DETECTION OF FRAUD IN ONLINE PAYMENTS USING MACHINE LEARNING

INTRODUCTION:

As the world is growing faster, with this the frauds and fraudsters are increasing very rapidly. By hacking the accounts or while doing some transactions through cards. In this project I have worked with a dataset where I analyzed in building a model capable in recognizing the dataset accurately of the activities of fraudulent. The steps we are going to follow is that by importing necessary libraries and loading them into the dataset and followed by Exploratory Data Analysis (EDA) for better understanding the characteristics of the data and in recognizing the potential issues such as missing values and skewed distributions.

Subsequently, we will preprocess the data by encoding categorical variables, normalizing numerical features, and handling any missing values. Feature engineering will then be employed to create new relevant features and select the most informative ones. The dataset will split into training and testing sets, enabling us to train and evaluate various machine learning models, including Logistic regression, Random Forest, and XGBoost. Model performance will be assessed using appropriate metrics, such as accuracy, precision, recall, and F1-Score, and visualized trough confusion matrices. Finally, we will be making the predictions on the test data and extract valuable insights into the patterns of fraudulent transactions.

DATA COLLECTION AND PREPROCESSING

The dataset we utilized is sourced from the synthetic dataset simulating financial transactions and this is first we are reading the dataset using the Pandas. It is crucial in understanding the source and context of the data, as it can influence the choice of preprocessing and modeling techniques. The dataset contains various numerical and categorical features representing various aspects of financial transactions.

Also, upon initial inspection, it is essential in checking for missing values which can impact on model performance. We are going to use imputation techniques on this dataset. In case the percentage of the missing values are low, we can do the mean and median imputation for numerical columns and mode imputations for categorical columns. In case missing vales are substantial. More advanced imputation methods like predictive imputations or creating a “missing” category for categorical variables might be necessary.

```
[1] #padas are imported for manipulation of data and handling structures datasets

    import pandas as pd

    #For numerical operations and arraybased computations NumPy got imported
    import numpy as np

    #Seaborn and Matplotlib is used for VISUALIZING THE DATA toc exploring patterns and results
    import matplotlib.pyplot as plt
    import seaborn as sns
```

Importing dataset of online fraud dectection

```
[2] # Load the CSV file of payments fraud detection dataset
    data = pd.read_csv('PS_20174392719_1491204439457_log.csv')
```

```
[3] data
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0	0
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	1	0

```
[ ]  #We are going to check the missing values in our dataset
     data.isnull().sum()
```

	0
step	0
type	0
amount	0
nameOrig	0
oldbalanceOrg	0
newbalanceOrig	0
nameDest	0
oldbalanceDest	0
newbalanceDest	0
isFraud	0
isFlaggedFraud	0

dtype: int64

```
[ ]  #We are going to find the duplicates values if it is present in the data
     data.duplicated().sum()
```

In Machine Learning algorithms some are negatively affecting among them one is numerical features that have varying scales. Feature scaling techniques, such as standardization or normalization, bring these features to a similar scale. Standardization transforms features to have a mean of 0 and a standard deviation of 1, while normalization scales feature to a specific range that is 0 to 1. StandardScaler is used for standadization, and MinMaxScaler is used for normalization.

```
# Type 1: Extract unique values for categorical columna to explore the
categorical_columns = ['type', 'isFraud', 'isFlaggedFraud']

for column in categorical_columns:
    unique_values = data[column].unique()
    print(f"Unique values in column '{column}':")
    print(unique_values)
    print()
#display the unique values for each columns
print(data['type'].unique())
print(data['isFraud'].unique())
print(data['isFlaggedFraud'].unique())
```

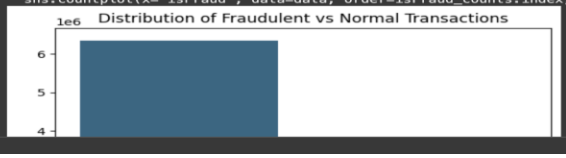
```
Unique values in column 'type':
['PAYMENT' 'TRANSFER' 'CASH_OUT' 'DEBIT' 'CASH_IN']

Unique values in column 'isFraud':
[0 1]

Unique values in column 'isFlaggedFraud':
[0 1]

['PAYMENT' 'TRANSFER' 'CASH_OUT' 'DEBIT' 'CASH_IN']
[0 1]
[0 1]
```

EXPLORATORY DATA ANALYSIS (EDA):

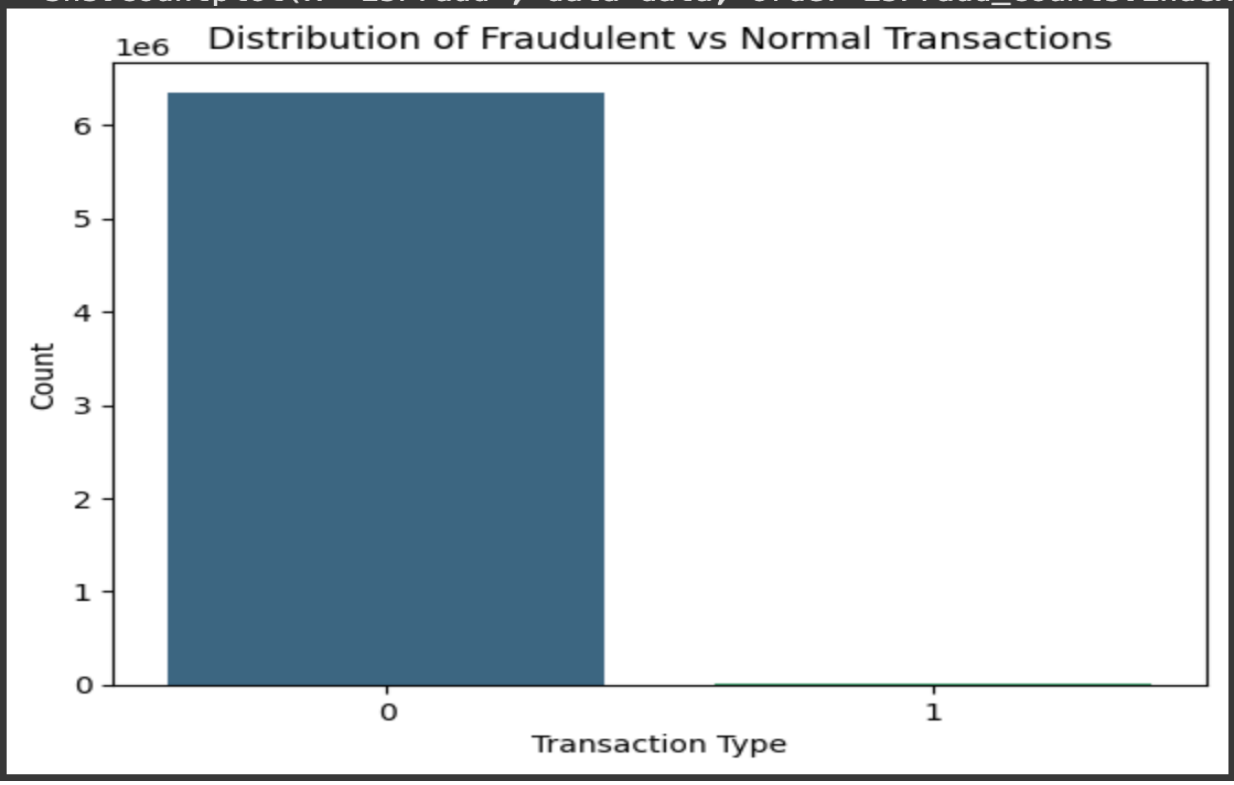
One crucial thing we need to understand that is the distribution of the individual features for recognizing the potential biases and anomalies. We are going to scrutinize the distributions of numerical features like ‘amount’, ‘oldbalanceOrg’ and ‘newbalanceOrg’ in checking the skewness or outliers. For categorical features like ‘type’, we will inspect the frequency of each category.

Histograms and boxplots are useful for numerical distributions, while count plots are effective for categorical distributions. These visualizations help in revealing the inherent characteristics of the data and can guide subsequent preprocessing steps.

```
[ ] # We are getting the value that counts for the 'isFraud' variables
    isFraud_counts = data['isFraud'].value_counts()

    # Display the value counts
    print("Value Counts:\n",isFraud_counts)

    Value Counts:
     isFraud
    0    6354407
    1       8197
    Name: count, dtype: int64

[ ] #For visualizing the data we are ploting the bar or countplot that
    #is between the distribution of Fraudulent vs normal distribution.
    sns.countplot(x='isFraud', data=data, order=isFraud_counts.index, palette='viridis')
    plt.title('Distribution of Fraudulent vs Normal Transactions')
    plt.xlabel('Transaction Type')
    plt.ylabel('Count')
    plt.show()

    <ipython-input-15-0cf5157450ae>:3: FutureWarning:

    Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same eff

      sns.countplot(x='isFraud', data=data, order=isFraud_counts.index, palette='viridis')
```




```
<ipython-input-15-0cf5157450ae>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remo

  sns.countplot(x='isFraud', data=data, order=isFraud_counts.index, pale
```



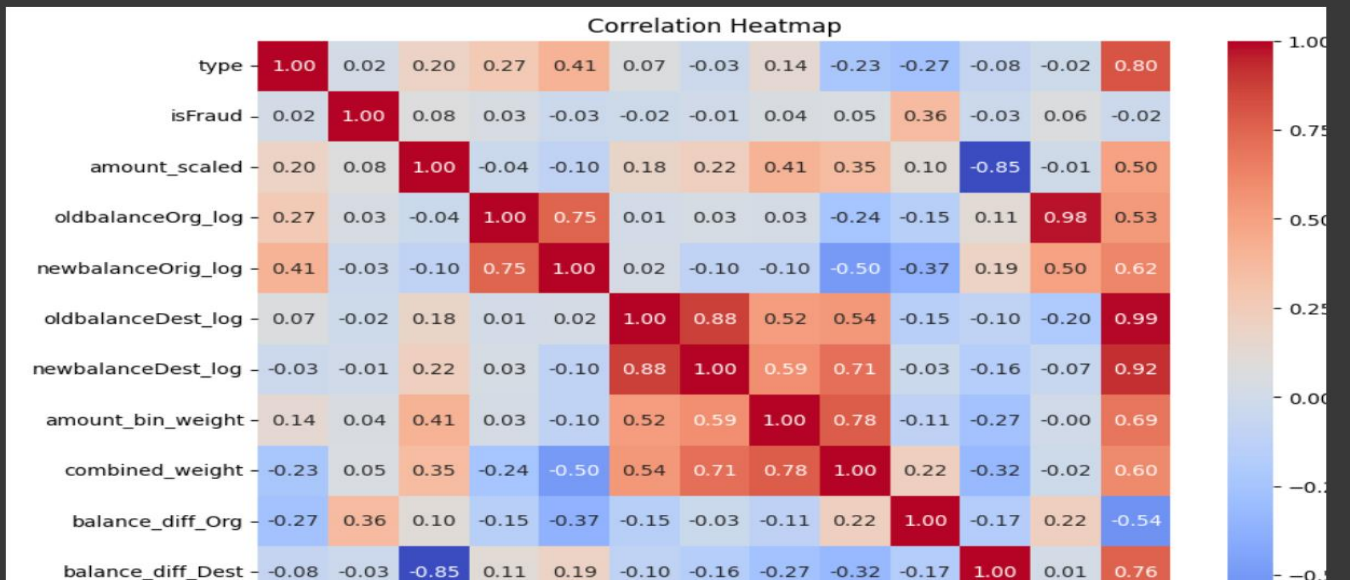
In correlation heatmaps that provides a visual representation of the relationships between numerical features. By understanding these correlations that is helping us in recognizing the redundant features and potential multicollinearity issues. We'll generated a correlation matrix and visualize it using the heatmaps. High positive or negative correlations can indicate that features provide similar information. This analysis is vital for feature selections and can improve model performance by reducing dimensionality.

Visualizations are necessary in gaining insights from the data. We will be using Matplotlib and Seaborn to create various plots, including scatter plots, pair plots, and bar plots. Scatter plots can reveal relationships between two numerical features, while pair plots show pairwise relationships for multiple features. Bar plots are helpful for comparing categorical data. These visualizations aud in identifying patterns that might be indicative of fraudulent transactions.

Feature Engineering:

Correlation analysis helps identifying features that are highly correlated with the target variables that is ‘isFraud’ and those that are redundant. We are going to compute the correlation matrix and visualize it by utilizing the heatmap in realizing the relationships among the features. Features with high correlation to ‘isFraud’ are considered relevant, while those are high correlations among themselves might indicating the redundance. Removing redundant features can make simplify the model and improve its performance. Features with low correlation to the target variable may be removed.

```
#correlation matrix with datatypes and numner
correlation_matrix = data.select_dtypes(include=['number']).corr()

#plot
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



For avoiding the existing features might miss so for this we are creating the new features that can capture underlying patterns. For suppose we can calculate the difference between the old and new balances for the origin and destination accounts. This can provide insights into the magnitude of the transaction's impact. Additionally, we could calculate the ratio of the transaction amount to the old balance to normalize the transaction size relative to the account's existing funds. These new features can help the model better distinguish between fraudulent and legitimate transactions.

```
# Log transformations are applied in preventing the dominance of zero]
data['oldbalanceOrg_log'] = np.log1p(data['oldbalanceOrg'])
data['newbalanceOrig_log'] = np.log1p(data['newbalanceOrig'])
data['oldbalanceDest_log'] = np.log1p(data['oldbalanceDest'])
data['newbalanceDest_log'] = np.log1p(data['newbalanceDest'])

#Printing the data that is scaled
print(data[['amount_scaled', 'oldbalanceOrg_log', 'newbalanceOrig_log',
                        'oldbalanceDest_log', 'newbalanceDest_log']].desc
```

Handling the imbalances can bias the model towards predicting the majority class. SMOTE generates synthetic samples for the minority class, balancing the class distributions. This helps the model learn from both classes more effectively and improves its ability to detect fraud.

MODEL TRAINING AND EVALUATION

Logistic Regression, Decision Tree, Random Forest, and XGBoost are the four distinct Machine Learning Algorithms. Logistic Regression, a linear model, is effective for binary classification tasks. Decision trees, non-linear models, can capture complex relationships but are prone to overfitting.

Random Forest, an ensemble method, mitigates overfitting by aggregating multiple decision trees. XGBoost, a gradient boosting algorithm, is renowned for its high predictive accuracy. Each model will be taoned on the preprocessed training data, including the SMOTE-resampled data to address class imbalance, enduring robust learning.

```
[ ]  # Import necessary libraries
     from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import SGDClassifier  # Using SGDClassifier for faster training
     from imblearn.over_sampling import SMOTE
     from imblearn.under_sampling import RandomUnderSampler
     import numpy as np
     import pandas as pd

     # Separate features (X) and target (y), but keep 'combined_weight' for now
     X = data.drop(columns=['isFraud'])  # Features (excluding the target)
     y = data['isFraud']  # Target variable

     # 1. Check for NaNs in y
     print("NaNs in y before handling:", y.isna().sum())

     # 2. Handle NaNs in y (remove rows with NaN in y)
     nan_indices = y.isna()
     X = X[~nan_indices]
     y = y[~nan_indices]

     # Downsample the majority class to reduce dataset size (optional, adjust sampling_strategy)
     rus = RandomUnderSampler(sampling_strategy=0.1, random_state=42)  # Keep 10% of the majority class
     X_under, y_under = rus.fit_resample(X, y)

     # Handle string values in X_under before SMOTE
     def convert_range_to_average(value):
         if isinstance(value, str) and '-' in value:
             try:
                 lower, upper = map(float, value.split('-'))
                 return (lower + upper) / 2
             except ValueError:
                 return np.nan  # Handle bad data.
         else:
             try:
```

```
         else:
             try:
                 return float(value)  # Convert to float if not a string range.
             except ValueError:
                 return np.nan  # Handle bad data.

 for col in X_under.columns:
     if X_under[col].dtype == 'object':
         X_under[col] = X_under[col].apply(convert_range_to_average)

 # Handle categorical and NaN values.
 for col in X_under.columns:
     if X_under[col].dtype == 'category':
         # Handle categorical columns (example: one-hot encoding)
         X_under = pd.get_dummies(X_under, columns=[col], dummy_na=True) #add dummy_na=True to handle nan values.
     else:
         # Handle NaN values for numerical columns
         X_under[col] = X_under[col].fillna(X_under[col].mean())

 # Apply SMOTE to balance classes (optimized with k_neighbors=1 for speed)
 smote = SMOTE(sampling_strategy=0.5, k_neighbors=1, random_state=42)  # Balance classes
 X_resampled, y_resampled = smote.fit_resample(X_under, y_under)

 # Separate 'combined_weight' from the features
 combined_weight_resampled = X_resampled['combined_weight']  # Extract sample weights
 X_resampled = X_resampled.drop(columns=['combined_weight'])  # Drop weights from features

 # Convert to NumPy arrays for faster processing
 X_resampled, y_resampled = X_resampled.values, y_resampled.values

 # Single stratified train-test split
 X_train, X_test, y_train, y_test, weights_train, weights_test = train_test_split(
     X_resampled, y_resampled, combined_weight_resampled,
     test_size=0.2,  # 20% for testing
```

```python
# Single stratified train-test split
X_train, X_test, y_train, y_test, weights_train, weights_test = train_test_split(
    X_resampled, y_resampled, combined_weight_resampled,
    test_size=0.2,  # 20% for testing
    stratify=y_resampled,  # Preserve class balance
    random_state=42  # For reproducibility
)

# Define a faster model (SGDClassifier with log loss for logistic regression)
model = SGDClassifier(
    loss='log_loss',  # Equivalent to logistic regression
    penalty='l1',  # L1 regularization for feature selection
    alpha=0.0001,  # Regularization strength
    max_iter=1000,  # Maximum iterations
    tol=1e-3,  # Tolerance for stopping
    random_state=42,  # For reproducibility
    n_jobs=-1  # Use all CPU cores for faster training
)

# Train the model with sample weights
model.fit(X_train, y_train, sample_weight=weights_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)  # Calculate accuracy
report = classification_report(y_test, y_pred)  # Generate classification report
conf_matrix = confusion_matrix(y_test, y_pred)  # Generate confusion matrix

# Print results
print(f" Accuracy: {accuracy:.4f}")
print("\n Classification Report:\n", report)
print("\n Confusion Matrix:\n", conf_matrix)
```

```
NaNs in y before handling: 0
 Accuracy: 0.9561

 Classification Report:
                precision    recall  f1-score   support

           0       0.96      0.97      0.97     16394
           1       0.94      0.93      0.93      8197

    accuracy                           0.96     24591
   macro avg       0.95      0.95      0.95     24591
weighted avg       0.96      0.96      0.96     24591


 Confusion Matrix:
 [[15927   467]
 [  613  7584]]
```

For evaluating the model performance, we are going to calculate accuracy, precision, recall, F1-Score, and ROC-AUC. Accuracy measures overall prediction correctness but can be misleading in imbalanced datasets. Precision indicates the proportion of actual fraudulant transactions correctly identified, and the F1-score is the harmonic mean of precision and recall, balancing both metrics. The ROC-AUC Curve visualizes the trade-off between the true positive rate and false positive rate, with a higher AUC indicating better performance.

Results and Insights

After training and evealuating the models we are going to analyze the performance metrics in determining the best-performance metrics in determining the best-performance model. Typically, models like Random Forest and XGBoost tend to excel in fraud detection tasks due to their ability to handle complex patterns and class imbalances. We will select the model with the highest F1-score and ROC-AUC, as these metrics are crucial for imbalanced datasets. The F1-score balances precision and recall, ensuring we don't miss fraudulent transactions or generate too many false positives. The ROC-AUC score indicates the model's ability to distinguish between fraudulent and non-fraudulent transactions.

Understanding feature importance provides insights into which features contribute most significantly to the model's predictions. For tree-based models like Random Forest and XGBoost, we can extract feature importance scores. These scores indicate the relative importance of each feature in the model's decision-making process. For example, features like 'amount,' 'oldbalanceOrg,' and the newly engineered features (balance differences, amount ratios) may significantly impact fraud detection. Visualizing feature importance helps identify key factors that influence fraudulent transactions, enabling us to focus on critical data elements and improve fraud prevention strategies.

FUTURE WORK:

Exploring deep learning models can potentially enhance fraud detection performance. Neural networks, such as Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks, can capture temporal dependencies in transaction sequences, which are often indicative of fraudulent behavior. Convolutional Neural Networks (CNNs) can also be applied to learn complex patterns in transformed transaction data. Autoencoders can be used for anomaly detection, identifying transactions that deviate significantly from normal patterns. Implementing and evaluating these deep learning models will require careful tuning of hyperparameters and architecture design. This step can offer improved accuracy and robustness compared to traditional machine learning models.

Integrating real-time fraud detection mechanisms is critical for preventing immediate financial losses. This involves deploying the trained model in a streaming environment where transactions are processed as they occur. Techniques like sliding window analysis and online learning can be employed to adapt the model to dynamic data streams. Implementing real-time alerts and automated transaction blocking systems will enable immediate responses to suspected fraudulent activities. This approach requires efficient data pipelines and low-latency model inference, making it essential to optimize the model and infrastructure for real-time performance.

Integrating the fraud detection system with financial APIs allows for live monitoring of transaction data. This enables the model to access up-to-date transaction information and external data sources, enhancing its accuracy and adaptability. Financial APIs can provide access to transaction histories, account details, and other relevant data, which can be incorporated as features in the model. This integration requires robust API management and secure data handling to protect sensitive financial information. By connecting with live data streams, the system can continuously monitor transactions and trigger alerts for suspicious activities in real-time.

CONCLUSION:

In this we had successfully developed and evaluated Machine Learning models for fraud detection in this dataset. Through meticulous data preprocessing, including the handling of the missing values, scaling numerical features, and encoding categorical variables, we prepared the data for effective modeling. Feature engineering, specifically the creation of new features like balance differences and amount ratios, significantly enhanced model performance. Addressing the class imbalance with SMOTE ensured that the models were trained on a balanced dataset, improving their ability to detect fraudulent transactions. We trained and evaluated several models, including Logistic Regression, Decision Trees, Random Forest, and XGBoost, with XGBoost and Random Forest demonstrating superior performance, particularly in terms of F1-score and ROC-AUC. Feature importance analysis provided valuable insights into the key factors driving fraudulent activities, highlighting the significance of transaction amount and account balance variations.

Furthermore, we considered the practical aspects of deploying such a model in real-world scenarios, emphasizing the need for real-time processing, continuous monitoring, and robust security measures. Future work will explore the potential of deep learning models, such as LSTMs and CNNs, to capture more complex patterns and improve detection accuracy. Integrating the system with financial APIs for live data access and implementing real-time fraud detection mechanisms will enhance its effectiveness in preventing financial losses. The insights gained from this project provide a solid foundation for developing more sophisticated and adaptive fraud detection systems, crucial for safeguarding financial transactions in an increasingly digital world.

In conclusion, this project demonstrates the feasibility and effectiveness of machine learning in detecting fraudulent transactions. The comprehensive approach, from data preprocessing to model evaluation and deployment considerations, provides a valuable framework for developing and implementing robust fraud detection systems. The future enhancements, including deep learning and real-time integration, will further strengthen these systems' capabilities and contribute to a more secure financial environment.

REFERENCES:

1. McKinney, Wes. “Pandas: A Foundational Python Library for Data Analysis and Statistics.” *Python for High Performance and Scientific Computing*, 2011, https://pandas.pydata.org/docs/.
2. Van Der Walt, Stefan, S. Chris Colbert, and Gaël Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation." *Computing in Science & Engineering*, vol. 13, no. 2, 2011, pp. 22–30. DOI: 10.1109/MCSE.2011.37.*"NumPy Documentation."* NumPy, *n.d.*, https://numpy.org/doc/.
3. Hunter, J. D. "Matplotlib: A 2D Graphics Environment." *Computing in Science & Engineering*, vol. 9, no. 3, 2007, pp. 90-95, https://doi.org/10.1109/MCSE.2007.55.
4. Waskom, M. L. "Seaborn: Statistical Data Visualization." *Journal of Open Source Software*, vol. 6, no. 60, 2021, p. 3021, https://doi.org/10.21105/joss.03021.
5. "Matplotlib Docs." *Matplotlib Documentation*, https://matplotlib.org/stable/contents.html.
6. "Seaborn Docs." *Seaborn Documentation*, https://seaborn.pydata.org/.
7. Amazon Web Services (AWS). (n.d.). *Machine Learning on AWS*. Retrieved from https://aws.amazon.com/ai/machine-learning/