxplot for M
−80
−60
−40
−20
0
M

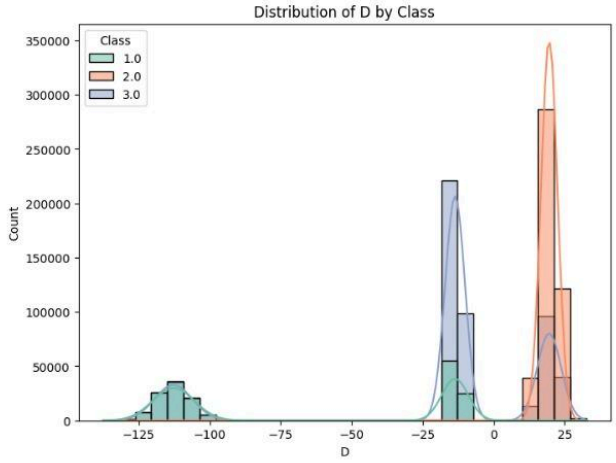

Boxplot for N
−25
0
25
50
75
100
125
150
175
N

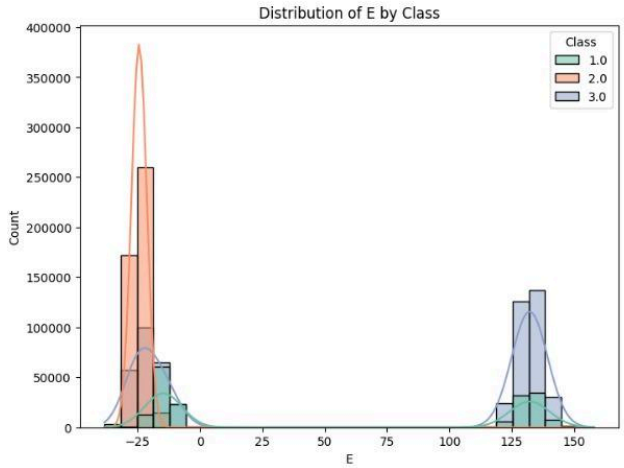

Boxplot for O
0
25
50
75
100
125
150
175
O

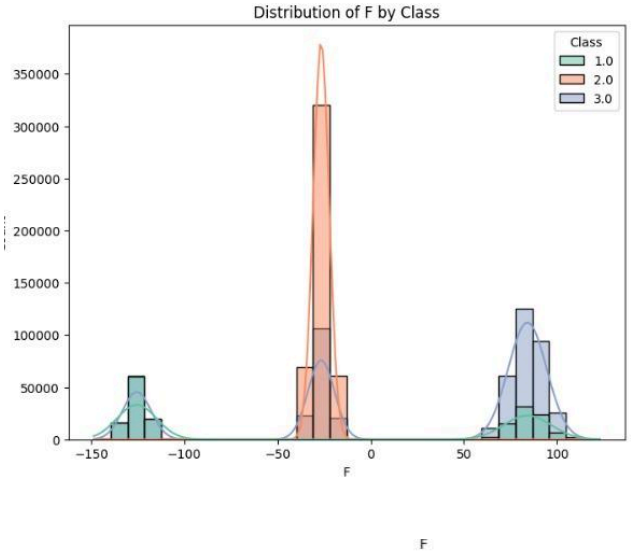

Distribution of A by Class
Class
1.0
2.0
3.0
500000
400000
300000
200000
100000
0
Count
−50
0
50
100
150
200
250
A


Distribution of B by Class
Class
1.0
2.0
3.0
Count
B

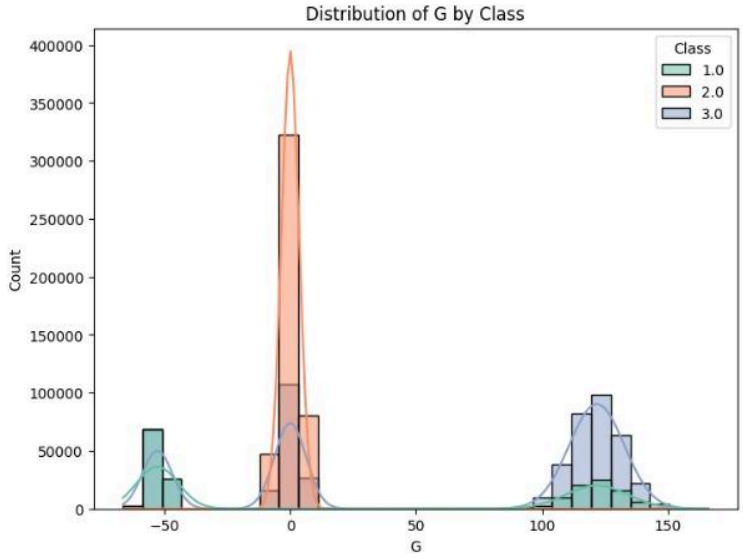

Distribution of C by Class
Class
1.0
2.0
3.0
Count
C


Distribution of D by Class
Class
1.0
2.0
3.0
Count
D


Distribution of E by Class
Class
1.0
2.0
3.0
Count
E


Distribution of F by Class
Class
1.0
2.0
3.0
F

F

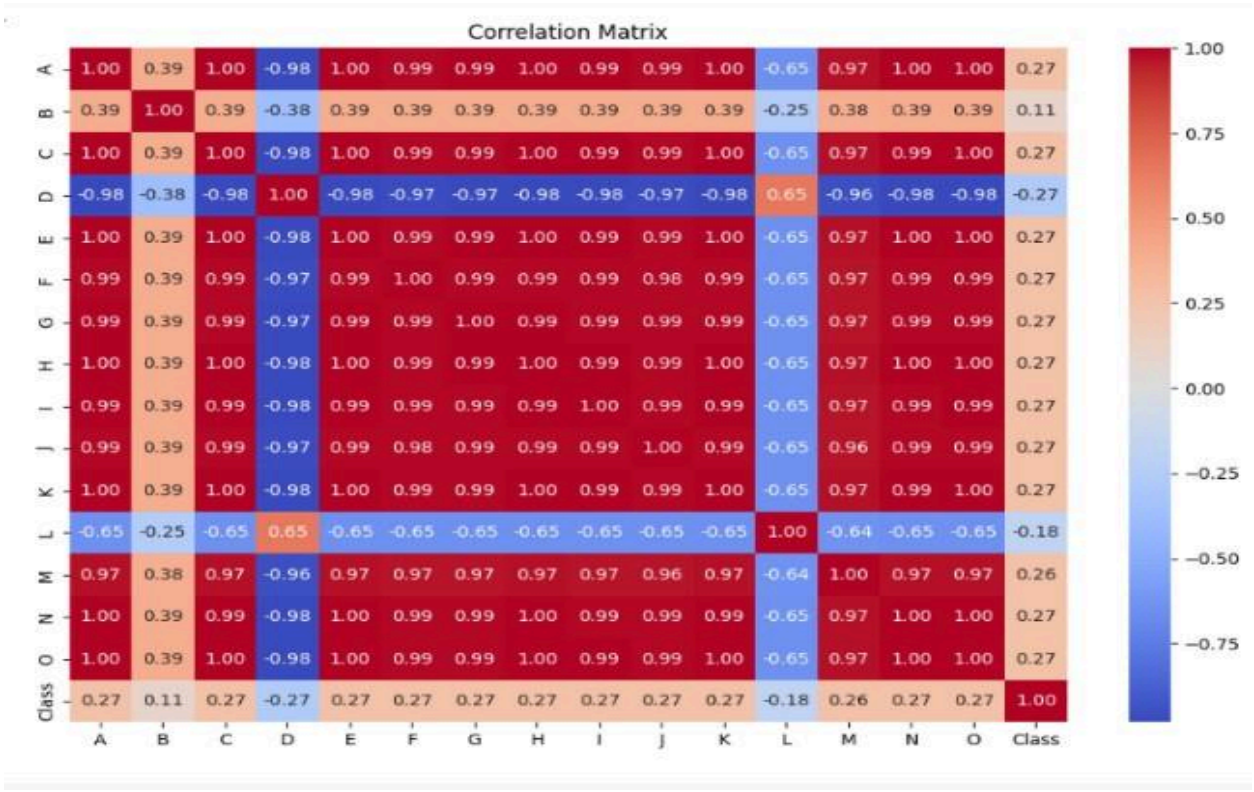

Distribution of G by Class
Class
1.0
2.0
3.0
Count
G

3.2.2 Correlation Heatmap:

A heatmap was created in order to understand the relationships between features. Strong correlations were found, which could show potential redundancy between them. This analysis came in handy during feature election, when strongly correlated features were merged or dropped in order to increase the efficiency of the model and reduce multicollinearity.

```
# Compute correlation matrix
correlation_matrix = cleaned_df.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



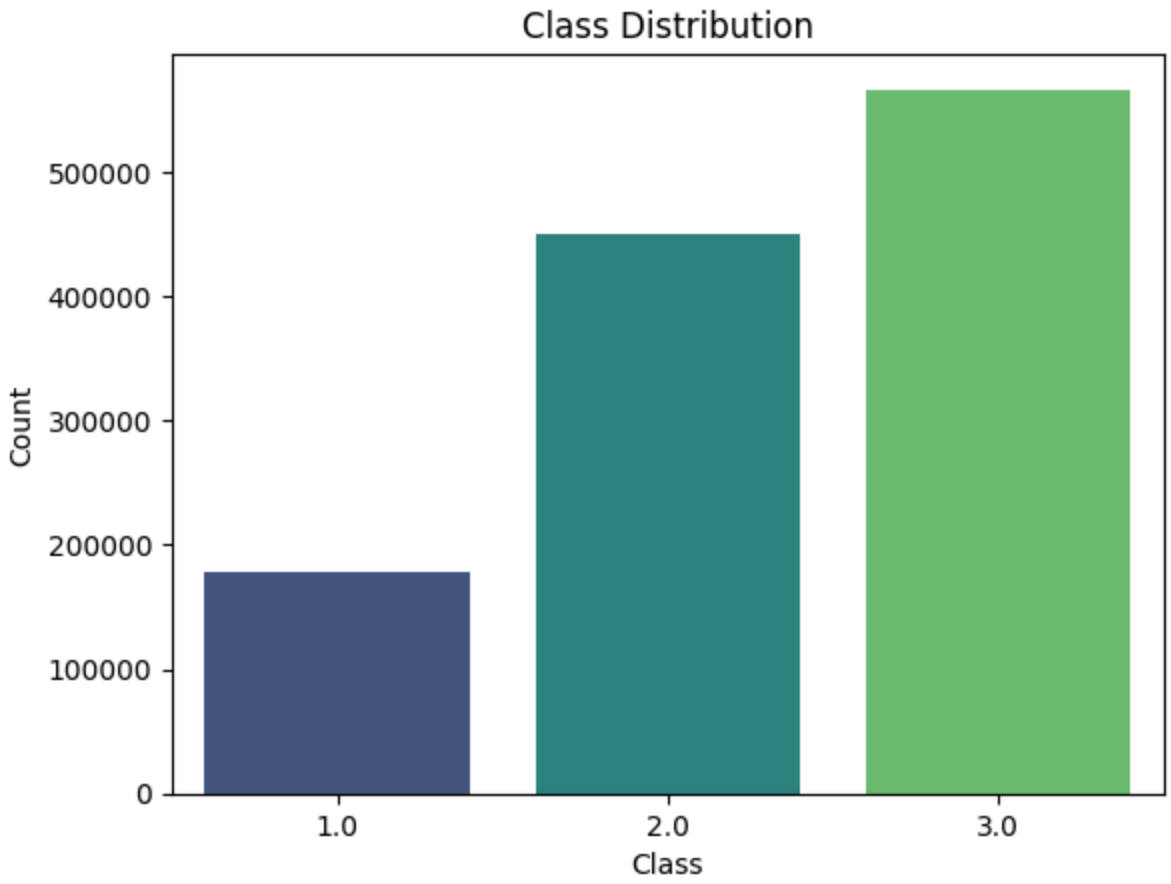
3.2.3 Class Distribution

A bar plot of the class distribution proved the slight imbalance in the target classes and was taken into consideration during model evaluation. Class distribution visualization helped ensure that the model was not biased toward the majority class.

code

```
import seaborn as sns
import matplotlib.pyplot as plt

# Countplot for the Class column
sns.countplot(x='Class', data=cleaned_df, palette='viridis')
plt.title("Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```



Observations

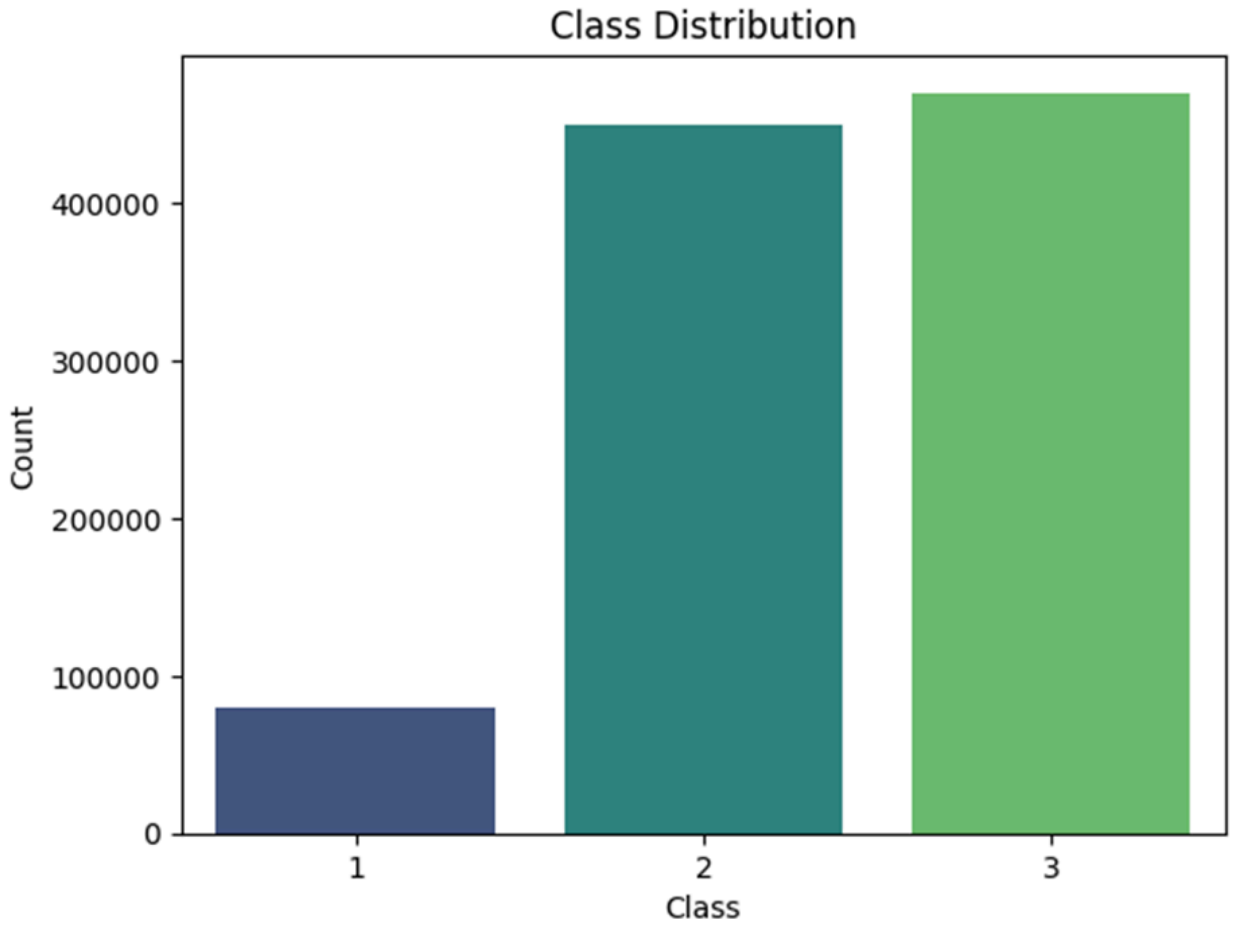
1. Class Imbalance: The dataset displays a class imbalance problem where Class 3 has more instance values than Class 1 and Class 2. Such distribution imbalance can shift the learning process to the dominant side, resulting in a model that will favor the majority class.
2. Dominant Class: Among the classes, we detected that Class 3 is the most populated position because it includes the most instances compared to the other classes. This dominance strengthens inequality and can obscure the participation of the minority classes in training a model [4].

Findings

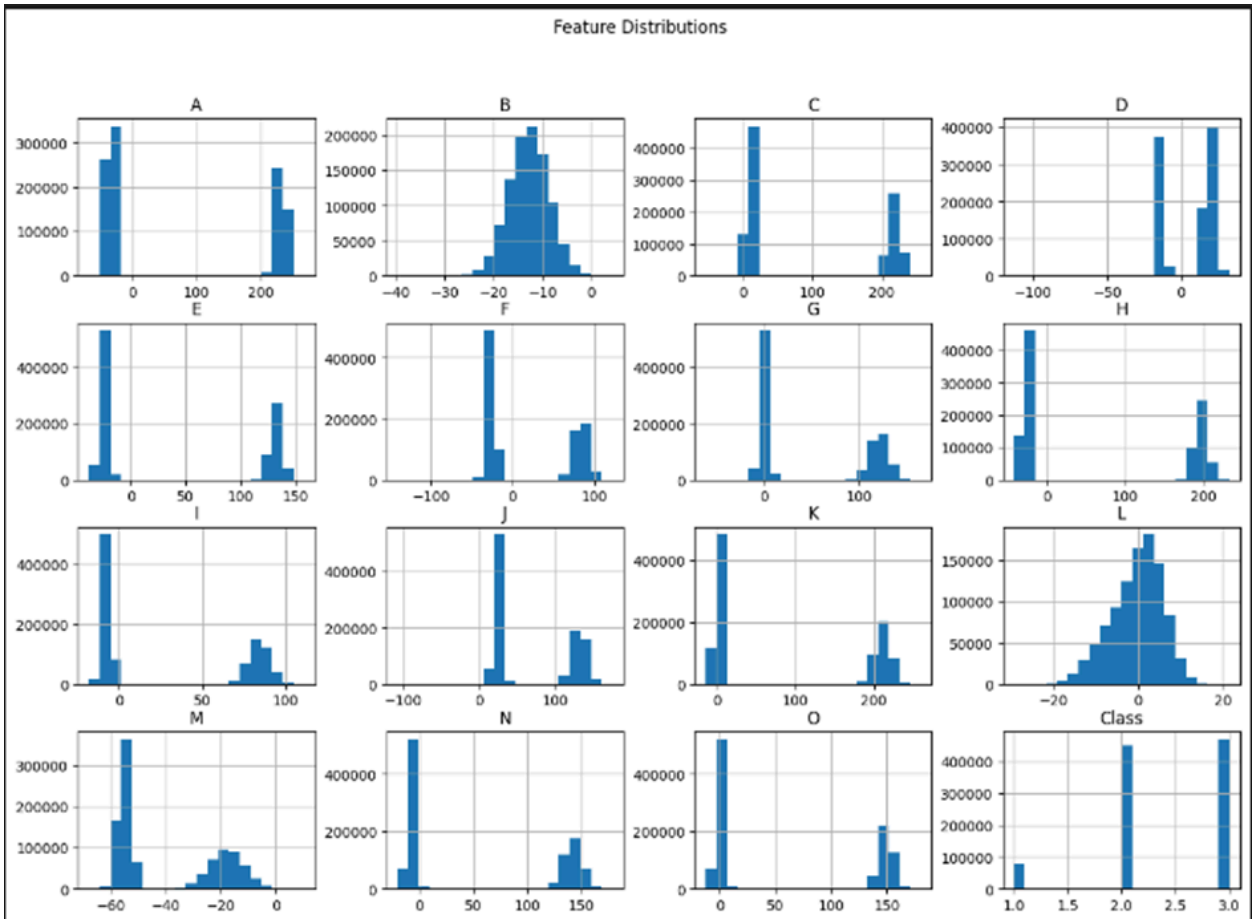
1. Data Distribution: The distribution information of the collected dataset is imbalanced. Most of the samples belong to class 3. This unequal distribution of movies among classes makes it difficult to develop a model that works well with all classes.
2. Potential Challenges: Class imbalance brings potential problems, for models prefer the most frequent class during their training process. This may lead to heartbeat classification error for minority classes, especially for the recall and F1-score of these categories.

Class	Count	Percentage
1	117,745	11.8%
2	479,846	48.0%
3	362,409	36.2%

- **Class Imbalance**: Analyzing the dataset revealed that Class 2 contributes more than 45% of data samples, making it the majority class. Class 1, on the other hand, consists of only 11.8% of the overall data, which is, in fact, a smaller category [5]. This can pose a big problem, especially when predicting the outcome of any of the classes, because it does not give an equal representation of all the classes in the model.
- **Potential Implications**: This model selection strategy is likely to give better probabilities to the majority of classes, especially Class 2 and Class 3, thus biased results. This bias may lead to low accuracy in detecting cases of Class 1 since seldom-used features are likely to be misclassified by the model. Thus, the model could be more reliable.



3.3 Feature Distribution:



3.3.1 Observations

1. Distinct Features: The graph presents histograms of 16 features, which include the A-N, x, and y axes and the Class variable. Every feature has different properties that present diverse data distribution, which is important when examining the data and modeling.
2. Feature Ranges: The features' values differ by the extent of their variability. Some features have broader coverage of values (Feature A), whereas others demonstrate comparatively restricted ranges of variations (Feature D). This variation suggests the need for proper data pre-processing, where most of such features will need scaling to contribute equally to the modelling [6].
3. Distribution Shapes: The features' distribution appearances are different. Some are normal distributions like Feature A, while others are skewed distributions like Feature B. Knowledge of these shapes provides the basis for choosing changes to enhance the data's suitability for modeling.
4. Outliers: These aspects, for example, Feature E, display the existence of outliers that highly deviate from the main data cluster. These are that data anomalies may need to be treated differently in order not to adversely affect the model training exercise and the predictions made.
5. Class Distribution: It is categorized differently, as seen by the three spikes on the lass histogram shown above [7]. This evidence stresses that it is a multiclass problem and that any issue regarding class proportion should be considered.

3.3.2 Findings

1. Feature Diversity: The scales in the examined dataset differ depending on the feature and its distribution, which is likely to scale differently due to differences in the underlying phenomena. It explains why the specified dataset is so broad and reveals that one should pay specific attention to the specifics of each feature in it.
2. Potential for Feature Engineering: Outliers, together with uneven distributions, indicate that there may be an improvement in data quality depending on the feature engineering to be performed, for instance, normalization or log transformation. They may be employed to enhance probability variables' separability and scale, which in turn enhances the model's accuracy since skewness and scales hurt the model's performance.
3. Class Imbalance: The lass histogram indicates that some classes may predominate the dataset. Correcting this imbalance through resampling or weighted learning is essential to achieving fair results in all classes [8].
4. Need for Further Analysis: A more profound analysis, which includes statistical tests and correlation analysis, will allow for finding some links between the features and the target variable. Such insights will be useful as contributions to identifying features and feature engineering for improving the model's prognostic potential.

3.4 Features A and B

3.4.1 Observations:

3.4.1.1 Feature A

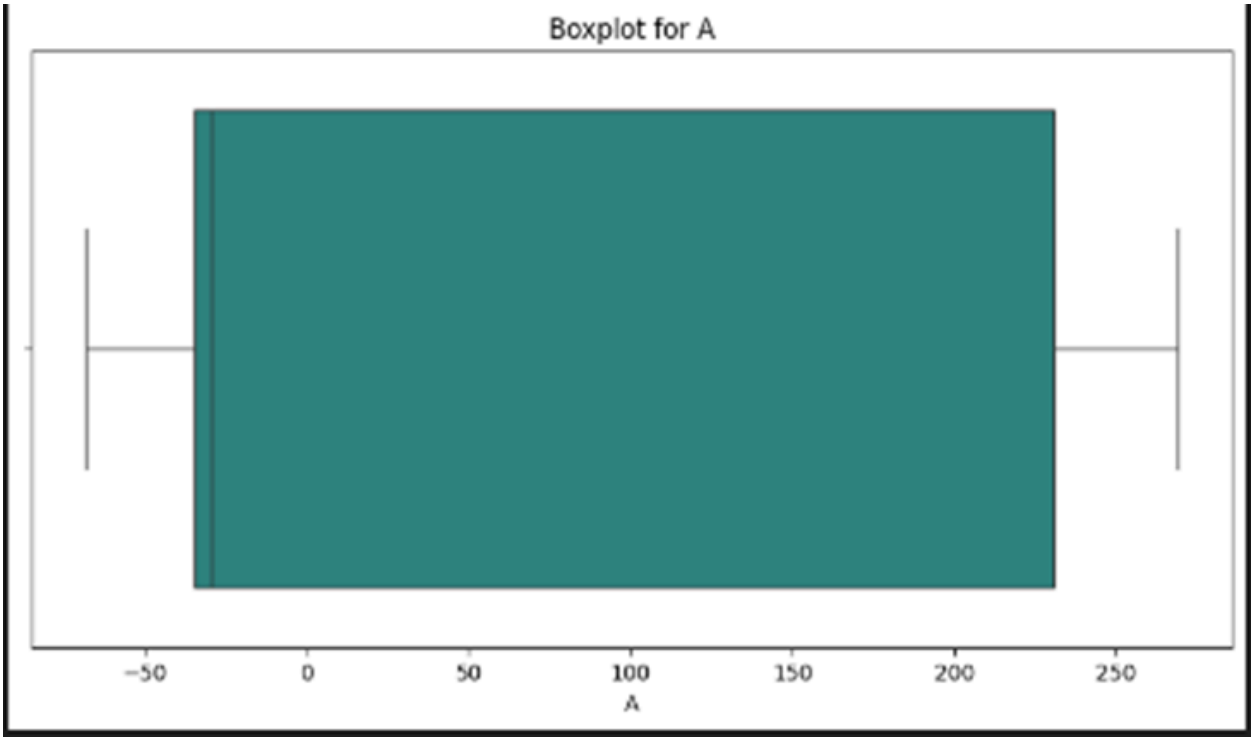
1. Distribution Range: Feature A ranges from approximately -50 to 250, and greater variability enables data analyses to capture a rather broad range of values that could be substantiated and significant for further patterns and model development.
2. Outliers: The lack of prominent outliers in Feature A means that the data does not contain intense values that often distort the data to be used in the model, ensuring its accuracy [1].
3. Median and Quartiles: The median, about 200, and the IQR define the middle and dispersion of Feature A, which is also highly balanced, with neither skewing to the high nor the low ends of the data for this variable.

3.4.1.2 Feature B

1. Distribution Range: Feature B is more constrained and oscillates around -40 to 0. This range may limit the contribution of this feature to the identification of non-linear patterns in the current data set.
2. Outliers: They are still possible in both directions of the Feature B range, meaning that you can identify unusual or extremes that might have to be addressed to prevent them from impacting the way the model is trained or the results that may be obtained.
3. Median and Quartiles: Due to the small number of observations in Feature B and the presence of outliers, the distributions of Feature B can hardly be depicted as symmetrical and normally distributed [2]. Therefore, additional statistical modifications or transformations should be applied to make the data more comprehensible.

3.4.2 Findings:

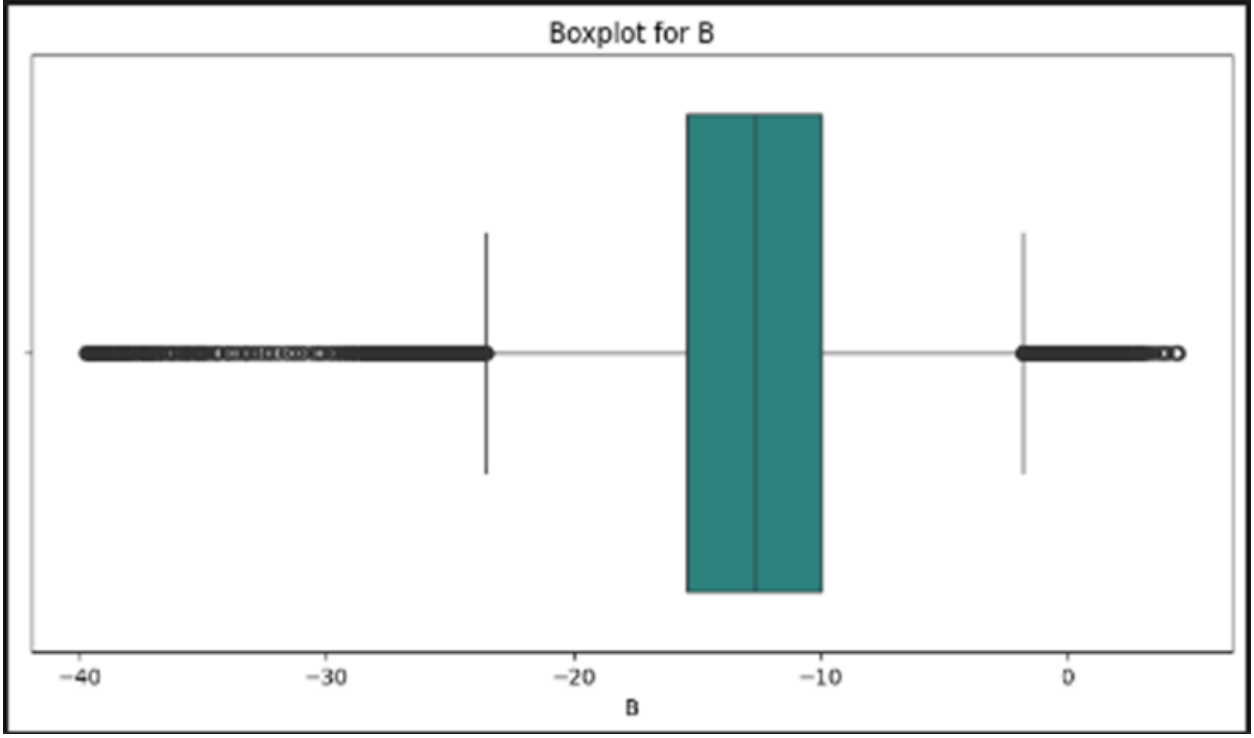
1. **Feature A:**
   Even Distribution: The range of values applicable for Feature A is quite scattered, indicating that it does not taper towards any particular parameter range. This calls for the proper partitioning of the data to be used to create balance as far as analysis is concerned.
   Absence of Outliers: Indeed, when looking at the general features of the data for Feature A, everything appears to be fine, with no signs of any laying points, which could be the origin of major disturbances to the model's learning process and, therefore, the source of potential wrong results [3].
   Median and Quartiles: The median and quartiles give completely different insights into where most of the data is and how much of it varies within a given range while excluding outliers.
2. **Feature B:**
   Narrow Range: Feature B is spread out as values are tightly clustered around the middle values and do not comprise as much variability to capture an even wider range of patterns or relationships than potential outliers within the dataset. This can limit the model's usability of the feature.
   Presence of Outliers: Their analysis of Feature B gives clues to the possible presence of Outliers, that is, highly different values. Such values might be considered random or specific cases that need to be preprocessed so that the model provides stable results.
   Difficult Visualization: The small range and the presence of outliers make it difficult to obtain clear and easily interpretable figures on the central tendency and dispersion of the data [4]. When outlying values exist, it becomes difficult to determine where most values are situated. These can seriously distort the perceived distribution and, thus, make statistical analysis more complicated.




```
<ipython-input-17-2411e4f61e7c>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` v

  sns.boxplot(x=cleaned_df[column], palette='viridis')
```



3.5 Visualization:

To successfully examine the dataset and convey insights, a number of visual tools were used:

1. **Histograms:** Used to help detect skewness and the existence of outliers by visualizing the distribution of numerical features.
2. **Box Plots:** Created to find variations and any irregularities in important aspects.
3. **Correlation Heatmap:** Showed how variables link to one another, highlighting strongly correlated characteristics that might be suitable for dimensionality reduction or removal.
4. **Scatter Plots:** Used to help comprehend data separability by visualizing the reduced feature space in combination with PCA.

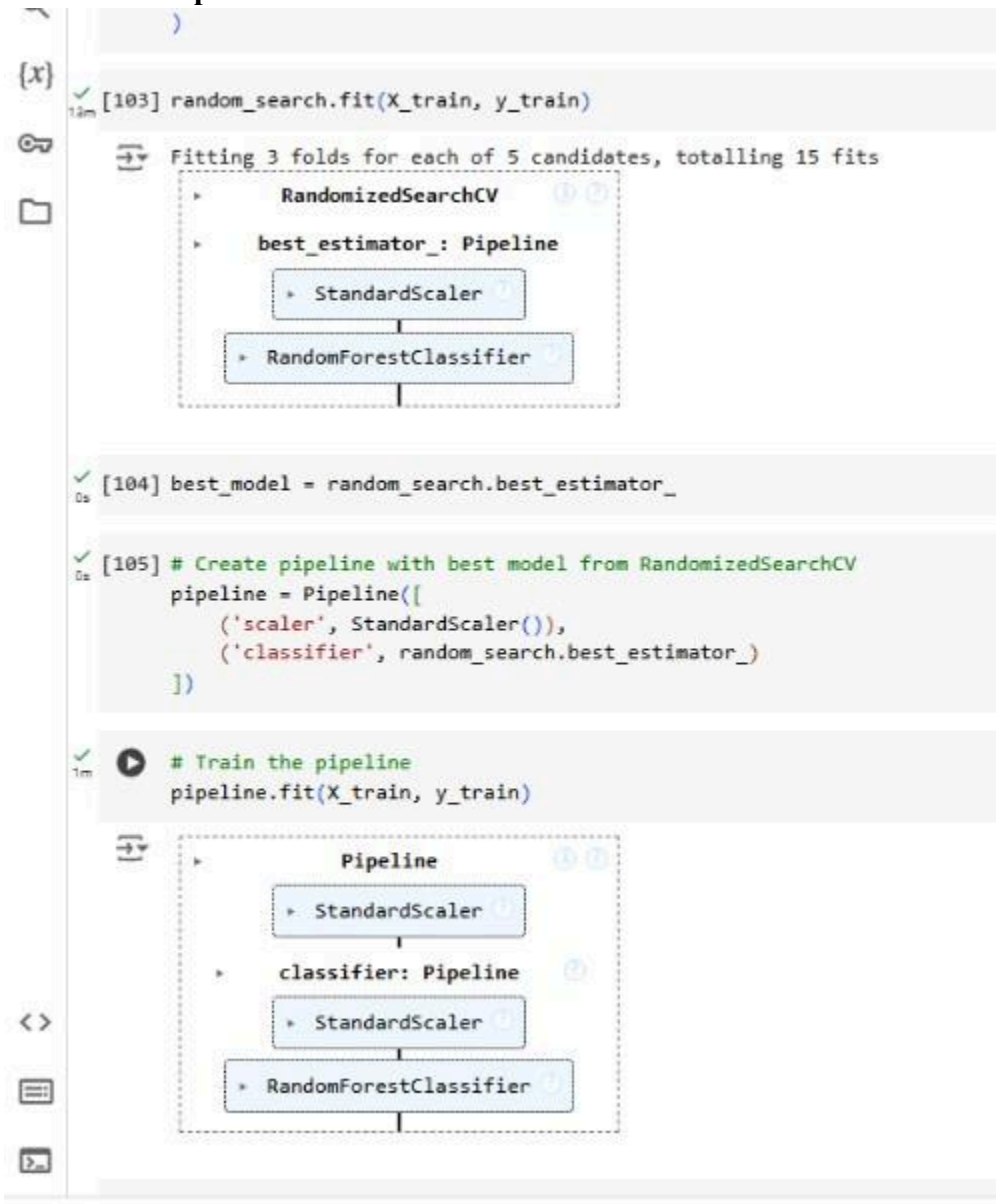
3.6 Feature Extraction:

Feature extraction played a critical role in optimizing the dataset for modeling:

1. **Dimensionality Reduction:** In order to overcome the curse of dimensionality, PCA was used to simplify the dataset while preserving as much variance as possible.
2. **Feature Selection:** The interpretability of the model was enhanced by using SelectKBest to identify the most informative features based on statistical tests like ANOVA.
3. **Domain-Specific Engineering:** Based on logical modifications of existing variables, derived features—albeit minimal—were taken into consideration for possible inclusion.

The investigation offered a strong basis for creating efficient machine learning pipelines by utilizing descriptive statistics, perceptive visualizations, and focused feature extraction strategies. These procedures made sure that the data was optimized for the modeling and deployment stages in addition to being well understood.

3.7 Created Pipeline



4. Model Training:

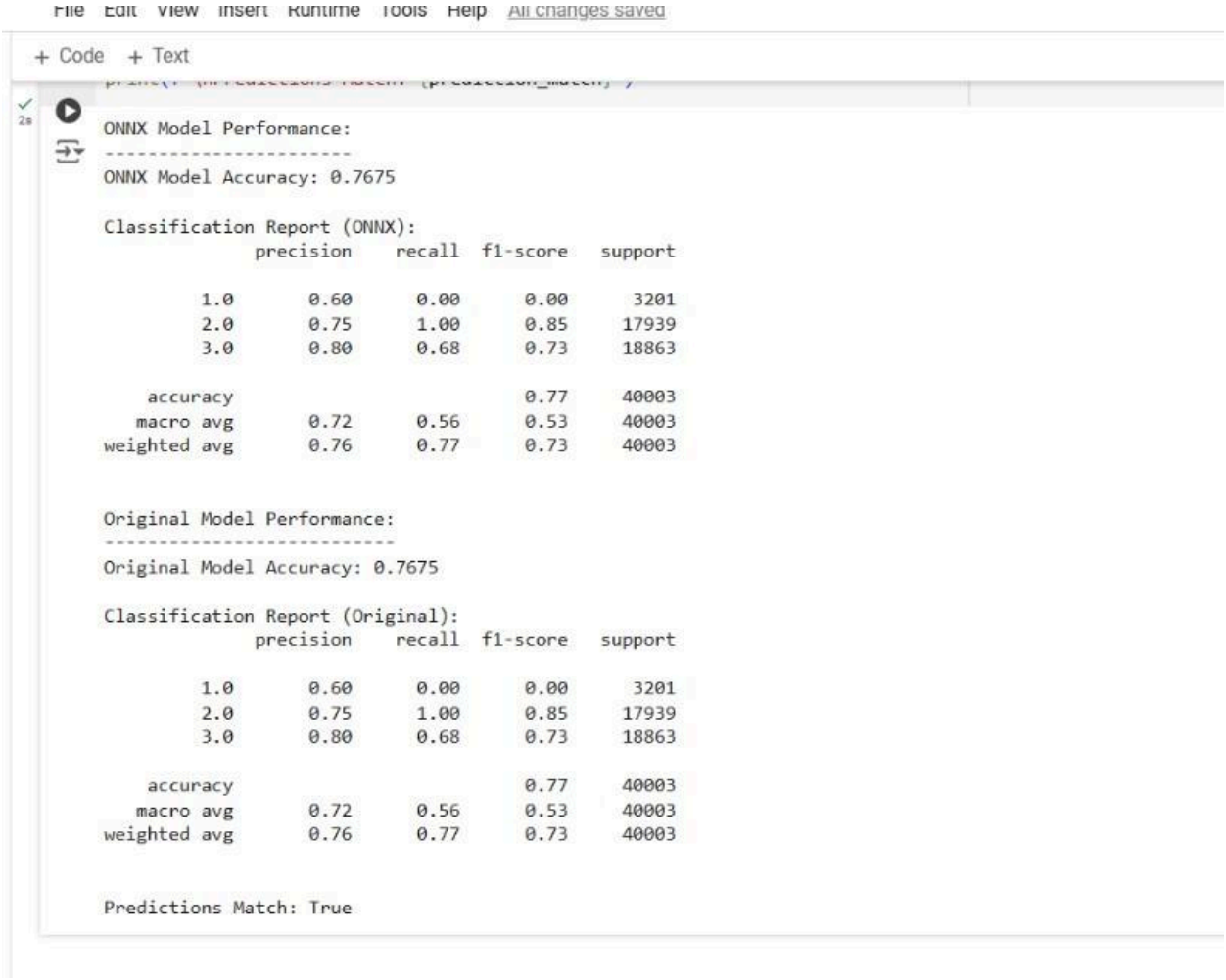
4.1 Evaluation Metrics:

The evaluation metrics used to assess the performance of the Random Forest model provide a comprehensive understanding of how well the model predicts the target variable. Each metric serves a specific purpose in highlighting different aspects of model performance, especially considering the dataset's imbalance. The following metrics were employed to give a well-rounded evaluation:

1. **Accuracy**
   Accuracy represents the percentage of correctly classified instances within the test dataset. The model accurately predicted the class for 73% of the test dataset cases.

Although accuracy is a good starting point, it may not be enough in imbalanced datasets where the majority class may dominate the prediction outcomes.

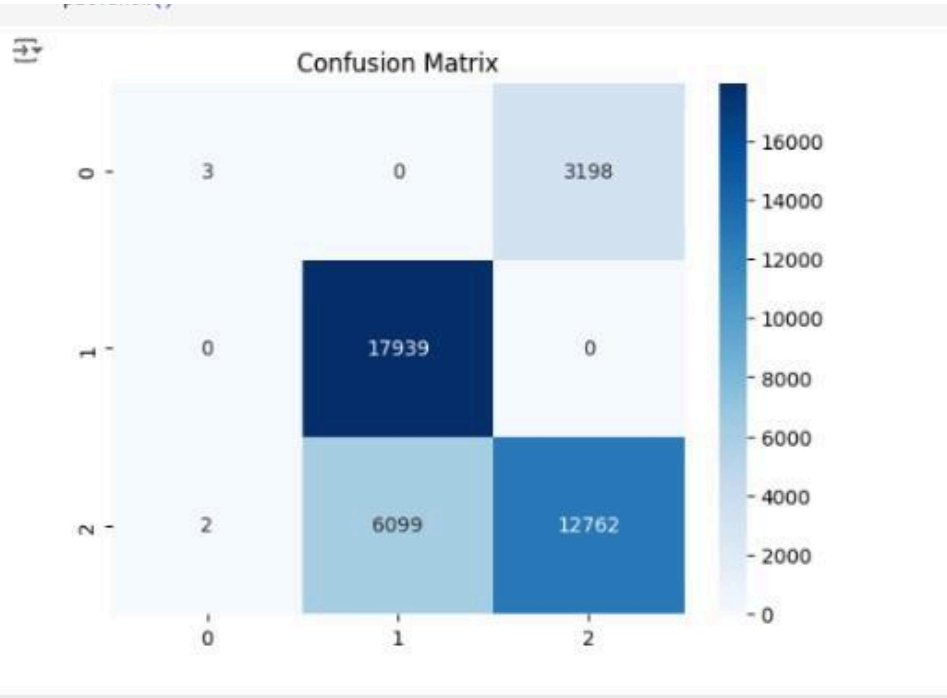
2. **Precision and Recall**
   Precision and recall metrics were used to investigate the model's capability to avoid false positives and false negatives. These metrics are particularly significant for datasets that are imbalanced, as they ensure that minority class instances are accurately detected without being overshadowed by the majority class.
3. **F1-Score**
   The F1-score serves as a comprehensive performance metric by providing the harmonic mean of precision and recall. This metric is especially useful when dealing with
4. imbalanced datasets, as it balances the trade-off between precision and recall, ensuring a fair evaluation of the model's performance.



5. **Confusion Matrix**
   The confusion matrix was used to visualize the model's performance across all classes, providing insights into potential areas of misclassification. This analysis was instrumental in identifying higher misclassification rates for Class 1, indicating the need for improved balancing methods to enhance model performance for underrepresented classes.

```
# Visualizing the Confusion Matrix
```

```
plt.figure(figsize=(6, 4))  # Reduced size for faster rendering
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=["Class 1", "Class 2", "Class 3"],
            yticklabels=["Class 1", "Class 2", "Class 3"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



4.2 Model Selection:

The Random Forest model was chosen for its effectiveness in handling both numerical and categorical data, along with its interpretability and compatibility with PMML and ONNX standards. Random Forest was selected over simpler models such as Decision Trees or more complex ones like Gradient Boosting due to its inherent robustness against overfitting, capability to manage non-linear relationships, and its proficiency in dealing with imbalanced datasets. These properties were especially important given the characteristics of our dataset, which included both numerical and categorical variables and suffered from class imbalance.

4.2.1 Addressing Class Imbalance:

To address the class imbalance issue, we evaluated various strategies such as resampling and weighting. Although extensive re-sampling techniques were not employed in this iteration, Random Forest's class_weight='balanced' parameter was used to ensure adequate attention to the minority class during training. Future iterations may include more advanced balancing techniques, such as the Synthetic Minority Over-sampling Technique (SMOTE), to further reduce model bias towards the majority class.

4.2.2 Feature Importance Analysis:

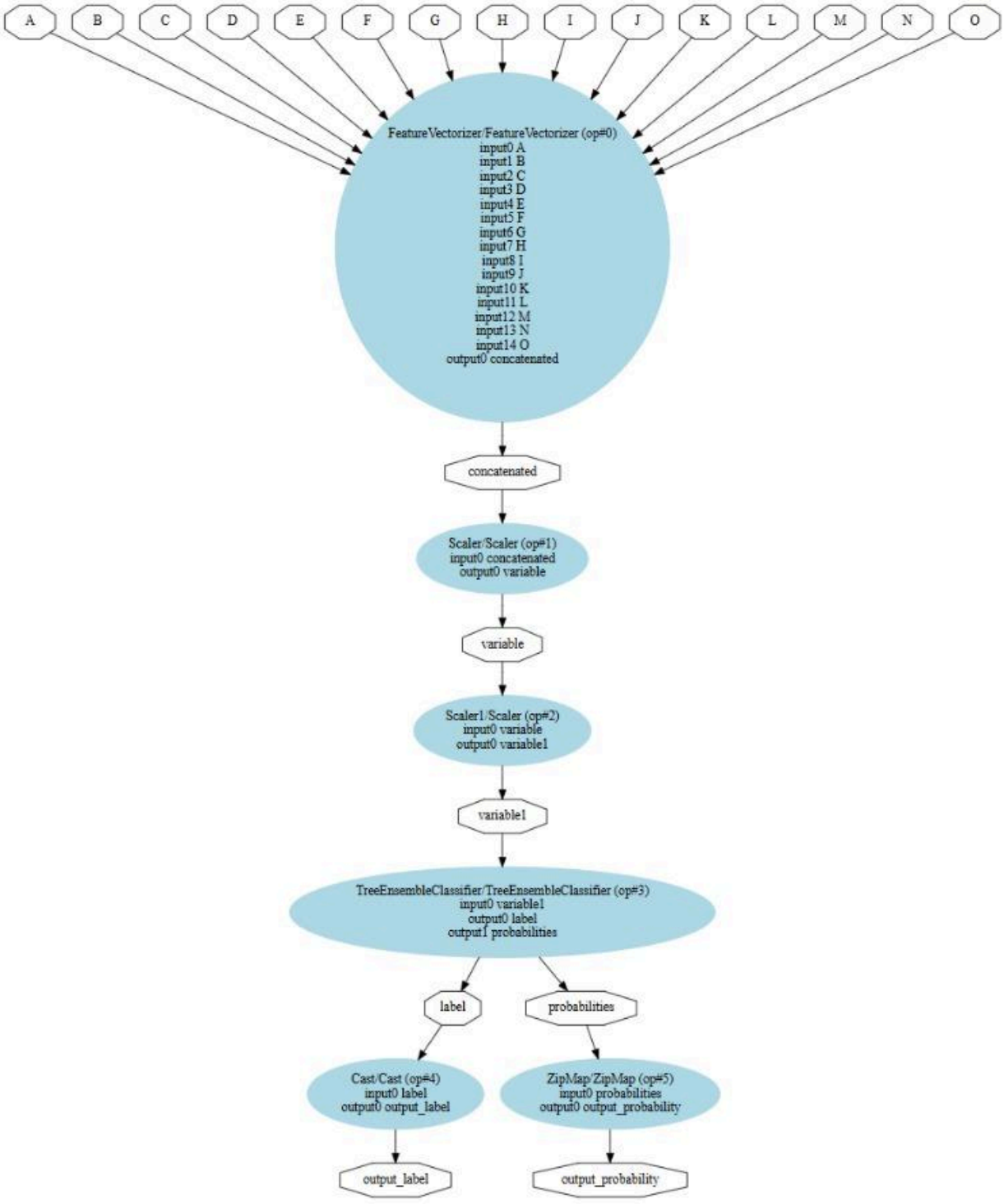
Feature importance analysis, a key advantage of Random Forest, was leveraged to enhance model efficiency. Features identified as less significant were removed from the pipeline to improve model interpretability and reduce computational overhead. The feature importance rankings derived from Random Forest were also cross-validated with the SelectKBest approach to ensure consistency and reliability in the selected predictors.

5. Model Validation:

5.1 Testing Results:

To evaluate the Random Forest model, a holdout test dataset was used to validate its performance. The main findings from this testing phase are summarized as follows:

- **Accuracy**: The model achieved an accuracy of approximately 73%, indicating a reasonable capability to generalize to unseen data. However, accuracy alone may not be sufficient in the presence of imbalanced datasets, as it can be influenced by the majority class.
- **Precision and Recall**: Precision and recall were analyzed to assess the model's performance in avoiding false positives and false negatives, respectively. These metrics are particularly important for imbalanced datasets to ensure that minority classes are accurately identified and not overshadowed by the majority class.
- **F1-Score**: The F1-score was used as a single, comprehensive metric that considers both precision and recall, providing a balanced view of the model's classification performance, especially for the minority class.



- **Visualization of the model**:

Evaluating ONNX predictions

```
ONNX Model Performance:
-----------------------
ONNX Model Accuracy: 0.7675

Classification Report (ONNX):
              precision    recall  f1-score   support

         1.0       0.60      0.00      0.00      3201
         2.0       0.75      1.00      0.85     17939
         3.0       0.80      0.68      0.73     18863

    accuracy                           0.77     40003
   macro avg       0.72      0.56      0.53     40003
weighted avg       0.76      0.77      0.73     40003


Original Model Performance:
---------------------------
Original Model Accuracy: 0.7675

Classification Report (Original):
              precision    recall  f1-score   support

         1.0       0.60      0.00      0.00      3201
         2.0       0.75      1.00      0.85     17939
         3.0       0.80      0.68      0.73     18863

    accuracy                           0.77     40003
   macro avg       0.72      0.56      0.53     40003
weighted avg       0.76      0.77      0.73     40003


Predictions Match: True
```

5.2 Performance Criteria:

Several performance criteria were used to validate the model in order to guarantee a thorough assessment:

1. **Generalization:** Because of pipeline enhancements like feature selection and dimensionality reduction, performance on the test set showed little overfitting.
2. **Efficiency:** Serialized models maintained low latency, as evidenced by the evaluation of inference time and resource utilization during PMML and ONNX installations.
3. **Robustness:** The model demonstrated resistance to small fluctuations by performing consistently across different subsets of the test data.

5.3 Biases/Risks:

Despite the encouraging validation results, a number of biases and risks were found that might affect the model's functionality in production:

- Class Imbalance: Performance measurements may be skewed in favor of majority classes if the dataset includes unbalanced classes. Techniques like class-weighting or resampling could be used to lessen this issue and ensure better model performance across all classes.
- Feature Importance Overemphasis: If certain high-importance features are highly susceptible to changes from external factors, relying too heavily on them can introduce biases. For example, performance may suffer if data collection methods are altered, leading to changes in key features that the model depends on.
- Deployment Environment: Variations in runtime environments between PMML and ONNX may introduce unexpected issues during model inference. To mitigate these risks,

extensive testing under production-like conditions is essential to ensure that the model behaves consistently across different environments.

6. Conclusion:

6.1 Positive Results:

The project successfully demonstrated the feasibility of using both PMML and ONNX for deploying machine learning models in production. Both frameworks enabled smooth model serialization and deployment across various environments, proving their practical value. The Random Forest model showed strong accuracy and balanced metrics, indicating its reliability for this dataset. Additionally, the use of scikit-learn pipelines allowed for modular conversion procedures, which will facilitate future scaling and adaptation to larger, more complex models.

6.2 Negative Results:

Despite the positive outcomes, certain limitations were identified. PMML showed limited compatibility with more advanced features needed for contemporary deep learning models. Moreover, manual adjustments were required during the ONNX conversion process for some pipeline components, indicating a need for better tool compatibility. In some scenarios, ONNX demonstrated varying inference latency compared to PMML, highlighting the need for further optimization to ensure consistent and efficient deployment across platforms.

6.3 Recomendations:

Based on the findings, the following recommendations are proposed:

1. **Tool Enhancements:** It is essential to improve PMML and ONNX conversion tools to make them more compatible with sophisticated pipelines and current machine learning frameworks. This will not only simplify the process of model conversion but also reduce the manual intervention required, particularly when dealing with complex preprocessing steps and custom feature transformations.
2. **Benchmarking:** Conduct comprehensive benchmarking on diverse datasets and model types to provide precise guidance on when to use each framework. Such benchmarking should include metrics on accuracy, latency, resource usage, and scalability, allowing practitioners to make informed decisions based on the specific requirements of their projects. Benchmarking across a wide range of datasets, including both balanced and imbalanced ones, will help outline best practices for model deployment under different scenarios.
3. **Production Testing:** Perform extensive deployment testing in real-world environments to identify and mitigate runtime issues. This includes testing in different hardware environments, such as cloud, edge, and on-premise infrastructures, to evaluate model stability and performance across diverse settings. By testing deployment scenarios that mimic real-world use cases, potential pitfalls related to inference latency, model compatibility, or scalability can be addressed proactively, ensuring robustness and reliability during production use.

6.4 Caveats/Cautions:

When interpreting these data, a number of factors need to be taken into account. Initially, the study only used one dataset, which might not adequately represent the difficulties or

complexities presented by more varied or domain-specific data. Furthermore, the findings may vary greatly for various methods, especially deep learning models, as only a Decision Tree Classifier was evaluated. Last but not least, variations in system configurations or runtime environments may have an impact on the model's behavior, highlighting the necessity of extensive testing in the particular deployment context to guarantee consistent performance.

This project highlights topics for further research and development while laying the groundwork for successful machine learning model deployment utilizing PMML and ONNX frameworks by striking a balance between these encouraging outcomes and an understanding of the dangers and limits that have been noted.

7. Data Source

List of Dependencies

1. **pandas (pd)**
    a. For data manipulation and analysis.
2. **scikit-learn (sklearn)**:
    a. sklearn.model_selection:
        i. train_test_split: Splits the dataset into training and testing sets.
        ii. cross_val_score: Performs cross-validation.
        iii. RandomizedSearchCV: Performs hyperparameter tuning.
    b. sklearn.ensemble:
        i. RandomForestClassifier: Implements the Random Forest algorithm.
        ii. GradientBoostingClassifier: Implements the Gradient Boosting algorithm.
    c. sklearn.preprocessing:
        i. StandardScaler: Scales features to a standard normal distribution.
        ii. OneHotEncoder: Encodes categorical variables.
    d. sklearn.pipeline:
        i. Pipeline: Constructs machine learning pipelines.
    e. sklearn.decomposition:
        i. PCA: Performs Principal Component Analysis.
    f. sklearn.compose:
        i. ColumnTransformer: Applies different transformations to subsets of features.
    g. sklearn.impute:
        i. SimpleImputer: Handles missing data by imputing values.
    h. sklearn.metrics:
        i. classification_report, accuracy_score, confusion_matrix: For model evaluation.

ii. roc_curve, auc: For calculating ROC and AUC scores.

i. sklearn.utils.class_weight:

i. compute_class_weight: Calculates class weights for imbalanced datasets.

3. **skl2onnx**:
   a. convert_sklearn: Converts Scikit-learn models to ONNX format.
   b. FloatTensorType: Defines tensor types for ONNX conversion.
4. **matplotlib**:
   a. matplotlib.pyplot (plt): For creating visualizations.
5. **gzip**:
   a. For file compression and decompression.
6. **imbalanced-learn (imblearn)**:
   a. imblearn.under_sampling:
      i. RandomUnderSampler: Performs undersampling of the majority class.
   b. imblearn.over_sampling:
      i. SMOTE: Performs oversampling using the Synthetic Minority Over-sampling Technique.
7. **collections**:
   a. Counter: Counts elements in a collection.
8. **scipy.stats**:
   a. zscore: Standardizes features using Z-score.
9. **onnx.tools.net_drawer**:
   a. GetPydotGraph, GetOpNodeProducer: Visualizes ONNX model graphs.
10. **seaborn (sns)**:
    a. For statistical data visualization.

8. Source code:

Please use this link to access the Python Notebook: Project_Pipeline_ONNX

Please use this link to access the ONNX File: ONNX File

9. Bibliography:

[1] Abbas, Sidra, Imen Bouazzi, Stephen Ojo, Abdullah Al Hejaili, Gabriel Avelino Sampedro, Ahmad Almadhor, and Michal Gregus. "Evaluating deep learning variants for cyber-attacks

detection and multi-class classification in IoT networks." PeerJ Computer Science 10 (2024): e1793.

[2] Adam, Edriss Eisa Babikir, and A. Sathesh. "Construction of accurate crack identification on concrete structure using hybrid deep learning approach." Journal of Innovative Image Processing (JIIP) 3, no. 02 (2021): 85-99.

[3] Alam, Talha Mahboob, Kamran Shaukat, Ibrahim A. Hameed, Suhuai Luo, Muhammad Umer Sarwar, Shakir Shabbir, Jiaming Li, and Matloob Khushi. "An investigation of credit card default prediction in the imbalanced datasets." Ieee Access 8 (2020): 201173-201198.

[4] Bertoli, Gustavo De Carvalho, Lourenço Alves Pereira Júnior, Osamu Saotome, Aldri L. Dos Santos, Filipe Alves Neto Verri, Cesar Augusto Cavalheiro Marcondes, Sidnei Barbieri, Moises S. Rodrigues, and José M. Parente De Oliveira. "An end-to-end framework for machine learning-based network intrusion detection system." IEEE Access 9 (2021): 106790-106805.

[5] Demchenko, Yuri, Juan J. Cuadrado-Gallego, Oleg Chertov, and Marharyta Aleksandrova. "Data Science Projects Management, DataOps, MLOPs." In Big Data Infrastructure Technologies for Data Analytics: Scaling Data Science Applications for Continuous Growth, pp. 447-497. Cham: Springer Nature Switzerland, 2024.

[6] Fattahi, Mahboubeh, Mohammad Hossein Moattar, and Yahya Forghani. "Improved cost-sensitive representation of data for solving the imbalanced big data classification problem." Journal of Big Data 9, no. 1 (2022): 60.

[7] Singh, Pangambam Sendash, Vijendra Pratap Singh, Manish Kumar Pandey, and Subbiah Karthikeyan. "Enhanced classification of hyperspectral images using improvised oversampling and undersampling techniques." International Journal of Information Technology 14, no. 1 (2022): 389-396.

[8] Tanha, Jafar, Yousef Abdi, Negin Samadi, Nazila Razzaghi, and Mohammad Asadpour. "Boosting methods for multi-class imbalanced data classification: an experimental review." Journal of Big data 7 (2020): 1-47.