Final Project

Credit Card Fraud Detection Using Machine Learning

Credit Card Fraud

- Detecting credit card fraud is crucial for protecting both financial institutions and customers.

- By identifying fraudulent activity early, companies can prevent unauthorized transactions, minimize financial losses, and build trust with customers.

Credit Card Statistic

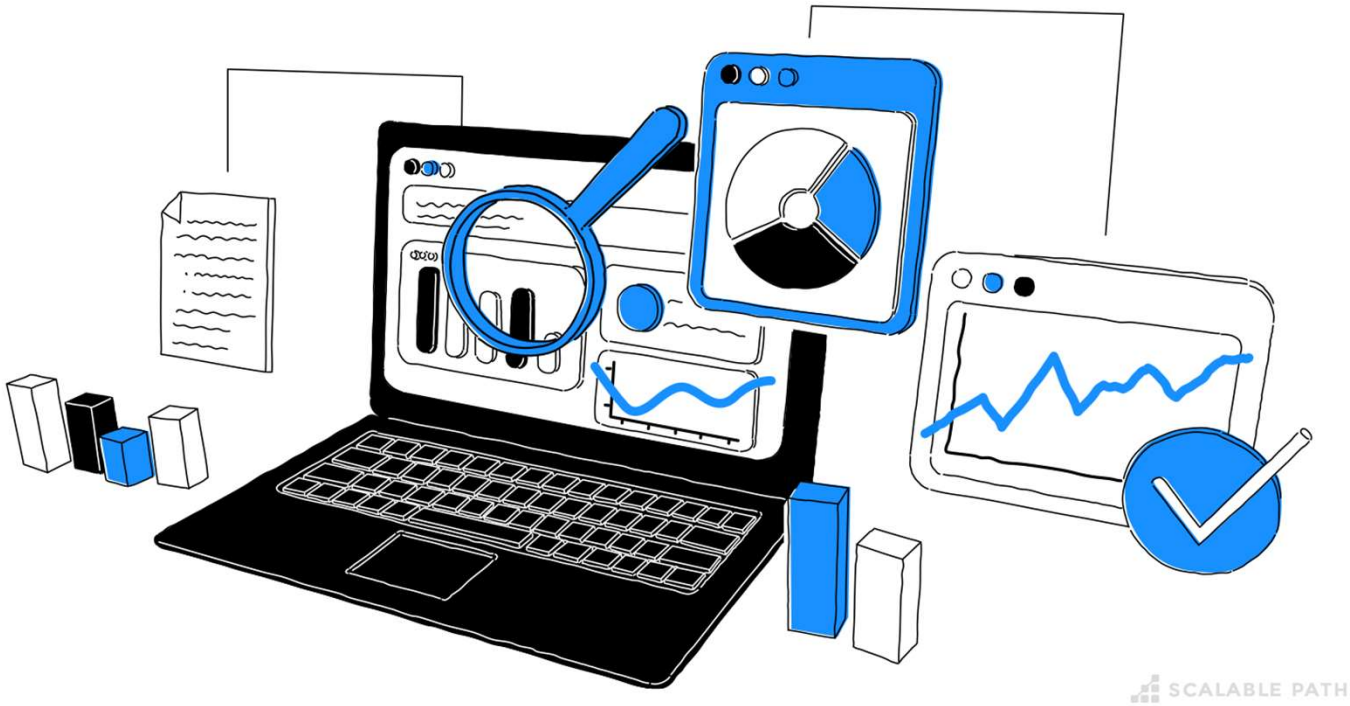
- **Losses**: In 2023, global credit card fraud losses were estimated to surpass $32 billion, and this figure has been growing annually. Losses are projected to exceed $40 billion by 2027 if current trends continue.
- **U.S. Dominance in Fraud Losses**: The U.S. consistently records the highest credit card fraud rates globally, accounting for nearly 36% of all card fraud losses. This is partly due to the widespread use of credit cards and higher adoption of online shopping.
- **Card-Not-Present (CNP) Fraud**: CNP fraud, which includes online or phone transactions where the card isn't physically presented, makes up **70-80% of all fraud cases**. This has been increasing with the growth of e-commerce.
- Around **40% of consumers** in North America have experienced at least one fraudulent transaction in the past year, leading to higher demand for fraud alerts and monitoring services.

Data Source

- The dataset used for this project is sourced from the Credit Card Fraud Detection dataset, available at Kaggle

- https://www.kaggle.com/code/gpreda/credit-card-fraud-detection-predictive-models/input.

Exploration & Analysis



Fraudulent Transactions

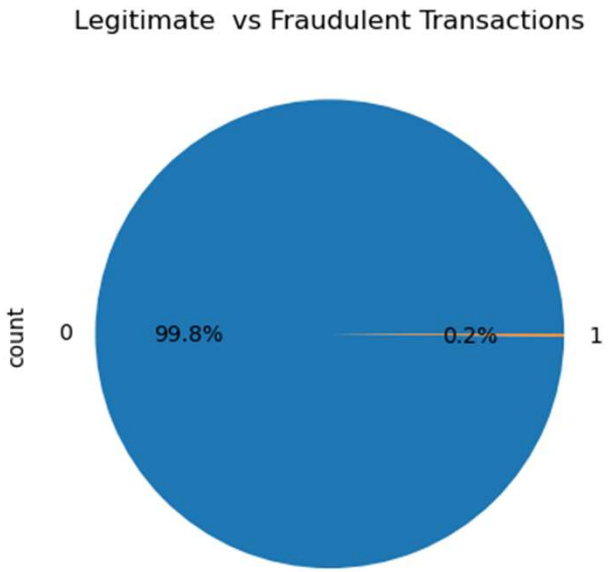
This dataset, which contains 284,807 rows and 31 columns.

Target Value is Class, which has 1 and 0 value. It is an integer type variable.

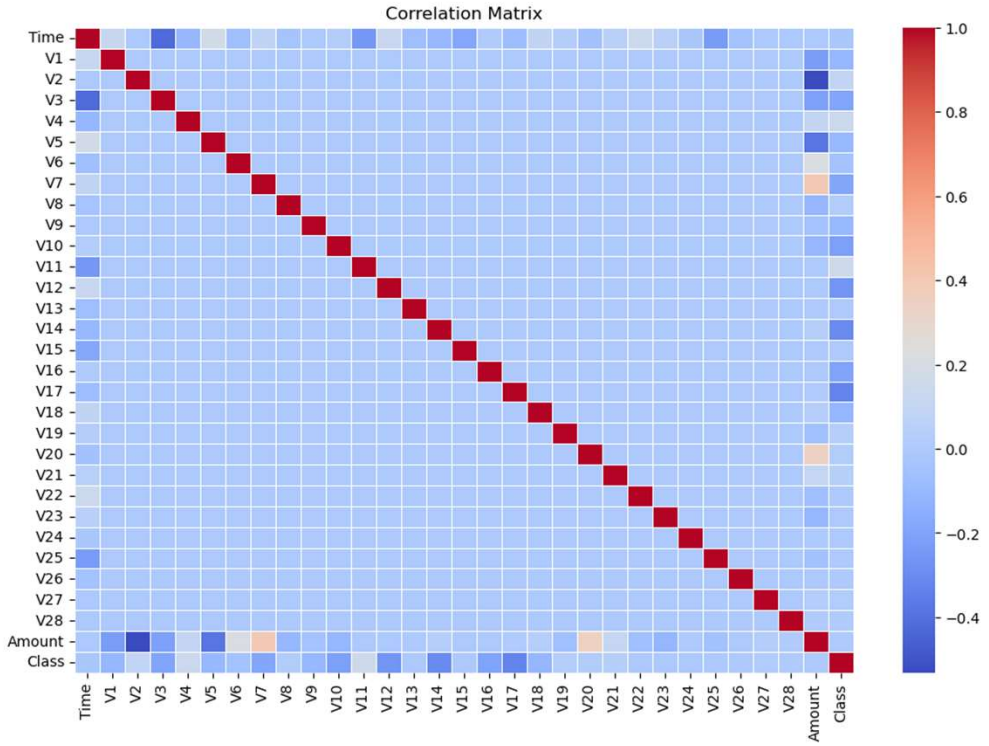
0 indicates legitimate transaction and 1 indicates fraud transaction

There are no null values

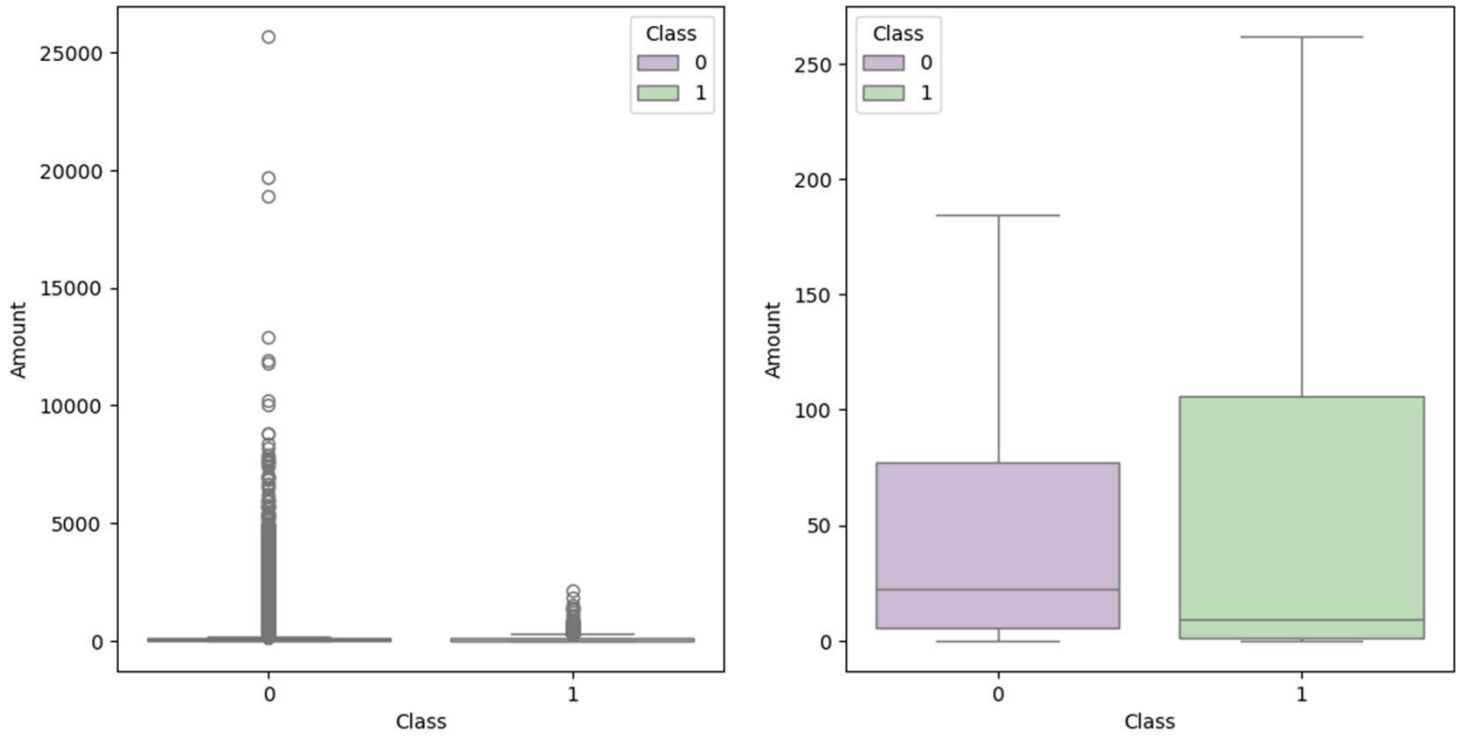
There are no duplicate values



Heat Map Analysis



Box Plot Class VS Amount



Supervised Learning

- Decision Tree
- Random Forest
- Logistic Regression
- XGBoost

Supervised Learning Out Comes

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.9953	0.4892	0.5549	0.3713
Random Forest	0.6698	0.6234	0.8069	0.5372
Decision Tree	0.7995	0.4706	0.7395	0.4833
XGBoost	0.999	0.88	0.84	0.86

Unsupervised Learning

- K-Means

- K-Means Accuracy : 0.5450392722088994

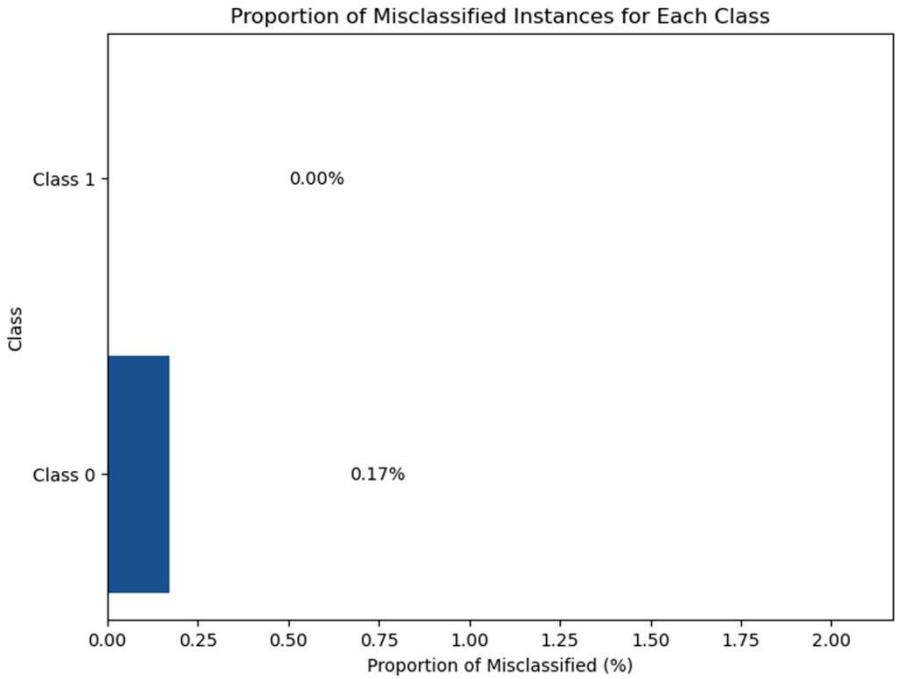
Deep Learning

- Sequential Model

```
]: # Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define a simple deep learning model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.1, verbose=1)

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy:", accuracy)
```

- Test Accuracy: 0.99983

Deep Learning Out Comes



Number of Misclassified Instances: 98

Proportion of Class 0 Misclassified: 0.17%

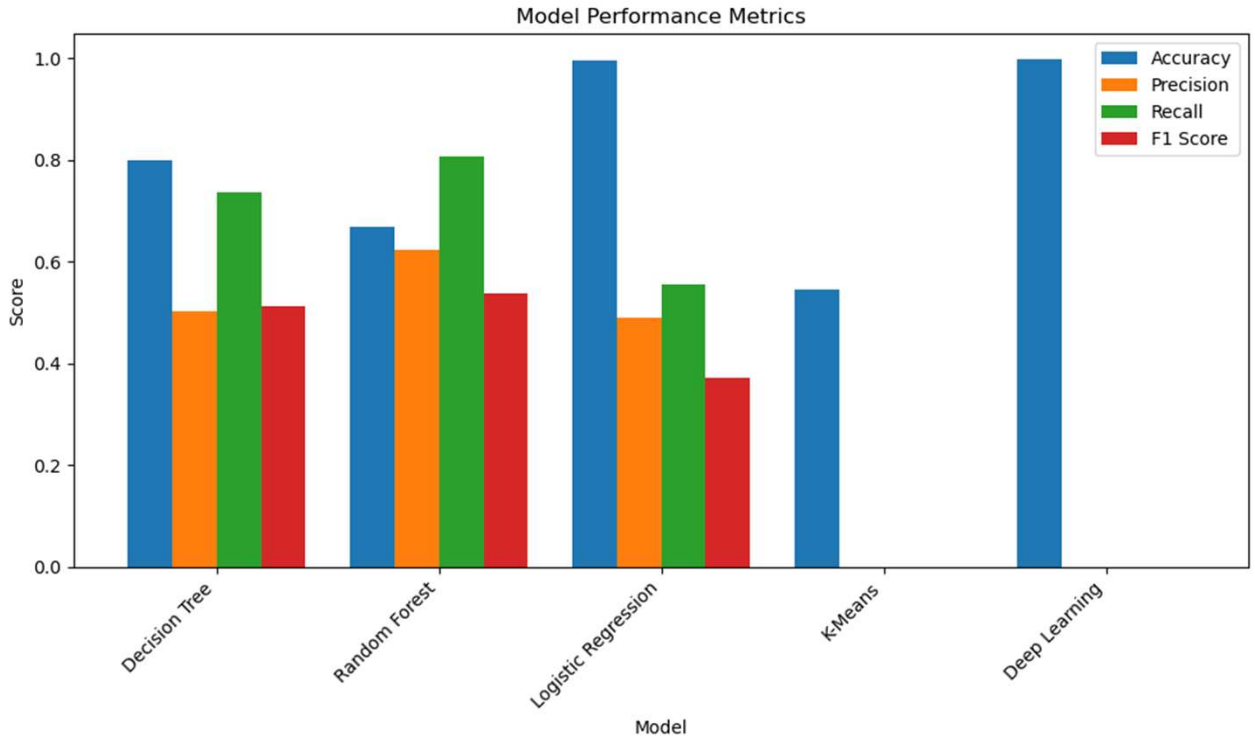
Proportion of Class 1 Misclassified: 0.00%

XGBoost

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Initialize the XGBoost classifier
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', scale_pos_weight=len(y_train[y_train==0]) / len

# Fit the model
xgb_model.fit(X_train, y_train)

# Make predictions
y_pred = xgb_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print('Classification Report:')
print(report)
```

```
Accuracy: 0.9995259997893332
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.88      0.84      0.86        98

    accuracy                           1.00     56962
   macro avg       0.94      0.92      0.93     56962
weighted avg       1.00      1.00      1.00     56962
```

Model Comparision



Observations

Model Performance Variability:
The accuracy of the models varied significantly, with Logistic Regression (0.995) and Deep Learning (0.998) showing exceptionally high accuracy. In contrast, Random Forest (0.670) and K-Means (0.545) performed poorly, indicating variability in model effectiveness.

Class Imbalance:
There is a substantial class imbalance in the dataset, with Class 0 having significantly more samples (56,864) than Class 1 (98). This imbalance heavily influences the models, particularly affecting precision and recall.

Precision and Recall Discrepancies:
High accuracy does not correlate with good precision and recall across all models. For instance, the Deep Learning model achieved high accuracy but failed to classify any instances of Class 1 (0.00 precision and recall).

XGBoost's Balanced Performance:
XGBoost exhibited a strong performance across both classes, achieving good precision (0.88) and recall (0.84) for Class 1, making it a favorable choice for practical applications.

Model Suitability:
While some models like Decision Trees and Random Forests had decent overall accuracy, they struggled with classifying the minority class effectively. This is evident in their lower precision and F1 scores.

Challenges

Class Imbalance Impact:
The significant imbalance between Class 0 and Class 1 led to models being biased towards the majority class, which is a common challenge in classification problems.
Overfitting:
Models like Deep Learning potentially overfit the training data, as indicated by their ability to accurately predict Class 0 while failing to recognize Class 1. This raises concerns about generalization.
Computational Resources:
Training and tuning advanced models like Deep Learning and XGBoost require significant computational resources and time, which can be a barrier, especially with large datasets.

Conclusion

Model Selection:
Based on the results, XGBoost appears to be the most effective model for this classification task, demonstrating a balanced ability to predict both classes while handling the class imbalance better than others.
Need for Class Handling Strategies:
To improve overall model performance, especially for minority classes, implementing strategies to handle class imbalance—such as resampling techniques or adjusted class weights—should be prioritized.
Further Model Refinement:
Continuous refinement and tuning of the selected models, particularly XGBoost and the Deep Learning model, will likely yield improvements in classification performance.