ITCS 3156 Final Project
Agampreet Bajaj

Credit Card Fraud Detection: Machine Learning Approach Using Transactional Features

Problem Introduction:

In this project, I want to understand the problem of credit card fraud and detect it using supervised learning methods. Because I am going into a full-time job this summer, I want to understand more about online payments and e-commerce. Fraud is pretty rare in comparison to the huge volume of legitimate transactions, making this a hard challenge in practical machine learning.

The goal of this project is to build machine learning models to predict whether a certain transaction is fraudulent or legitimate, based on a set of features. This will be a binary classification problem, with a highly imbalanced target variable. I will not be looking solely at overall accuracy, as that would be a big problem from this class imbalance. I will focus on different metrics, such as precision, recall, F1-score, and the area under the curve (AUC) for a Receiver Operating Characteristic curve (ROC).

This pipeline will include exploratory data analysis and visualizations, data preprocessing, model training, model evaluation, and a comparison of 2 different machine learning models. In this scenario, I will use Logistic Regression as a baseline model and Random Forest to capture the complex nonlinear relationships in the data. I hope to understand how these models detect fraud, which features are most important, and how it all works in a real-world setting.

Data Introduction:

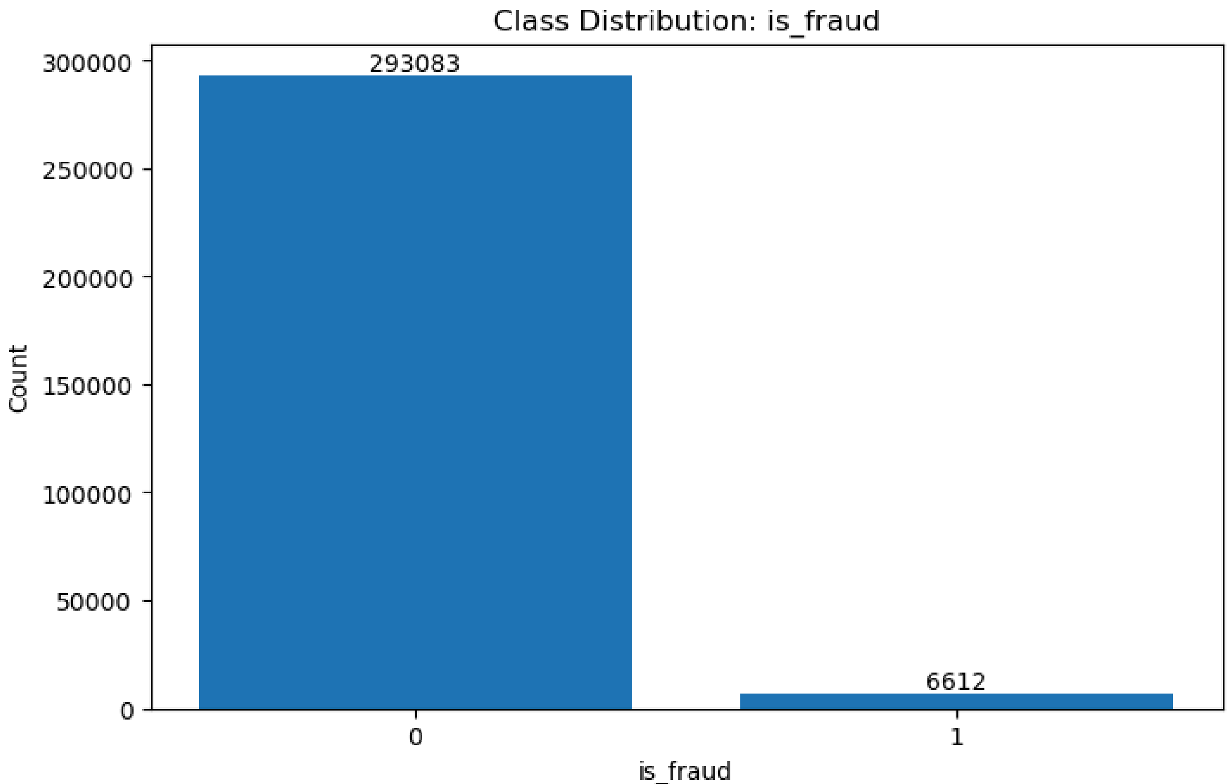
The dataset for this project is a large collection of credit card transactions. Each row is a single transaction from a user, with each column representing properties of the user, the transaction, or the outcome (fraudulent or not). Our goal will use the column features to predict the outcome (**is_fraud**), where 0 represents a legitimate transaction and 1 represents a fraudulent transaction.

This dataset contains a base of 299,695 transactions and 17 attributes, providing a great amount of data for training and testing models. However, the target variable (**is_fraud**) is highly imbalanced: Where only 6,612 transactions are under the label of fraud, and the remaining 293,083 transactions are legitimate. This can be represented as 2.2% fraud and 97.8% legitimate transactions. The imbalance makes this problem realistic, but more challenging. Let us take a closer look at our base attributes:

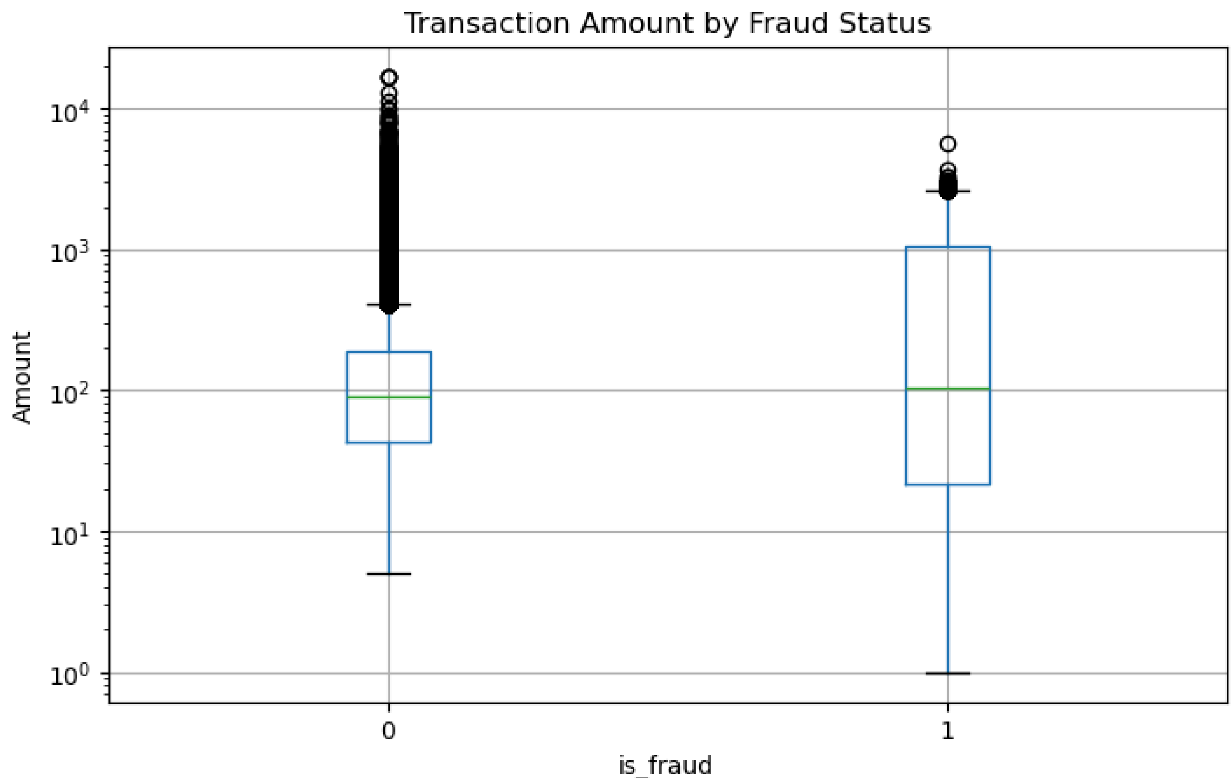
transaction_id	ID	Unique identifier for each transaction
user_id	ID	Identifier used for what user initiated what transaction
account_age_days	Numeric	Number of days since account creation
total_transactions_user	Numeric	Total number of past transactions by user
avg_amount_user	Numeric	Average transaction amount for the user
amount	Numeric	Amount of current transaction
country	Categorical	Country connected to the user
bin_country	Categorical	Country connected to the card issuer
channel	Categorical	Channel of transaction (online, in-person, etc.)
merchant_category	Categorical	Category of merchant (retail, travel, etc.)
promo_used	Numeric	Checks for promotion or coupon
avs_match	Numeric	Result of AVS check
cvv_result	Numeric	Result of CVV check
three_ds_flag	Numeric	Indicates if 3DS authentication was used
transaction_time	Datetime	Timestamp of transaction occurrence
shipping_distance_km	Numeric	Distance between billing and shipping locations
is_fraud	Binary	1 if transaction is fraud, 0 otherwise (legitimate)

Later on, many of these raw features will be transformed to be more useful. Some columns that serve as identifiers, rather than predictors, will be excluded from the modeling process. As seen, this dataset provides a realistic foundation for building and evaluating my project's models.

Data Visualization/Analysis:



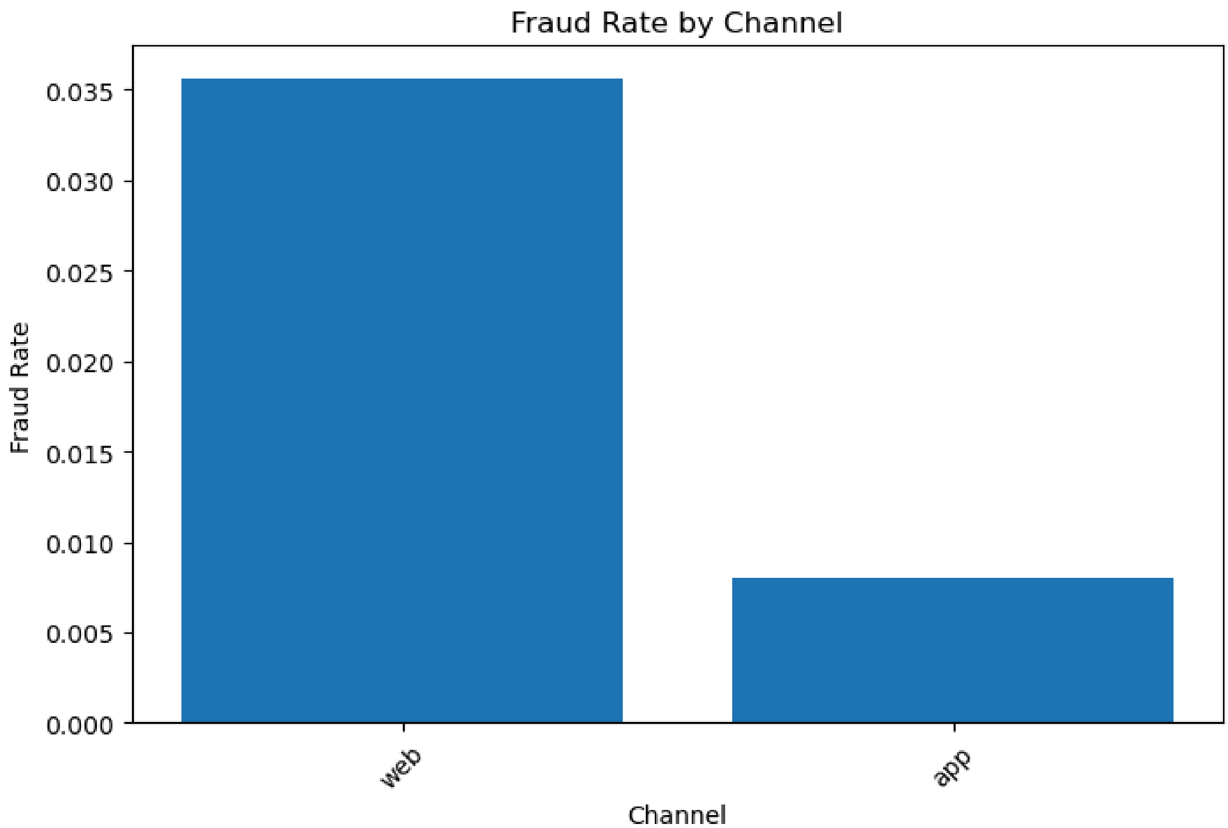
This graph shows the distribution of the target variable. The majority of the transactions are labeled as non-fraudulent, while just a fraction of the data is fraudulent. This imbalance implies that a naive model could do well here in achieving high accuracy. But this also means we need to rely more on other metrics.



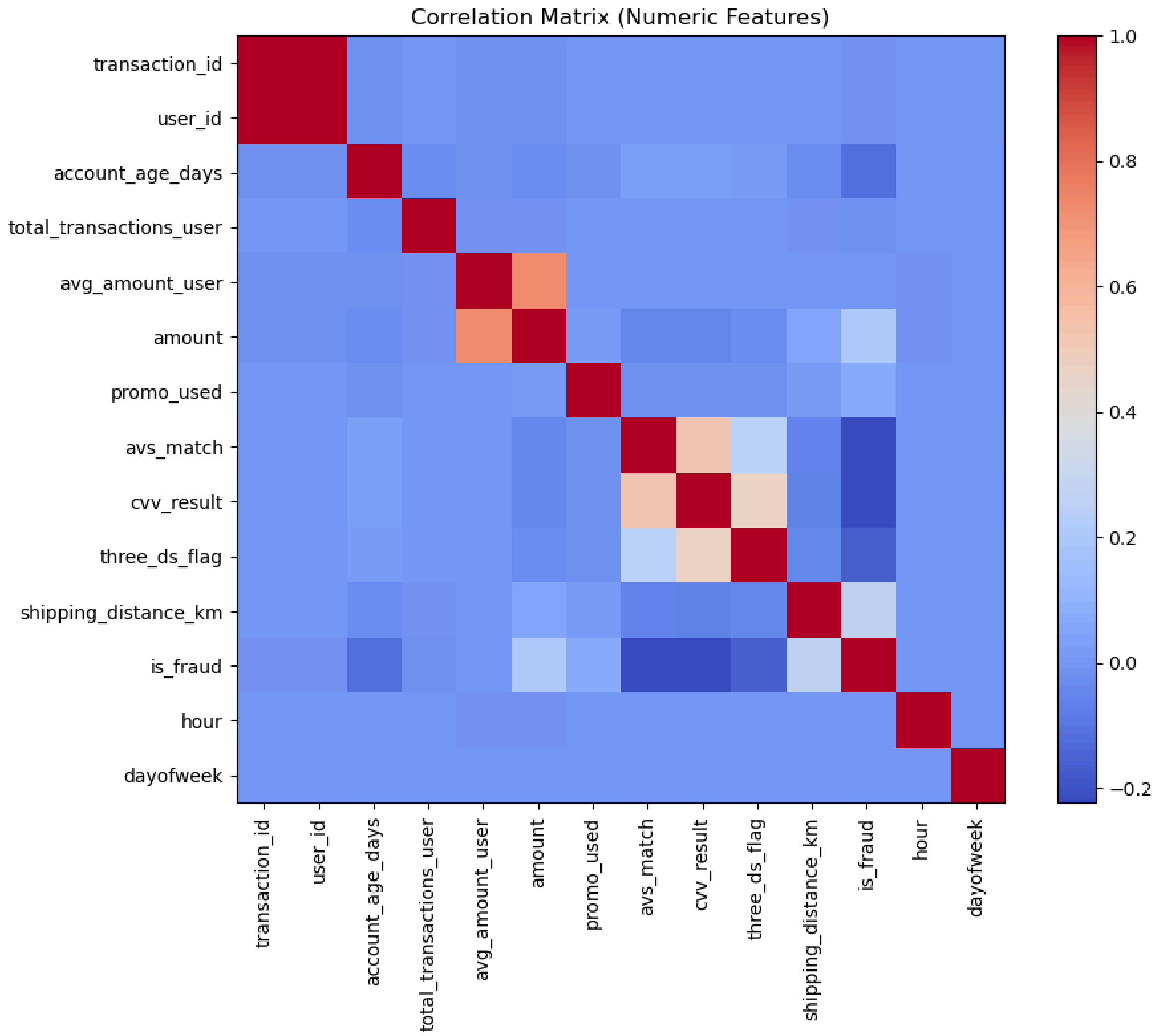
This graph shows the transaction amount by fraud status. There is a substantial overlap in both groups, where fraudulent transactions seem to have more high-value outliers and a wider spread of data towards large values. This means unusually large values or are more likely to be flagged as fraud, but this doesn’t mean the amount alone is enough to separate classes. The log scale on the y-axis is used to help visualize the amounts in transactions on the plot.

```
Fraud rate by channel:
```

channel	fraud_rate	num_transactions
web	0.035644	152226
app	0.008042	147469



This graph shows the fraud rate based on the transaction channel. For each category of channel (in this instance, website vs phone app for banking), the bar represents the proportion of fraudulent transactions. In these results, fraudulent transactions are much more likely to happen web-wise than app-wise. This tells us that certain channels (like websites) are more exposed to risky behaviors and have less secure environments. There can also be common ways to enact fraud on a website, as it came out much before mobile applications.



This is a correlation matrix shown for all numeric features. Each cell is the Pearson correlation between different variables. Dark colors show strongly correlated relationships. An example is our target variable of **is_fraud,** having a low linear correlation with these individual features. This is expected because fraud detection is an extremely complex problem and needs powerful models that can use complex patterns to capture non-linear interactions.

Preprocessing and Feature Engineering:

1. Convert timestamp to time features
2. Remove non-predictive identifier columns
3. Check for missing values
4. Define target and feature matrix
5. Create train-test split (stratified)
6. Identify categorical and numerical categories
7. Build pipeline

Before model training, I applied many series of preprocessing and feature engineering steps to prep the data. Our main goals were to convert the raw fields into meaningful features, removing columns that could cause overfitting, and build a reusable preprocessing pipeline to be applied to the data.

The first step was to engineer time-based features from the column, **transaction_time**. I converted the column to a proper datetime type and extracted the hour and dayofweek feature from it. These features can potentially capture fraud occurrences across different hours and days, which I previously observed during EDA. After this, I dropped the original column of **transaction_time** since the raw feature was not directly useful for modeling.

Next, I removed identifier columns that don't have any information for prediction. I dropped the transaction and user IDs, as they are uniquely identified rows, but have no intrinsic properties of transactions. The inclusivity of such "features" would hurt many models, especially tree-based models that try to find patterns from features. It would allow the model to memorize unique identifiers instead of learning general relationships.

I then defined the target variable and the design matrix (feature matrix). They target, y, is the binary label for **is_fraud**. The feature matrix, X, consists of the remaining columns after we dropped the identifiers and the label. I split the data into training and testing sets, using an 80/20 train-test split and stratification on **is_fraud**. What this means is that the split preserves the original fraud rate in both sets, which is very crucial, given our target label is highly imbalanced. Otherwise, the minority class would be extremely underrepresented in testing.

To prepare the data for the different models, I separated feature columns into numerical and categorical. Numeric features will also include the hour and dayofweek columns that were created from the original **transaction_time**. The separation of both types of data allowed me to use **ColumnTransformer**-based preprocessing for my pipeline. Numerical features are standardized using **StandardScaler**, and categorical features are transformed with **OneHotEncoder**, where we used the parameter **handle_unknown= "ignore"** to safely handle unseen categories in the data. The transformer is made to drop any columns not stated explicitly, the ensure that we only use the chosen feature sets we made for the models.

The preprocessing pipeline is wrapped inside a scikit-learn **Pipeline** for each classifier to be used later in the project. The design ensures that the exact same preprocessing steps are applied consistently during training, cross-validation, and our final evaluations on the test set.

Machine Learning Algorithms:

To model fraud detection, I will implement and compare two supervised learning algorithms: Logistic Regression and Random Forest Classifier. Both models will be trained on the same data that I will give from the preprocessing pipeline. Using these two models will enable me to assess the tradeoff between a simple, linear, and interpretable model and a complex, non-linear ensemble model.

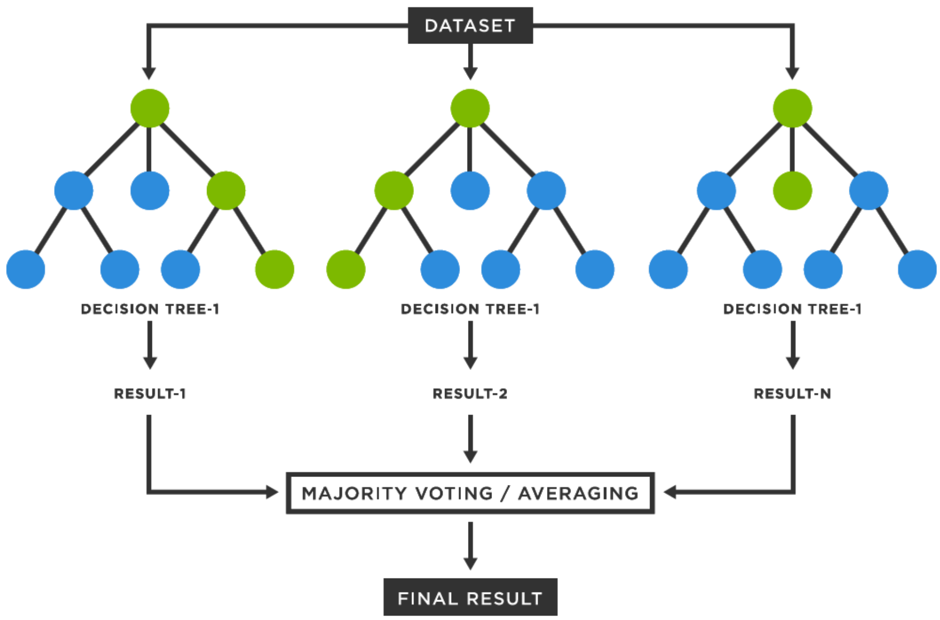
1. Logistic regression - A common baseline model because of its simplicity and interpretability. It models the log-odds of the positive class as a linear combination of our input features. This means given an input feature x, it will estimate its probability of being y=1, being a fraudulent transaction, as:

$$P(y = 1 \mid x) = \sigma(wTx + b)$$

Where w is a vector for learned weights, b is the bias term, and sigma is the sigmoid function. The parameters are learned by maximizing the likelihood of the training data. Logistic regression serves as a baseline model, being trained on standardized numeric features and one-hot encoded categorical features from the preprocessing pipeline. Since the data is highly imbalanced, I enabled class weighting so that fraud receives more weight in training. This helps the model pay more attention to the minority class and improve recall compared to the unweighted version. Overall, logistic regression provides a fast and simple baseline for evaluating the benefits of complex models later on.

2. Random Forest Classifier - This algorithm is an ensemble method based on decision trees. It builds many individual trees on different bootstraped samples of the training data and combines the predictions by majority vote. At each split,

the algorithm only considers a random subset of features. The randomness of the model is what helps reduce variance, making it more robust than just a single decision tree. They are well-suited to fraud detection. First, they capture non-linear relationships and complex relations between features without requiring explicit feature engineering. Second, they are quite insensitive to feature scaling, but I will still use the same preprocessing pipeline for consistency across models. Third, random forests measure feature importance, which makes it possible to identify which variables contribute most to the predictions. In this implementation, the random forest classifier is also made with class weighting to address my class imbalance with **is_fraud**. Increasing decision trees and setting a fixed random seed will help stabilize the performance of the model.

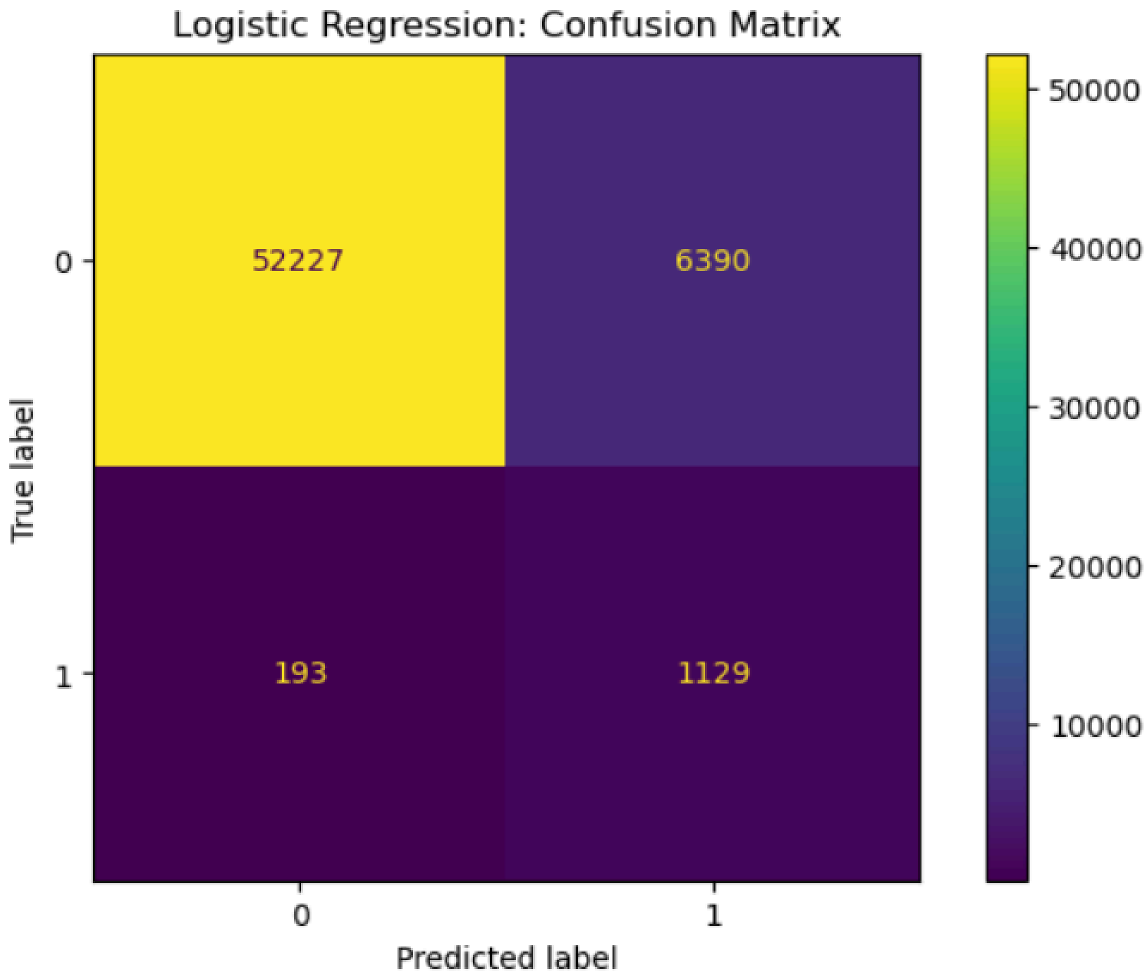


Modeling/Results:

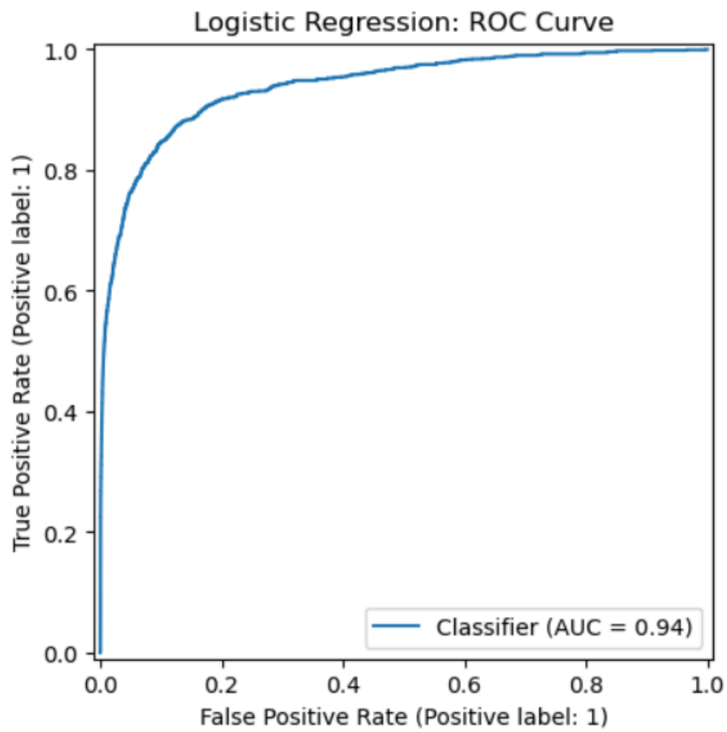
Logistic Regression:

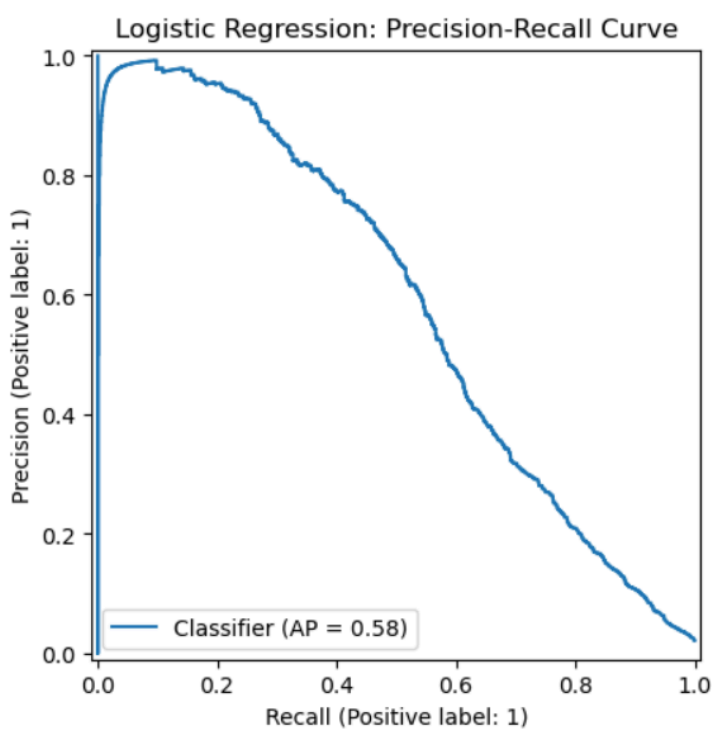
- Accuracy: 89%
- Precision: 0.15
- Recall: 0.85
- F1-Score: 0.26
- ROC-AUC: 0.94
- PR-AUC: 0.58

These numbers mean the model is very aggressive at catching fraud, as it correctly identifies 85% of all fraud transactions, seen by its high recall. But, it has a lot of false alarms, shown by its low precision of 15% of transactions flagged as fraud are actually fraud.



Given the confusion matrix for this model, 52,227 are true negatives and 1,129 are true positives. 6,390 are false positives (legit transactions incorrectly flagged as fraud) and 193 are false negatives (fraudulent transactions missed by the model). The low number of false negatives compared to the true positives is what is causing high recall.



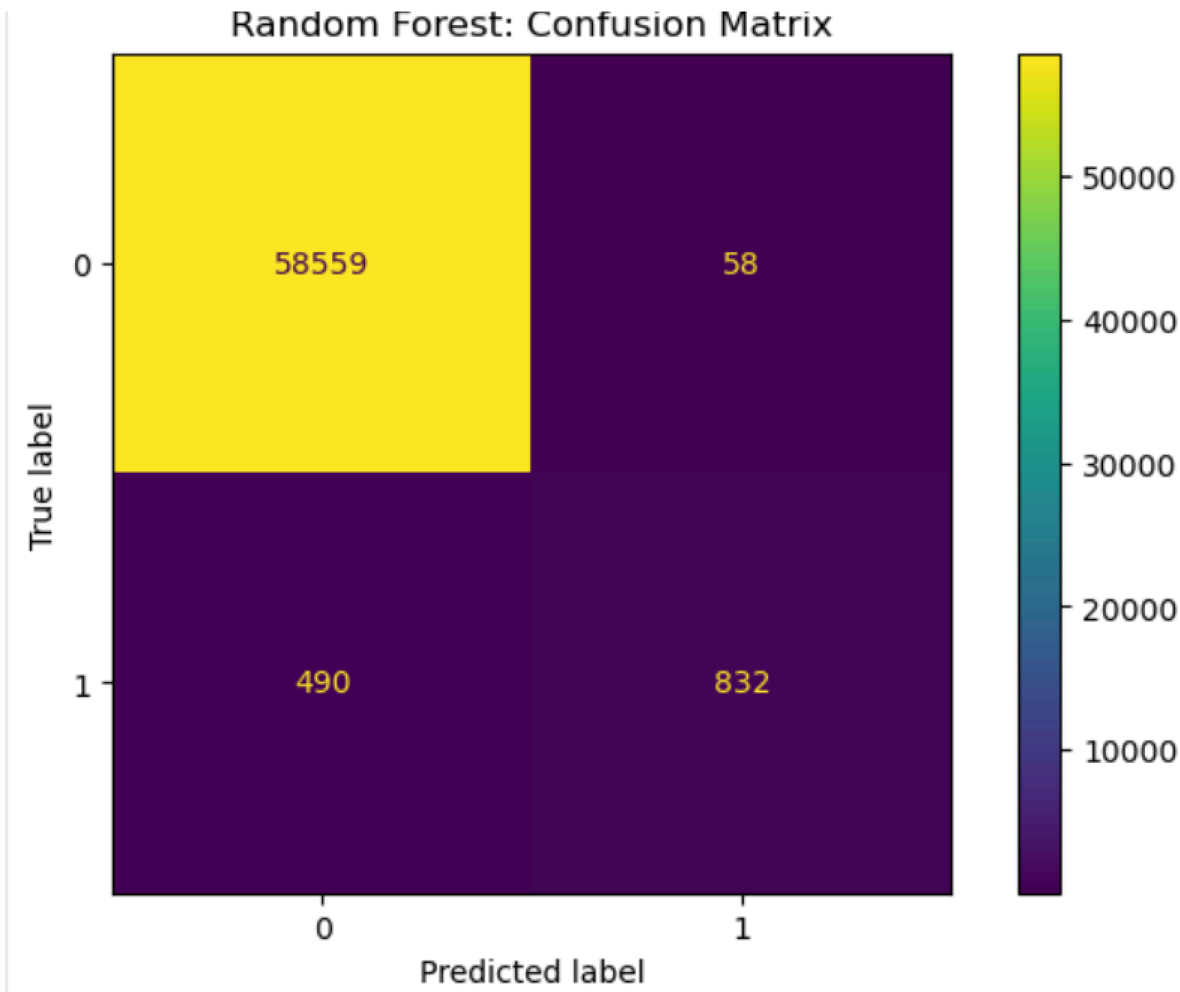


The ROC cruve has an AUC of about 0.94, which shows strong ability to rank points across its thresholds. The precision-recall curve shows an average precision of 0.58, and a drop in precision as the recall increases.

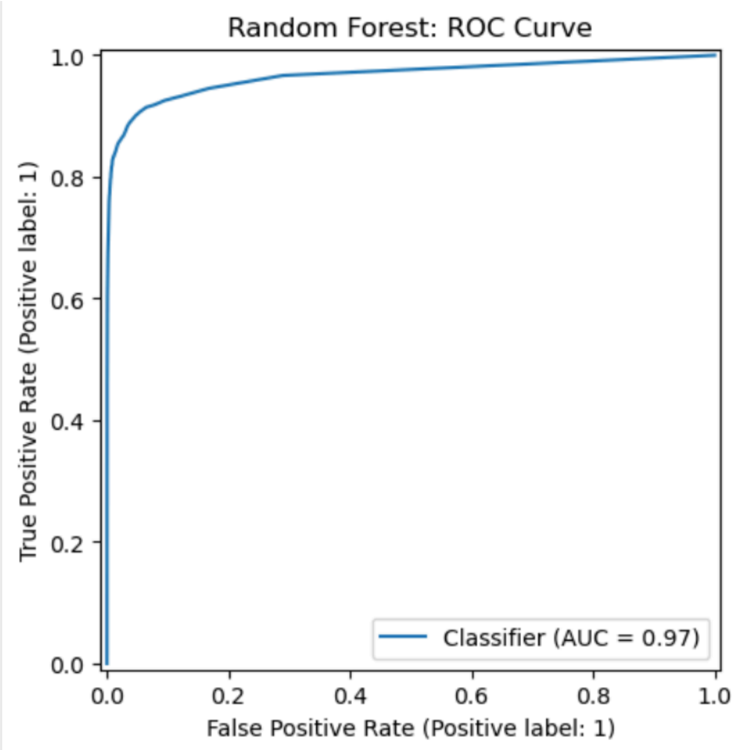
Random Forest Classifier:

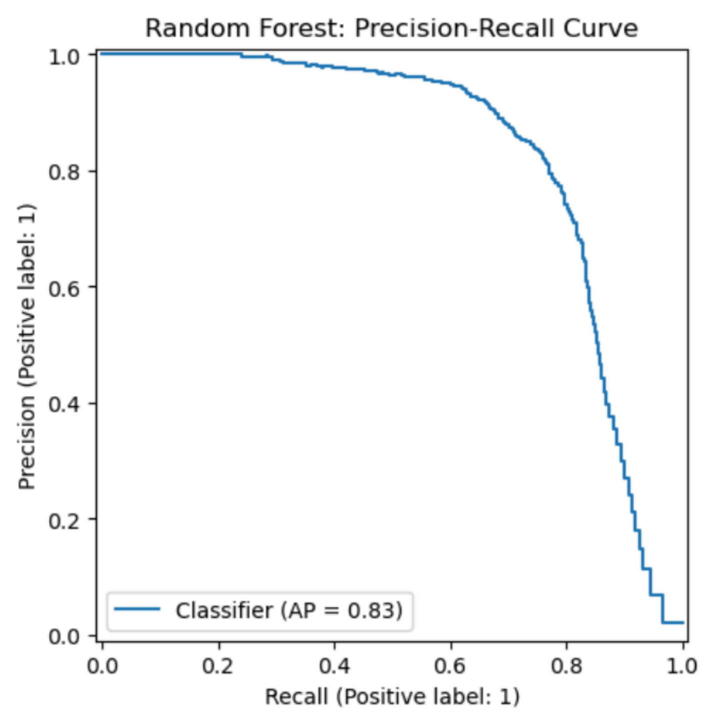
- Accuracy: 99%
- Precision: 0.93
- Recall: 0.63
- F1-Score: 0.75
- ROC-AUC: 0.97
- PR-AUC: 0.83

Compared to the logistic regression model, this random forest increases precision dramatically, from 0.15 to 0.93, but it still maintains a good recall score, 0.63. When my random forest predicts fraud, it is correct 93% of the time, and catches about 63% of all fraudulent transactions. The higher F1-Score shows this, as the model has an improved balance between precision and recall.



In this model, 58,559 are true negative, 832 are true positives, 85 are false positives, and 490 are false negatives. Random Forest reduces false positives from logistic's 6,390 to just 58. This is a huge improvement in terms of not falsely accusing legitimate customers. Our false negatives did increase though, from 193 to 490, and that is why recall is lower in logistic regression.





The ROC curve for random forest has an AUC of about 0.97, which is much higher than logistic regression, meaning it has a much better time ranking frad vs non-fraud transactions. The precision-recall curve has an average precision of .83, which improves on logistic's 0.58, and showing that the model can maintain high precision and recall.

Why this is not Overfitting:

Although random forest achieves an unlikely accuracy of 0.99, this is due to the strong class imbalance from the beginning. A naive model that always predicts 'not fraud' means it would be 98% accurate. Why, because the imbalance favors non-fraudulent the most, meaning if you guess a random example, you have a much better chance of getting it right in terms of non-fraud. The improvement from 0.98 to 0.99 is driven my reducing false positives, not by memorizing the tests data (overfitting). Our random forests ROC-AUC (0.97) and PR-AUC (0.83) means the model generalizes better than overfitting. This means even though an accuracy of 0.99 seems high and suspicious, it is not because of the imbalance and the small amounts of few positives on the test set.

Conclusion:

In this project, I built a credit card fraud detection using a large and highly imbalanced dataset. My analysis of the dataset confirmed that a small fraction of transactions are frauds and finding certain patterns, such as the recall and precision, could be useful signals for modeling. These patterns include higher fraud rates for specific channels and different times of day.

After this project, I have learned different lessons. In imbalanced problems, accuracy being used by itself is very misleading, as a naive model can overfit and still have good accuracy. You need to use different metrics: recall, precision, and F1-score, for example. I also saw how different algorithms can express trade-offs. Logistic regression was a high-recall model that behaved like a baseline model, as it was the most naive of the 2 models I trained. Random forest had a much more precise detection. This project also helped me gain hands on experience in designing a full on ML Pipeline, using tools like scikit-learn's pipelines and column transformers to keep my workflow clean.

Github Source Code:

Works Cited

"Random Forest Classification in Python With Scikit-Learn: Step-by-Step Guide (with Code Examples)." *DataCamp*, updated 31 Oct. 2025, www.datacamp.com/tutorial/random-forests-classifier-python.

Kavlakoglu, Eda. "What Is Random Forest?" *IBM*, www.ibm.com/think/topics/random-forest.

"Understanding Logistic Regression in Python." *DataCamp*, updated 11 Aug. 2024, www.datacamp.com/tutorial/understanding-logistic-regression-python.

Olamendy, Juan C. "Choosing the Right Metrics: Recall, Precision, PR Curve, and ROC Curve Explained." *Medium*, 2024, medium.com/@juanc.olamendy/choosing-the-right-metrics-recall-precision-pr-curve-and-roc-curve-explained-682259961cbe.

Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3rd ed., O'Reilly, 2022.

"RocCurveDisplay." *scikit-learn 1.7.2 Documentation*, scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html.

Brownlee, Jason. "**Overfitting and Underfitting With Machine Learning Algorithms.**" *Machine Learning Mastery*, 10 Apr. 2019, machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/.

Acknowledgement

I, Agampreet Bajaj, acknowledge that I used documentations for the libraries in Python to do this assignment. I also used many articles and sites to understand the algorithms and models fo the project.

I also acknowledge the use of AI in completing this assignment and would like to provide an explanation of how I used ChatGPT as a tool. I needed help creating my helper function as I did not know how to use the plotting tools for my algorithms in Python.

Prompt: Can you give me an explanation on why you used RocCurvDisplay and PrecisionRecallDisplay and what it does

Answer:
Nice question, this is exactly the kind of thing profs love seeing you understand.
Here's a clear explanation you can use in your report (and for yourself):
Why I used RocCurveDisplay
What it does:
RocCurveDisplay in scikit-learn is a helper class that plots the ROC curve (Receiver Operating Characteristic curve) for a classifier. The ROC curve shows:

- x-axis: False Positive Rate (FPR)
- y-axis: True Positive Rate (TPR) = Recall

Each point on the curve corresponds to a different decision threshold on the predicted probability of fraud (for example, flag as fraud if probability > 0.5, or 0.3, etc.).
Why it's useful in this project:

- It tells you how well the model separates fraud vs non-fraud across all possible thresholds, not just at 0.5.
- The ROC-AUC (area under the ROC curve) is a single number summary:
    - 0.5 ≈ random guessing
    - 1.0 = perfect ranking

Why I used PrecisionRecallDisplay
What it does:
PrecisionRecallDisplay plots the Precision–Recall (PR) curve:

- x-axis: Recall (for fraud)
- y-axis: Precision (for fraud)

Again, each point is a different threshold. It also lets you compute Average Precision (PR-AUC).

Why it's especially important here:

- Our data is highly imbalanced (fraud is ~2%). In this situation, the ROC curve can look very good even when the model isn't that practical.
- The PR curve focuses directly on the positive (fraud) class:
  - Recall: "Of all actual frauds, how many did we catch?"
  - Precision: "Of all transactions we flagged as fraud, how many were truly fraud?"
- PR-AUC is much more sensitive to how well the model performs on rare fraud cases.