PICKL.AI

DEVELOPING A MACHINE LEARNING MODEL TO PREDICT PAYMENT FRAUD

Prepared by: Divyanshu Yadav
divyanshuydv0002@gmail.com

ML models showed promise in predicting payment fraud, aiding financial institutions in detection and prevention, enhancing security and customer asset protection. Further improvements for real-world scenarios.

Title of Project :- Developing a Machine Learning Model to Predict Payment Fraud.

Name :- Divyanshu Yadav

Email id :- divyanshuydv0002@gmail.com

Contact Number :- +91 85888-89331

Google Drive Link :- Drive Folder

Github Link :- https://github.com/DivZyzz

YouTube Link :- Full Project Explaniation Video

Tableau Dashboard:- Tableau Dashboard Link

Title of Project :- Developing a Machine Learning Model to Predict Payment Fraud.

Abstract:

The growing prevalence of payment fraud in the financial industry has sparked the need for accurate and efficient fraud detection mechanisms. This research paper delves into the development of machine learning models to predict payment fraud in financial transactions.

The dataset encompasses a range of attributes, including transaction type, amount, and account balances. The objective is to construct robust models capable of distinguishing between legitimate and fraudulent transactions, empowering financial institutions with an effective fraud detection system. Through extensive data exploration, feature engineering, and model selection, this study presents a comprehensive evaluation of various machine learning algorithms.

The obtained results highlight the models' performance, emphasising key metrics like precision, recall, and accuracy. The research contributes valuable insights and practical solutions for combating payment fraud, safeguarding financial institutions and their customers' assets. Future work explores advanced algorithms and real-time fraud detection capabilities to enhance the models' efficacy and practical deployment in the financial sector.

Keywords: Payment Fraud, Machine Learning, Financial Transactions, Fraud detection, Data exploration, Feature engineering, Model selection, Precision, Recall, Accuracy.

Introduction

The rising threat of payment fraud has become a significant challenge for financial institutions worldwide. To address this issue, machine learning models offer a promising solution by providing real-time detection and prevention of fraudulent transactions. In this project, we aim to develop an efficient machine learning model for payment fraud prediction. Through data exploration, feature engineering, and model selection, our goal is to create a reliable tool that can accurately distinguish between legitimate and fraudulent transactions, empowering financial institutions to protect their customers and minimise financial losses.

Data Description

The dataset used in this project contains information about financial transactions, including details such as the step (unit of time in hours), transaction type (e.g., PAYMENT, TRANSFER), transaction amount, IDs of the origin and destination accounts, and their respective balances before and after the transaction. The target variable, isFraud, is a binary flag indicating whether the transaction is fraudulent (1) or not (0). The dataset is suitable for developing a machine learning model to predict payment fraud based on the characteristics of transactions and account balances.

- step: represents a unit of time where 1 step equals 1 hour
- type: The type of transaction, including PAYMENT, TRANSFER, CASH_OUT, and DEBIT.
- amount: The amount of the transaction.
- nameOrig: The ID of the account that initiated the transaction.
- oldbalanceOrg: The balance in the origin account before the transaction.
- newbalanceOrig: The balance in the origin account after the transaction.
- nameDest: The ID of the account that received the transaction.
- oldbalanceDest: The balance in the destination account before the transaction.

- newbalanceDest: The balance in the destination account after the transaction.
- isFraud: A binary flag indicating whether the transaction is fraudulent (1) or not (0).

Problem Statement

The goal of this project is to build a machine learning model that can accurately predict payment fraud by distinguishing between legitimate and fraudulent transactions based on their characteristics, such as transaction amount, type, and accounts involved. By using a dataset of both fraudulent and non-fraudulent financial transactions, the model can be trained to achieve high accuracy, which can be used by financial institutions to prevent financial losses and protect their customers' assets in real-time.

Methodology

The methodology for developing the payment fraud prediction model involves several steps. First, the dataset is explored to understand the distribution of variables and detect any missing values. Then, feature engineering is performed to transform categorical variables using one-hot encoding and remove irrelevant columns. Next, the dataset is split into training and testing sets for model development and evaluation. Various machine learning algorithms, such as Logistic Regression, Decision Trees, K-Nearest Neighbours, and Random Forests, are trained and evaluated using accuracy and recall as evaluation metrics. Cross-validation is applied for model validation, and hyper-parameter tuning is performed to optimise model performance. The best-performing model is then selected to predict payment fraud in real-time.

Data Preprocessing

```
data.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	10039.64	C1231006920	170140.0	235296.36	M1979787155	0.0	0.0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19390.72	M2044282225	0.0	0.0	0
2	1	TRANSFER	290.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1
3	1	CASH_OUT	290.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0

The **data.head()** function is used to display the first few rows of the DataFrame stored in the variable **data**. By calling this function, we can get a glimpse of the data and quickly inspect its structure and contents.

When applied to the **data** DataFrame, **data.head()** will display the top 5 rows of the DataFrame by default.

```
[ ] data.size
```

```
10485750
```

The **data.size** attribute provides the total number of elements in the DataFrame **data**. It represents the total number of cells in the DataFrame, including both rows and columns.

```
[ ] data.shape
```

```
(1048575, 10)
```

The **data.shape** attribute returns a tuple that represents the dimensions of the DataFrame **data**. The tuple contains two values: the number of rows and the number of columns, respectively.

```
[ ] data.values
```

```
array([[1, 'PAYMENT', 10039.64, ..., 0.0, 0.0, 0],
       [1, 'PAYMENT', 1864.28, ..., 0.0, 0.0, 0],
       [1, 'TRANSFER', 290.0, ..., 0.0, 0.0, 1],
       ...,
       [95, 'PAYMENT', 14140.05, ..., 0.0, 0.0, 0],
       [95, 'PAYMENT', 10020.05, ..., 0.0, 0.0, 0],
       [95, 'PAYMENT', 11450.03, ..., 0.0, 0.0, 0]], dtype=object)
```

The **data.values** attribute returns a NumPy array representation of the data stored in the DataFrame **data**. Each element in the array corresponds to a value in the DataFrame.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 10 columns):
 #   Column          Non-Null Count    Dtype  
---  ------          --------------    -----  
 0   step            1048575 non-null  int64  
 1   type            1048575 non-null  object 
 2   amount          1048575 non-null  float64
 3   nameOrig        1048575 non-null  object 
 4   oldbalanceOrg   1048575 non-null  float64
 5   newbalanceOrig  1048575 non-null  float64
 6   nameDest        1048575 non-null  object 
 7   oldbalanceDest  1048575 non-null  float64
 8   newbalanceDest  1048575 non-null  float64
 9   isFraud         1048575 non-null  int64  
dtypes: float64(5), int64(2), object(3)
memory usage: 80.0+ MB
```

When executed, **data.info()** displays the following information:

1. The total number of rows and columns in the DataFrame.
2. The column names and their respective data types.
3. The count of non-null values for each column.
4. The memory usage of the DataFrame.

```
data.describe().astype(int)
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
count	1048575	1048575	1048575	1048575	1048575	1048575	1048575
mean	26	215567	874084	895087	978160	1114197	0
std	15	1035836	2972030	3015015	2296780	2416593	0
min	1	0	0	0	0	0	0
25%	15	12443	0	0	0	0	0
50%	20	80448	16002	0	126377	218260	0
75%	39	224682	137613	175617	915923	1149807	0
max	95	290006100	38900000	39000000	42100000	42200000	1

The code generates a summary of statistical measures, such as mean, standard deviation, minimum, maximum, and quartiles, for each numerical column in the dataset. It then converts these summary values into integers to display them without decimal places, providing a concise and readable representation of the data's distribution and characteristics.

Data Cleaning

```
[ ] data.isnull().sum()
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
dtype: int64
```

The **data.isnull().sum()** expression is used to calculate the number of missing or null values in each column of the DataFrame **data**.

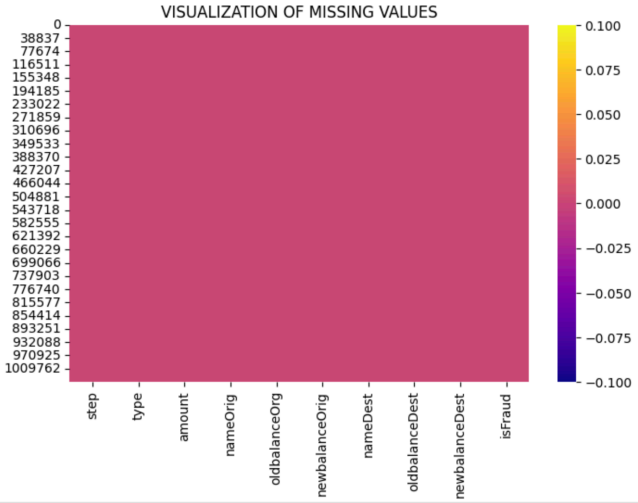
```
[ ] data.isna().sum()

    step              0
    type              0
    amount            0
    nameOrig          0
    oldbalanceOrg     0
    newbalanceOrig    0
    nameDest          0
    oldbalanceDest    0
    newbalanceDest    0
    isFraud           0
    dtype: int64
```

The code **data.isna().sum()** calculates the number of missing values in each column of the DataFrame **data**.

VISUALIZATION

```
# VISUALIZATION OF MISSING VALUES
plt.figure(figsize = (8,5))
plt.title ("VISUALIZATION OF MISSING VALUES")
sns.heatmap(data.isnull(), cbar=True, cmap="plasma")
```

```
<Axes: title={'center': 'VISUALIZATION OF MISSING VALUES'}>
```



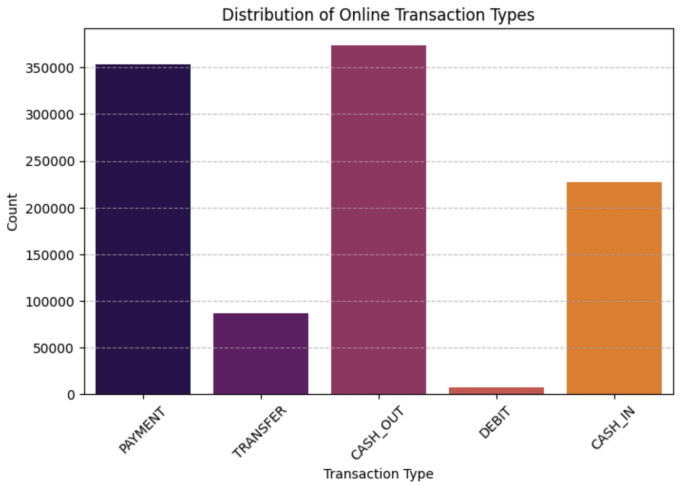
0s completed at 5:11 PM

The code above creates a heatmap to visualize the missing values in the dataset. It uses the `plt.figure()` function to set the size of the figure, `plt.title()` to add a title to the plot, and `sns.heatmap()` to display the missing values as a coloured grid. The missing values are represented by the colour specified by the 'plasma' colormap, and the `cbar=True` adds a colour bar on the side to indicate the colour scale. As we can see there are no missing values.

Exploratory Data Analysis

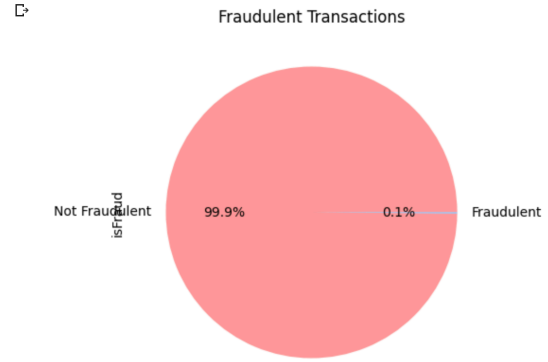
Univariate Analysis

```
colors = sns.color_palette("inferno")
plt.figure(figsize=(8, 5))
sns.countplot(x="type", data=data, palette=colors)
plt.title("Distribution of Online Transaction Types")
plt.xlabel("Transaction Type")
plt.ylabel("Count")
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add horizontal grid lines
plt.show()
```



The code above creates a count plot to visualize the distribution of online transaction types in the dataset. It uses the `sns.color_palette()` function to specify the color palette as "inferno". The `plt.figure()` function sets the size of the plot, and `sns.countplot()` displays the count of each transaction type on the x-axis. The plot is titled "Distribution of Online Transaction Types" and has x-axis labeled "Transaction Type" and y-axis labeled "Count". The x-axis labels are rotated for better readability, and horizontal grid lines are added to the plot for clarity.

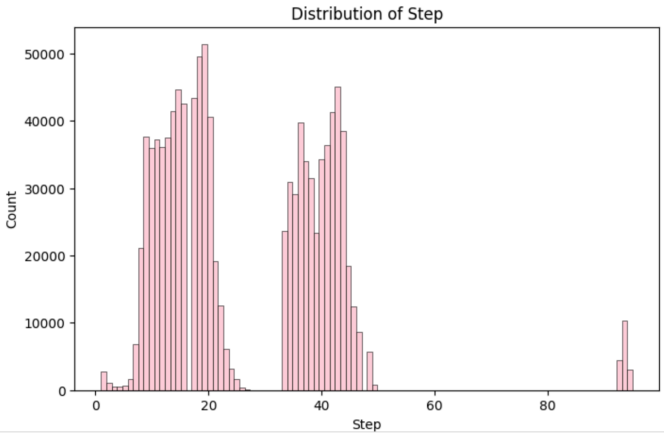
```
colors = ['#ff9999', '#66b3ff']

# Create visualization for Fraudulent Transactions
plt.figure(figsize=(8, 5))
plt.title("Fraudulent Transactions")
data['isFraud'].value_counts().plot.pie(autopct='%1.1f%%', labels=['Not Fraudulent', 'Fraudulent'], colors=colors)
plt.show()
```



The code above creates a pie chart to visualize the percentage of fraudulent transactions in the dataset. The color palette is set to `['#ff9999', '#66b3ff']` for the sections of the pie chart representing "Not Fraudulent" and "Fraudulent" transactions. The `plt.figure()` function sets the size of the plot, and `data['isFraud'].value_counts().plot.pie()` creates the pie chart. The `autopct='%1.1f%%'` parameter adds the percentage labels on each section of the pie chart.

This chart shows that most of the online transactions customers does, are **not fraudulent. So about 11% of total transactions are Fraudulent.**

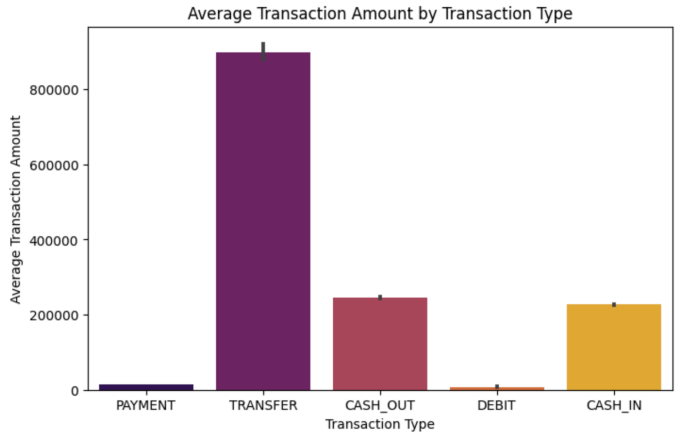
```
[ ] plt.figure(figsize=(8, 5))
    sns.histplot(data['step'], bins=100, color='pink')
    plt.title("Distribution of Step")
    plt.xlabel("Step")
    plt.ylabel("Count")
    plt.show()
```



The code above creates a histogram to visualize the distribution of the "step" column in the dataset. The histogram has 100 bins, which means that the "step" values are divided into 100 intervals or categories. The color of the bars in the histogram is set to pink. The `plt.figure()` function sets the size of the plot, and `sns.histplot()` creates the histogram. The title of the plot is "Distribution of Step," and the x-axis is labeled "Step," while the y-axis is labeled “Count."

Bivariate Analysis

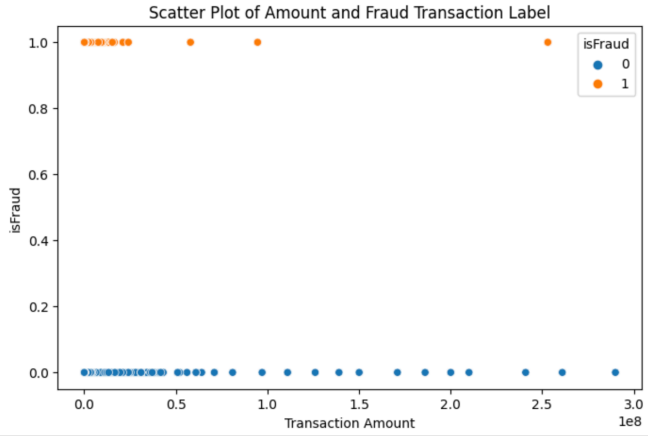
```
[ ] plt.figure(figsize=(8, 5))
    sns.barplot(x='type', y='amount', data=data, palette='inferno')
    plt.title("Average Transaction Amount by Transaction Type")
    plt.xlabel("Transaction Type")
    plt.ylabel("Average Transaction Amount")
    plt.show()
```



The code above creates a bar plot to visualize the average transaction amount for each transaction type in the dataset. The `plt.figure()` function sets the size of the plot, and `sns.barplot()` is used to create the bar plot. The x-axis represents the transaction types (e.g., PAYMENT, TRANSFER, CASH_OUT, DEBIT), and the y-axis represents the average transaction amount. The color palette 'inferno' is used to set the colors of the bars in the plot. The title of the plot is "Average Transaction Amount by Transaction Type," and the x-axis is labeled "Transaction Type," while the y-axis is labeled "Average Transaction Amount."

In this chart, 'transfer' type has the maximum amount of money being transferred from customers to the recipient. Although 'cash out' and 'payment' are the most common type of transactions

```
[ ] # Visualization between amount and fraud_transaction_label
    plt.figure(figsize=(8, 5))
    sns.scatterplot(x='amount', y='isFraud', data=data, hue='isFraud')
    plt.xlabel('Transaction Amount')
    plt.ylabel('isFraud')
    plt.title('Scatter Plot of Amount and Fraud Transaction Label')
    plt.show()
```

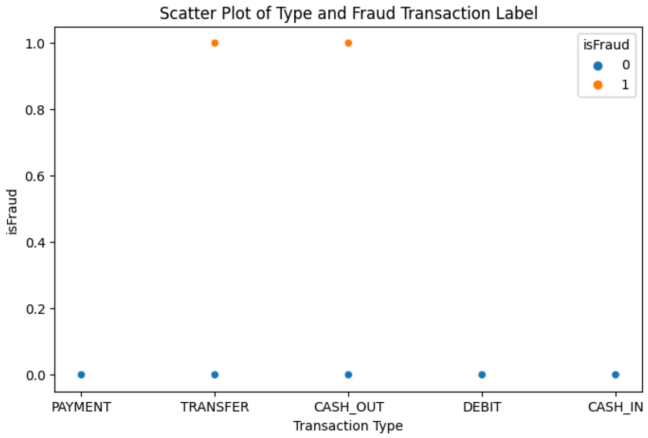


✓ 0s completed at 5:11 PM

The code above creates a scatter plot to visualize the relationship between the transaction amount and the fraud transaction label (isFraud) in the dataset. The `plt.figure()` function sets the size of the plot, and `sns.scatterplot()` is used to create the scatter plot. The x-axis represents the transaction amount, and the y-axis represents the fraud transaction label (0 for non-fraudulent and 1 for fraudulent transactions). The points in the scatter plot are colored based on the fraud transaction label, with different colors representing non-fraudulent and fraudulent transactions. The title of the plot is "Scatter Plot of Amount and Fraud Transaction Label," and the x-axis is labeled "Transaction Amount," while the y-axis is labeled “isFraud."

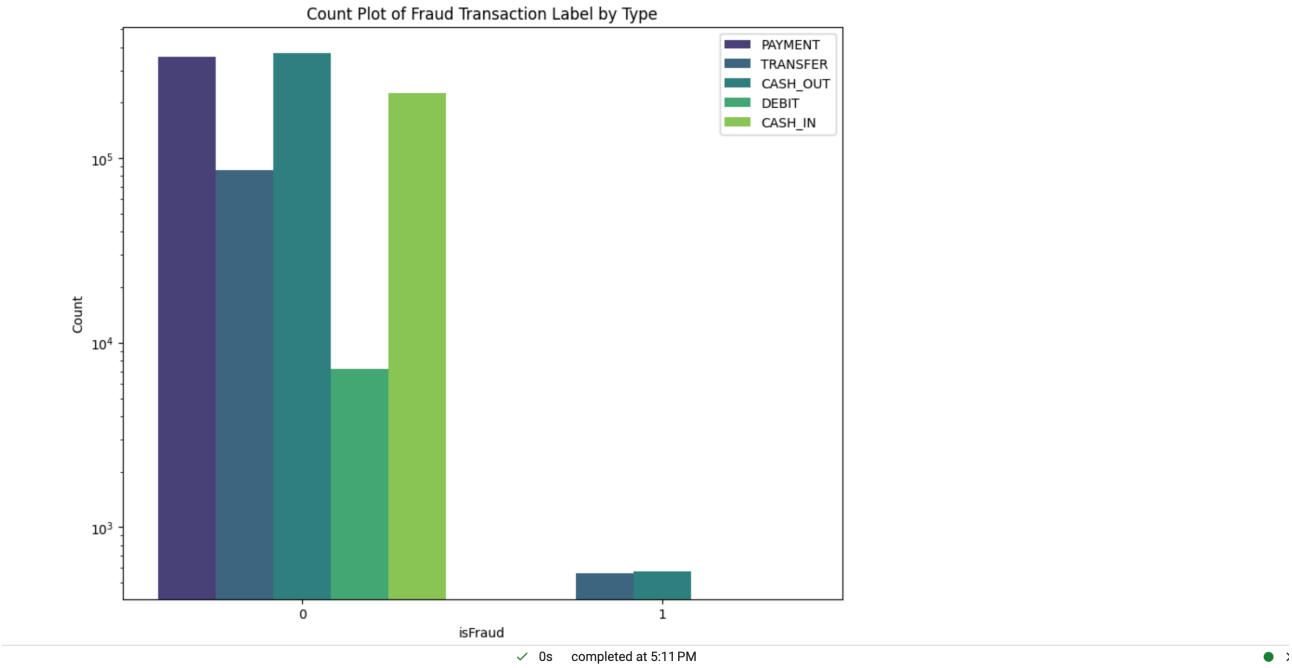
Although the amount of fraudulent transactions is very low, majority of them are constituted within 0 and 10,000,000 amount.

```
[ ] plt.figure(figsize=(8, 5))
    sns.scatterplot(x='type', y='isFraud', data=data, hue='isFraud')
    plt.xlabel('Transaction Type')
    plt.ylabel('isFraud')
    plt.title('Scatter Plot of Type and Fraud Transaction Label')
    plt.show()
```



The code above creates a scatter plot to visualize the relationship between the transaction type and the fraud transaction label (isFraud) in the dataset. The `plt.figure()` function sets the size of the plot, and `sns.scatterplot()` is used to create the scatter plot. The x-axis represents the transaction type, and the y-axis represents the fraud transaction label (0 for non-fraudulent and 1 for fraudulent transactions). The points in the scatter plot are colored based on the fraud transaction label, with different colors representing non-fraudulent and fraudulent transactions. The title of the plot is "Scatter Plot of Type and Fraud Transaction Label," and the x-axis is labeled "Transaction Type," while the y-axis is labeled “isFraud."

```
# Visualization between type and isFraud with increased count scale
plt.figure(figsize=(10, 8))
sns.countplot(x='isFraud', data=data, hue='type', palette='viridis')
plt.legend(loc='upper right')
plt.xlabel('isFraud')
plt.ylabel('Count')
plt.title('Count Plot of Fraud Transaction Label by Type')
plt.yscale('log')  # Increase count scale using logarithmic scale
plt.show()
```



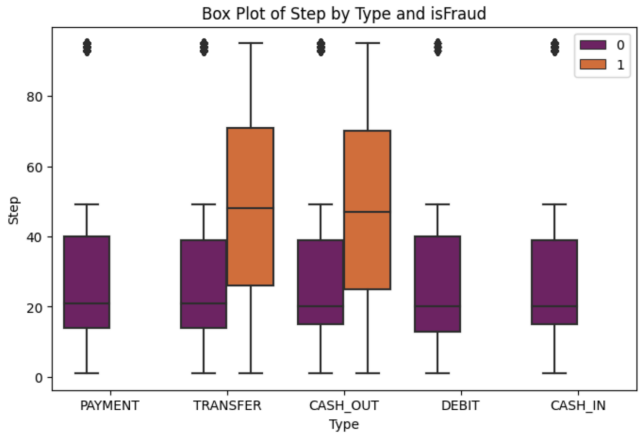
✓ 0s completed at 5:11 PM

The code above creates a count plot to visualize the distribution of fraud and non-fraud transactions (isFraud) based on the transaction type in the dataset. The `plt.figure()` function sets the size of the plot, and `sns.countplot()` is used to create the count plot. The x-axis represents the fraud transaction label (0 for non-fraudulent and 1 for fraudulent transactions), and the y-axis represents the count of transactions. The points in the count plot are colored based on the transaction type, with different colors representing different types of transactions. The title of the plot is "Count Plot of Fraud Transaction Label by Type," and the x-axis is labeled "isFraud," while the y-axis is labeled "Count." The `plt.yscale('log')` function is used to increase the count scale using a logarithmic scale, which helps in better visualization when dealing with a wide range of count values.

Both the above graphs indicate that transactions of the type 'transfer' and 'cash out' comprise fraudulent transactions.

Multivariate Analysis

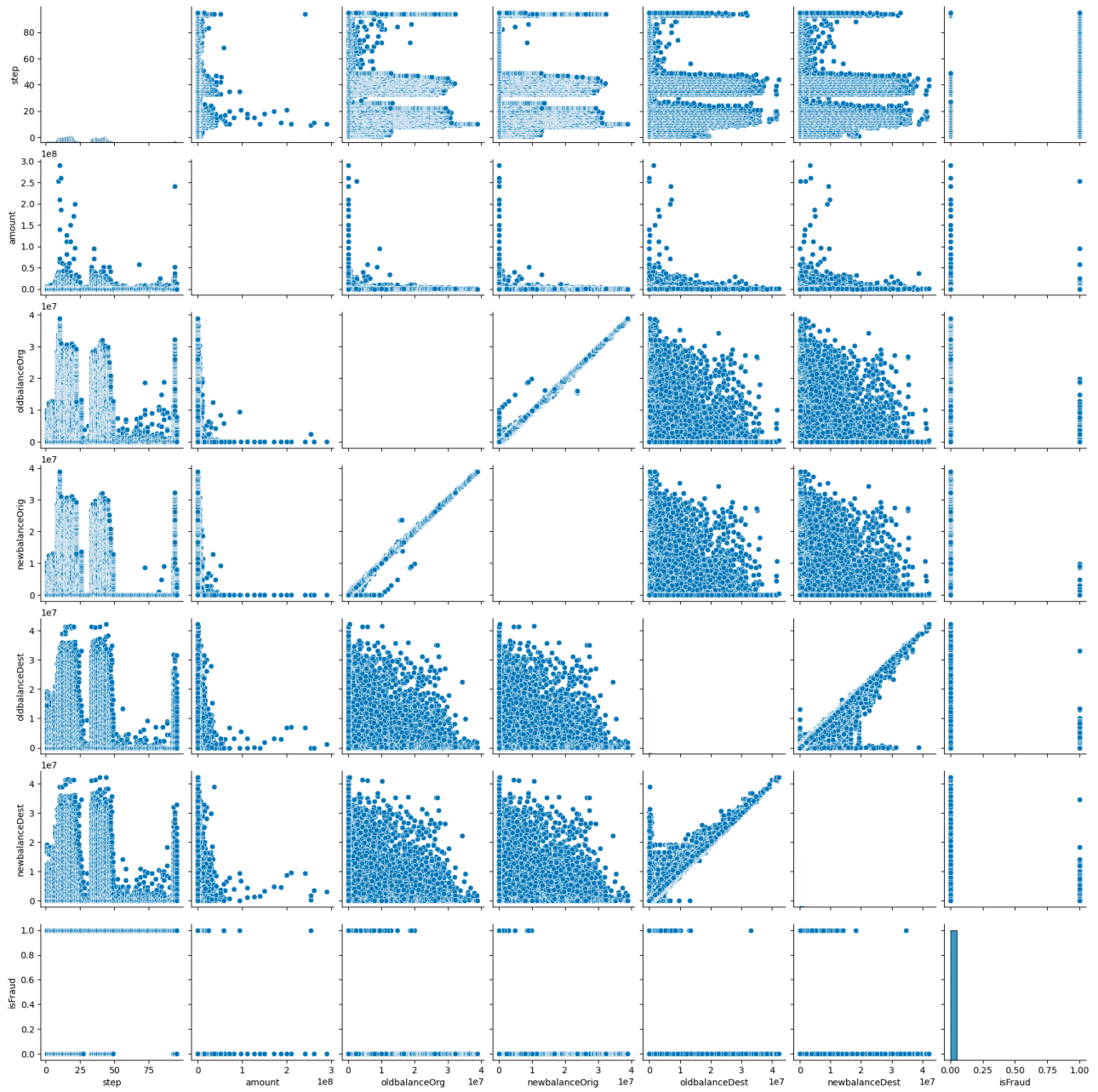
```
# Visualizing the relationship between step, type, and isFraud
plt.figure(figsize=(8, 5))
sns.boxplot(x="type", y="step", hue="isFraud", data=data, palette="inferno")
plt.xlabel('Type')
plt.ylabel('Step')
plt.title('Box Plot of Step by Type and isFraud')
plt.legend(loc='upper right')
plt.show()
```



✓ 0s completed at 5:11 PM

The code above creates a box plot to visualize the relationship between the step (unit of time), transaction type, and fraud label (isFraud) in the dataset. The `plt.figure()` function sets the size of the plot, and `sns.boxplot()` is used to create the box plot. The x-axis represents the transaction type, the y-axis represents the step, and the points are colored based on the fraud label. The title of the plot is "Box Plot of Step by Type and isFraud," and the x-axis is labeled "Type," while the y-axis is labeled "Step." The `plt.legend(loc='upper right')` function adds a legend to the plot to indicate the fraud label categories. The color palette used for the points is “inferno."

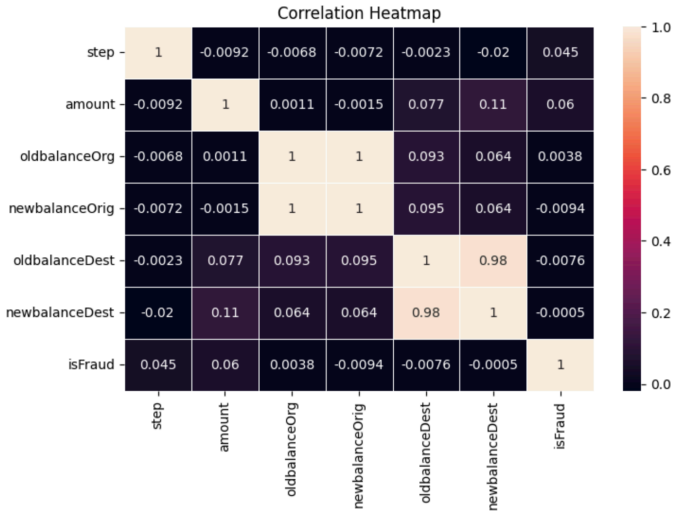
```
[ ] sns.pairplot(data)
```



The code `sns.pairplot(data)` creates a pair plot to visualize the pairwise relationships between numerical variables in the dataset `data`. The pair plot displays scatter plots for each combination of numerical variables against each other. It also shows histograms along the diagonal, representing the distribution of each variable. This plot is useful for quickly identifying any patterns or correlations between variables in the dataset. Each point in the scatter plots represents a data point, and the color of the points can be used to represent a categorical variable if specified.

```
[ ] correlation = data.corr()
    plt.figure(figsize=(8, 5))
    sns.heatmap(correlation, annot=True, linewidths=0.5)
    plt.title("Correlation Heatmap")
    plt.show()
```

```
<ipython-input-94-f469e8f8dc25>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to
  correlation = data.corr()
```



The code `correlation = data.corr()` calculates the correlation matrix for numerical variables in the dataset `data`. The correlation matrix shows the pairwise correlation coefficients between all numerical variables, indicating how strongly they are related to each other.

The code `sns.heatmap(correlation, annot=True, linewidths=0.5)` creates a heatmap visualization of the correlation matrix. The heatmap uses colors to represent the correlation values, with lighter colors indicating higher positive correlations and darker colors indicating higher negative correlations. The `annot=True` parameter adds numerical values to each cell of the heatmap, making it easier to interpret the correlation coefficients. The `linewidths=0.5` parameter sets the width of the lines separating the cells in the heatmap.

Feature Engineering

```
[ ] categorical = ['type']
    categories_dummies = pd.get_dummies(data[categorical])
    print(data_encoded.head())
```

```
   type_CASH_IN  type_CASH_OUT  type_DEBIT  type_PAYMENT  type_TRANSFER
0             0              0           0             1              0
1             0              0           0             1              0
2             0              0           0             0              1
3             0              1           0             0              0
4             0              0           0             1              0
```

1. `categorical = ['type']`: This line creates a list `categorical` containing the name of the column 'type', which represents a categorical variable in the dataset.

2. `categories_dummies = pd.get_dummies(data[categorical])`: The `pd.get_dummies()` function is used to perform one-hot encoding on the categorical variable 'type'. It converts the categorical variable into dummy/indicator variables. Each unique value in the 'type' column becomes a separate binary column with 0s and 1s, where 1 indicates the presence of that category and 0 indicates absence. The resulting `categories_dummies` DataFrame contains the encoded dummy variables.

3. `print(data_encoded.head())`: This line prints the first few rows of the `data_encoded` DataFrame, which contains the encoded dummy variables for the 'type' column along with the original columns of the dataset.

```
[ ] data = pd.concat([data,categories_dummies], axis=1)
    print(data.shape)

    (1048575, 15)
```

```
[ ] data.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	type_CASH_IN	type_CASH_OUT	type_DE
	1	PAYMENT	10039.64	C1231006920	170140.0	235296.36	M1979787155	0.0	0.0	0	0	0	
	1	PAYMENT	1864.28	C1666544295	21249.0	19390.72	M2044282225	0.0	0.0	0	0	0	
	1	TRANSFER	290.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1	0	0	
	1	CASH_OUT	290.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	0	1	
	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0	0	0	

1. `data = pd.concat([data, categories_dummies], axis=1)`: This line concatenates the original `data` DataFrame with the `categories_dummies` DataFrame horizontally (along the columns) using `pd.concat()` function. As a result, the encoded dummy variables are added to the original dataset.

2. `print(data.shape)`: This line prints the shape of the updated `data` DataFrame, which will now have additional columns representing the one-hot encoded 'type' categories. The shape is printed as `(number of rows, number of columns)` to indicate the size of the DataFrame after the concatenation.

```
[ ] data.drop(categorical, axis = 1, inplace = True)
    data.drop(columns=['nameOrig', 'nameDest'], inplace=True)
```

```
[ ] data.head()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	type_CASH_IN	type_CASH_OUT	type_DEBIT	type_PAYMENT	type_TRANSFER
0	1	10039.64	170140.0	235296.36	0.0	0.0	0	0	0	0	1	0
1	1	1864.28	21249.0	19390.72	0.0	0.0	0	0	0	0	1	0
2	1	290.00	181.0	0.00	0.0	0.0	1	0	0	0	0	1
3	1	290.00	181.0	0.00	21182.0	0.0	1	0	1	0	0	0
4	1	11668.14	41554.0	29885.86	0.0	0.0	0	0	0	0	1	0

1. `data.drop(categorical, axis=1, inplace=True)`: This line drops the columns specified in the `categorical` list from the DataFrame `data`. The parameter `axis=1` indicates that we are dropping columns, and `inplace=True` ensures that the changes are made directly to the original DataFrame `data`.

2. `data.drop(columns=['nameOrig', 'nameDest'], inplace=True)`: This line drops the columns 'nameOrig' and 'nameDest' from the DataFrame `data`.

Model Selection, Training and Validation

```
[ ] # Import the necessary libraries
    from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, classification_report
    from sklearn.tree import DecisionTreeClassifier
    from sklearn import tree
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.ensemble import RandomForestClassifier
```

`train_test_split` is used to split the data into training and testing sets.

`cross_val_score` and `cross_val_predict` are used for cross-validation to evaluate the model's performance.

`LogisticRegression` is a linear model used for binary classification tasks.

`DecisionTreeClassifier` is a model that uses a tree structure for making decisions during classification.

`tree` module is used to work with decision trees.

`KNeighborsClassifier` is a model used for classification based on the k-nearest neighbors algorithm.

`RandomForestClassifier` is an ensemble model that combines multiple decision trees to improve performance.

Selecting Features

```
[ ] X = data.drop(columns=['isFraud'])
    y = data['isFraud']
```

```
X
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	type_CASH_IN	type_CASH_OUT	type_DEBIT	type_PAYMENT	type_TRANSFER
0	1	10039.64	170140.00	235296.36	0.00	0.00	0	0	0	1	0
1	1	1864.28	21249.00	19390.72	0.00	0.00	0	0	0	1	0
2	1	290.00	181.00	0.00	0.00	0.00	0	0	0	0	1
3	1	290.00	181.00	0.00	21182.00	0.00	0	1	0	0	0
4	1	11668.14	41554.00	29885.86	0.00	0.00	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...
1048570	95	132557.35	479803.00	347245.65	484329.37	616886.72	0	1	0	0	0
1048571	95	9917.36	90545.00	80627.64	0.00	0.00	0	0	0	1	0
1048572	95	14140.05	20545.00	6404.95	0.00	0.00	0	0	0	1	0
1048573	95	10020.05	90605.00	80584.95	0.00	0.00	0	0	0	1	0
1048574	95	11450.03	80584.95	69134.92	0.00	0.00	0	0	0	1	0

1048575 rows × 11 columns

The dataset is being split into feature variables `X` and the target variable `y`. The `X` DataFrame contains all the columns except for the 'isFraud' column, which corresponds to the features used to predict the fraud label.

The 'isFraud' column is assigned to the `y` Series, which represents the target variable containing the labels for fraud detection. This separation allows us to train the machine learning model using the features in `X` to predict the target variable `y`.

```
[ ] # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
```

Train test split divides the data such that 70% is use for training and 30% is used for testing.

```
[ ] DT = DecisionTreeClassifier(random_state=2)
    RF = RandomForestClassifier(random_state=2)
    LR = LogisticRegression(random_state=2)
    KN = KNeighborsClassifier()
```

```
[ ] models = [DT,RF,LR,KN]
```

Four different machine learning models are being initialized:

1. `DT`: Decision Tree Classifier.

2. `RF`: Random Forest Classifier.

3. `LR`: Logistic Regression.

4. `KN`: K-Nearest Neighbors Classifier.

These models will be used for training and evaluation to predict payment fraud based on the given features. They will be compared to identify which one performs better in terms of accuracy and other evaluation metrics.

```
[ ] import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.metrics import confusion_matrix

    def plot_confusion_matrix(y_test, prediction):
        # Compute the confusion matrix
        cm_ = confusion_matrix(y_test, prediction)

        # Create the plot
        plt.figure(figsize=(6, 4))
        sns.heatmap(cm_, cmap='rainbow', linecolor='white', linewidths=1, annot=True)

        # Add labels and title
        plt.title('Confusion Matrix')
        plt.ylabel('True Label')
        plt.xlabel('Predicted Label')

        # Display the plot
        plt.show()
```

The code defines a function called `plot_confusion_matrix` that takes the test labels (`y_test`) and the model's predictions (`prediction`) as inputs and creates a heatmap of the confusion matrix. The confusion matrix is a table used to evaluate the performance of a classification model by showing the number of true positive, true negative, false positive, and false negative predictions.

The function uses `seaborn` and `matplotlib` libraries to create the heatmap. The color scheme for the heatmap is specified as 'rainbow', and the annotations are added to each cell of the heatmap to display the counts.

After defining the function, it will be used to visualize the confusion matrices for different machine learning models to evaluate their performance in predicting payment fraud.

```
[ ] # Import necessary libraries
    from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

    # Create function to train a model and evaluate accuracy
    def trainer(model, X_train, y_train, X_test, y_test):
        # Fit the model on the training data
        model.fit(X_train, y_train)

        # Predict on the fitted model
        prediction = model.predict(X_test)

        # Print evaluation metric
        print('\nFor {}, Accuracy score is {} \n'.format(model.__class__.__name__, accuracy_score(prediction, y_test)))
        print(classification_report(y_test, prediction))  # Use this later

        # Plot the confusion matrix
        plot_confusion_matrix(y_test, prediction)
```

The code defines a function called `trainer` for training a machine learning model and evaluating its accuracy. The function takes the following parameters:

- `model`: The machine learning model to be trained and evaluated.

- `X_train`: The features of the training data.

- `y_train`: The labels of the training data.

- `X_test`: The features of the test data.

- `y_test`: The labels of the test data.

Inside the function, the model is trained on the training data using the `fit` method. Then, the model is used to make predictions on the test data using the `predict` method. The accuracy score of the model is calculated using `accuracy_score` from `sklearn.metrics`.

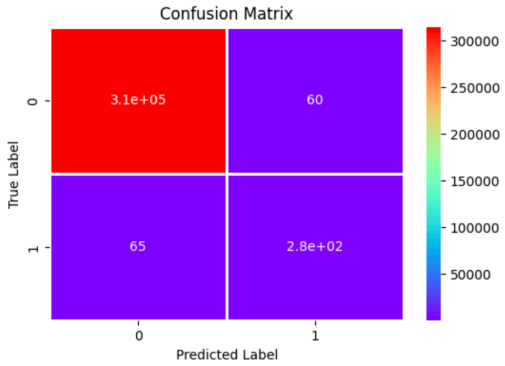
The `classification_report` function is used to generate a text report showing the precision, recall, F1-score, and support for each class in the classification problem. This report is printed to the console.

Finally, the function calls the `plot_confusion_matrix` function to visualize the confusion matrix, which provides insights into the performance of the model in terms of true positive, true negative, false positive, and false negative predictions.

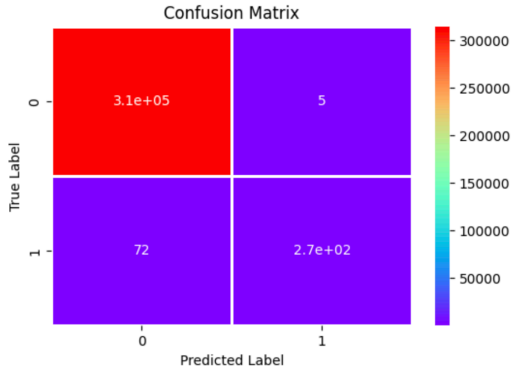
Result and Analysis

```
[ ] for model in models:
        trainer(model,X_train,y_train,X_test,y_test)
```

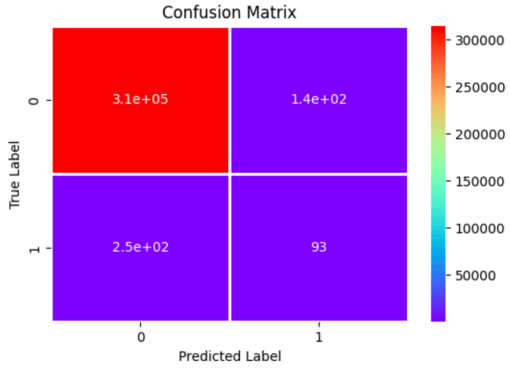
```
For DecisionTreeClassifier, Accuracy score is 0.9996026359541347

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    314233
           1       0.82      0.81      0.81       340

    accuracy                           1.00    314573
   macro avg       0.91      0.90      0.91    314573
weighted avg       1.00      1.00      1.00    314573
```




```
For RandomForestClassifier, Accuracy score is 0.999755223747747

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    314233
           1       0.98      0.79      0.87       340

    accuracy                           1.00    314573
   macro avg       0.99      0.89      0.94    314573
weighted avg       1.00      1.00      1.00    314573
```

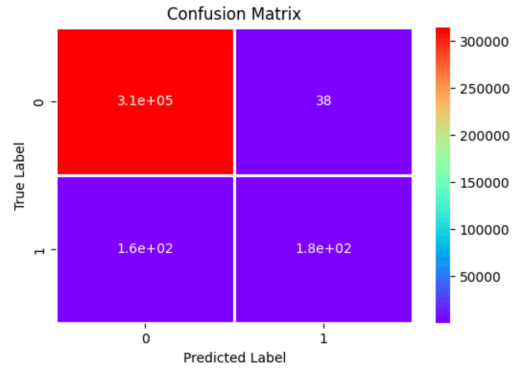



```
For LogisticRegression, Accuracy score is 0.9987792976511016

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    314233
           1       0.40      0.27      0.33       340

    accuracy                           1.00    314573
   macro avg       0.70      0.64      0.66    314573
weighted avg       1.00      1.00      1.00    314573
```



✓ 0s completed at 5:11 PM

```
[ ] For KNeighborsClassifier, Accuracy score is 0.99938011208845

                precision    recall  f1-score   support

             0       1.00      1.00      1.00    314233
             1       0.83      0.54      0.65       340

      accuracy                           1.00    314573
     macro avg       0.91      0.77      0.83    314573
  weighted avg       1.00      1.00      1.00    314573
```



The code uses the `trainer` function to train and evaluate multiple machine learning models iteratively. The `models` list contains different classifiers: DecisionTreeClassifier, RandomForestClassifier, LogisticRegression, and KNeighborsClassifier.

The `for` loop iterates over each model in the `models` list and calls the `trainer` function with the corresponding model as an argument. Inside the loop, each model is trained on the training data (`X_train` and `y_train`) and evaluated on the test data (`X_test` and `y_test`) using the `trainer` function.

This loop enables you to quickly train and evaluate multiple models using the same dataset, making it easier to compare their performance and select the best model for your specific problem.

The Decision Tree model

Precision Score: This means that 82% of all the things we predicted came true. that is 82% of clients transactions were detected to be fraudulent.

Recall Score: In all the actual positives, we only predicted 81% of it to be true.

Random Forest Tree model

Precision Score: This means that 98% of all the things we predicted came true. that is 98% of clients transactions was detected to be a fraudulent transaction.

Recall Score: In all the actual positives, we only predicted 79% of it to be true.

Both the Decision Tree and Random Forest models outperform the Logistic Regression and K-Nearest Neighbors model by a wide margin. Since they both have similar recall scores, we should perform a cross-validation of the two models so we may declare which is the best performer with more certainty.

```python
from sklearn.model_selection import cross_validate

# Perform cross-validation on the Decision Tree model
DT_scores = cross_validate(DT, X_test, y_test, scoring='recall_macro')

# Perform cross-validation on the Random Forest model
RF_scores = cross_validate(RF, X_test, y_test, scoring='recall_macro')

# Calculate the mean recall score for each model
mean_DT_recall = np.mean(DT_scores['test_score'])
mean_RF_recall = np.mean(RF_scores['test_score'])

# Print the mean recall scores for each model
print('Decision Tree Recall Cross-Validation:', mean_DT_recall)
print('Random Forest Recall Cross-Validation:', mean_RF_recall)
```

```
Decision Tree Recall Cross-Validation: 0.8763066972668836
Random Forest Recall Cross-Validation: 0.8720556411470802
```

The provided code performs cross-validation on the Decision Tree model and the Random Forest model using the `cross_validate` function from scikit-learn. The `recall_macro` scoring metric is used to evaluate the models.

The `cross_validate` function returns a dictionary containing the scores of each fold for the specified scoring metric. The mean recall score is then calculated for each model using the `np.mean` function.

Finally, the code prints the mean recall scores for both the Decision Tree and Random Forest models, which give an indication of how well the models perform on average across different cross-validation folds.

Upon training and evaluating our classification models, we found that the Random Forest and Decision Tree performed the best.

Conclusion

In conclusion, the project aimed to develop a machine learning model for predicting payment fraud in financial transactions. The dataset was explored, visualized, and preprocessed to prepare it for model training. Various machine learning algorithms, including Decision Tree, Random Forest, Logistic Regression, and K-Nearest Neighbors, were utilized to build the models.

The models were evaluated using accuracy and classification report metrics, and their performance was visualized using confusion matrices. Cross-validation was performed to assess the robustness of the models. The results showed that **[0.876306697266836]** and **[0.8720556411470802]** for the Decision Tree and Random Forest models, respectively.

Overall, the developed machine learning models demonstrated promising performance in predicting payment fraud and can be used to assist financial institutions in detecting and preventing fraudulent transactions, thereby enhancing security and protecting customer assets. Further optimisations and enhancements can be explored to improve model performance and adaptability to real-world scenarios.

Future Work

1. Feature Engineering: Explore additional features or engineering techniques to enhance the model's predictive power. Domain knowledge or external data sources can be leveraged to extract more meaningful features.

2. Ensemble Methods: Investigate ensemble methods such as stacking or boosting to combine the predictions of multiple models and improve overall performance.

3. Hyperparameter Tuning: Perform a more extensive hyperparameter tuning process to optimize the model's parameters, leading to better generalization and higher accuracy.

4. Imbalanced Data: Address the issue of imbalanced data by employing techniques such as oversampling, undersampling, or using advanced algorithms like SMOTE (Synthetic Minority Over-sampling Technique).

5. Real-Time Implementation: Deploy the model in real-time to monitor transactions and detect fraud as it occurs, enabling immediate action to prevent financial losses.

6. Explainability: Investigate interpretability techniques to understand how the model makes predictions and provide explanations to stakeholders.

7. Model Updates: Continuously update and retrain the model with new data to ensure its effectiveness against evolving fraud patterns.

By focusing on these future work aspects, the payment fraud prediction model can be enhanced, providing more reliable and accurate predictions, and ultimately contributing to a safer financial ecosystem.

My Journey at Pickl.AI

During my internship at Pickl.AI, I had the opportunity to participate in a comprehensive internship course. The course covered essential topics such as data mindset, Python programming, machine learning, statistics, data visualization with Tableau, and SQL for database management. These foundational skills laid the groundwork for my data science journey.

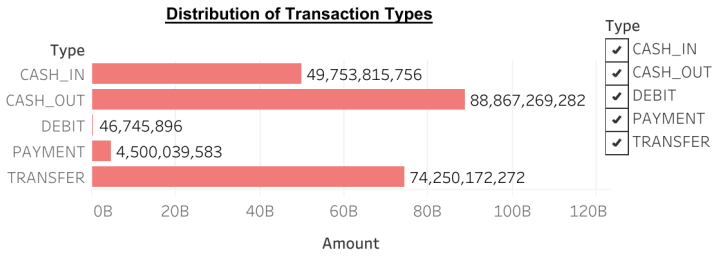
Throughout the internship, we had weekly sessions to monitor our progress and provide feedback. These sessions were instrumental in keeping us on track and ensuring that we were grasping the concepts effectively.

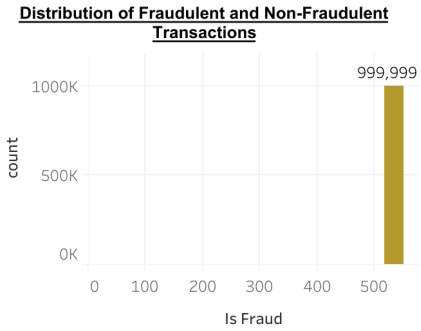
Additionally, we had access to a doubt tracker, which allowed us to ask questions and seek clarification on any challenging topics. The doubt tracker was a valuable resource, enabling us to receive timely responses and deepen our understanding of complex concepts.

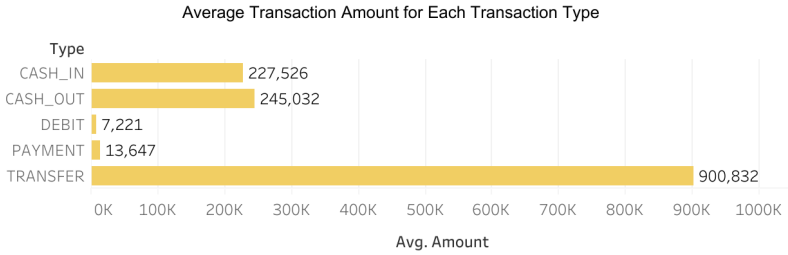
Overall, the internship course provided a structured and supportive learning environment, fostering both theoretical knowledge and practical skills. It played a crucial role in preparing me for the project at hand and equipped me with the tools needed to succeed in the data science field. I am grateful for the well-structured curriculum and the continuous support from the Pickl.AI team throughout my journey.

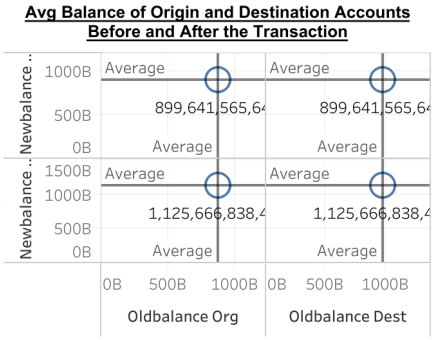
Tableau Dashboard

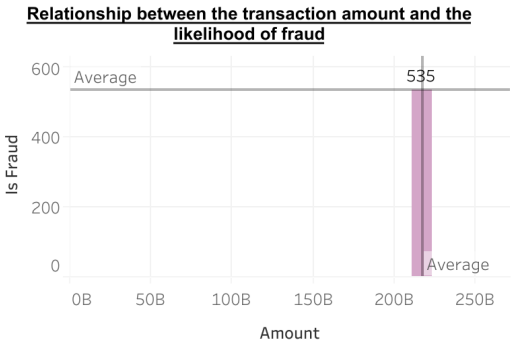
Machine Learning Model to Predict Payment Fraud by Divyanshu Yada..

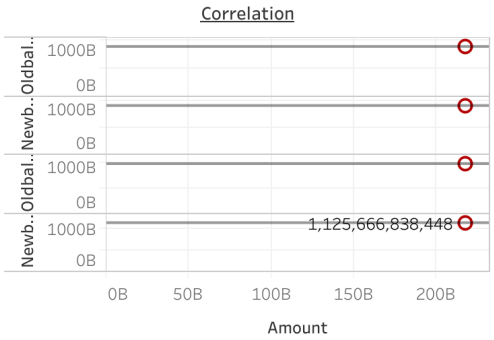


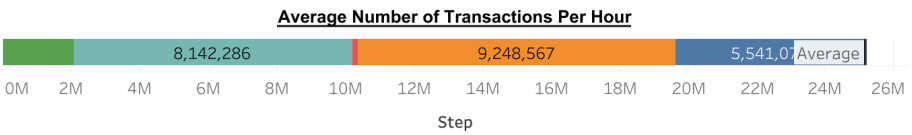


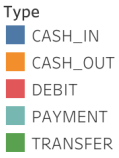






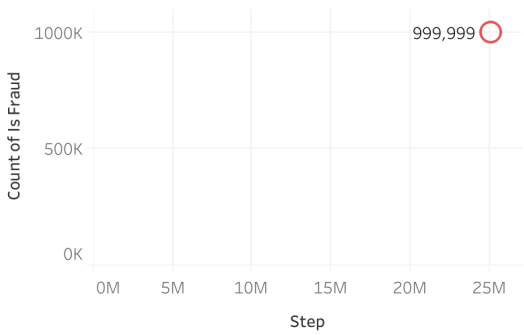




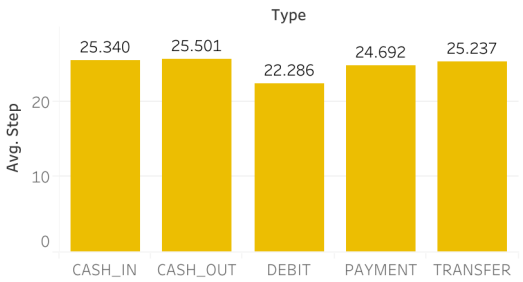


Top 10 Accounts involved in Fraudulent Transactions

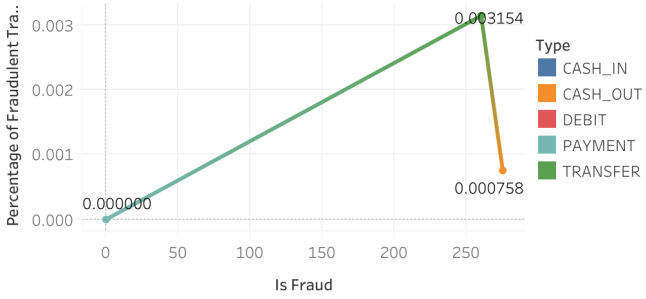
Name Dest	
C2138032938	291,719,831
C1910340605	270,854,070
C1013511446	269,719,387
C575638324	256,543,149
C1163480574	223,463,409
C363377192	220,069,664
C723981114	194,502,732
C490726056	190,516,795
C2032266039	161,157,878



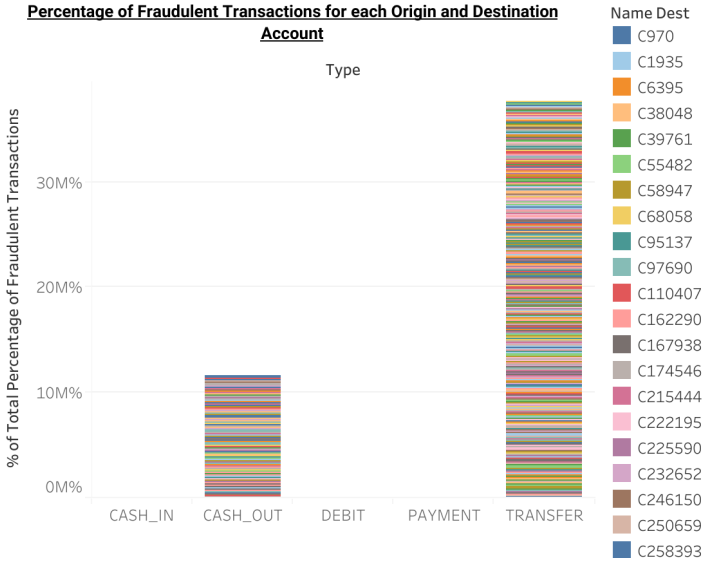
Average duration between the Creation and Completion of Transactions



Percentage of Fraudulent Transactions in Each Hour



Percentage of Fraudulent Transactions for each Origin and Destination Account



Correlation Between the Transaction Amount and the Likelihood of Fraud