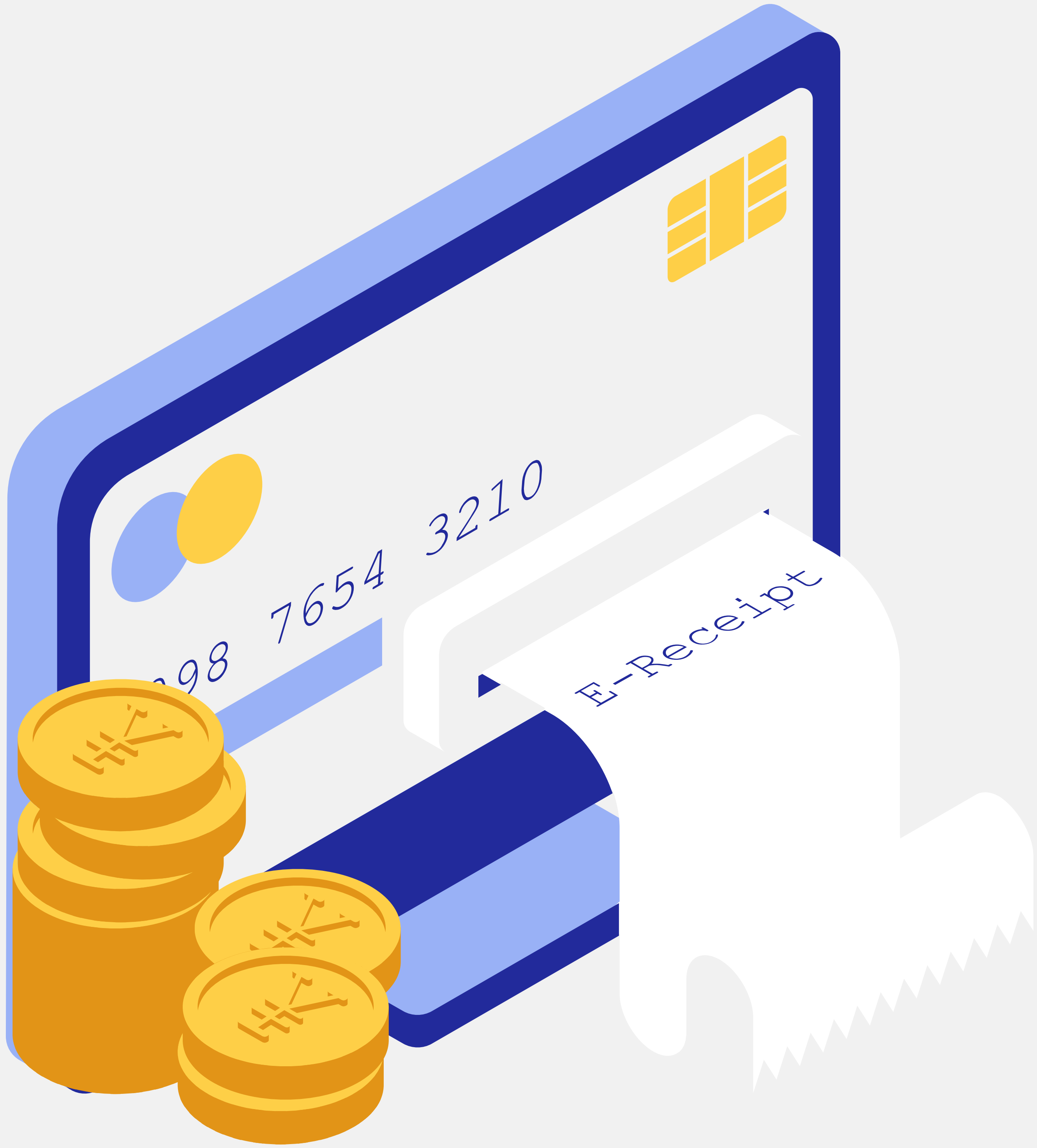



Final Project

# Detecting Credit Card Fraud using Machine Learning

By Veehjay and Sarp

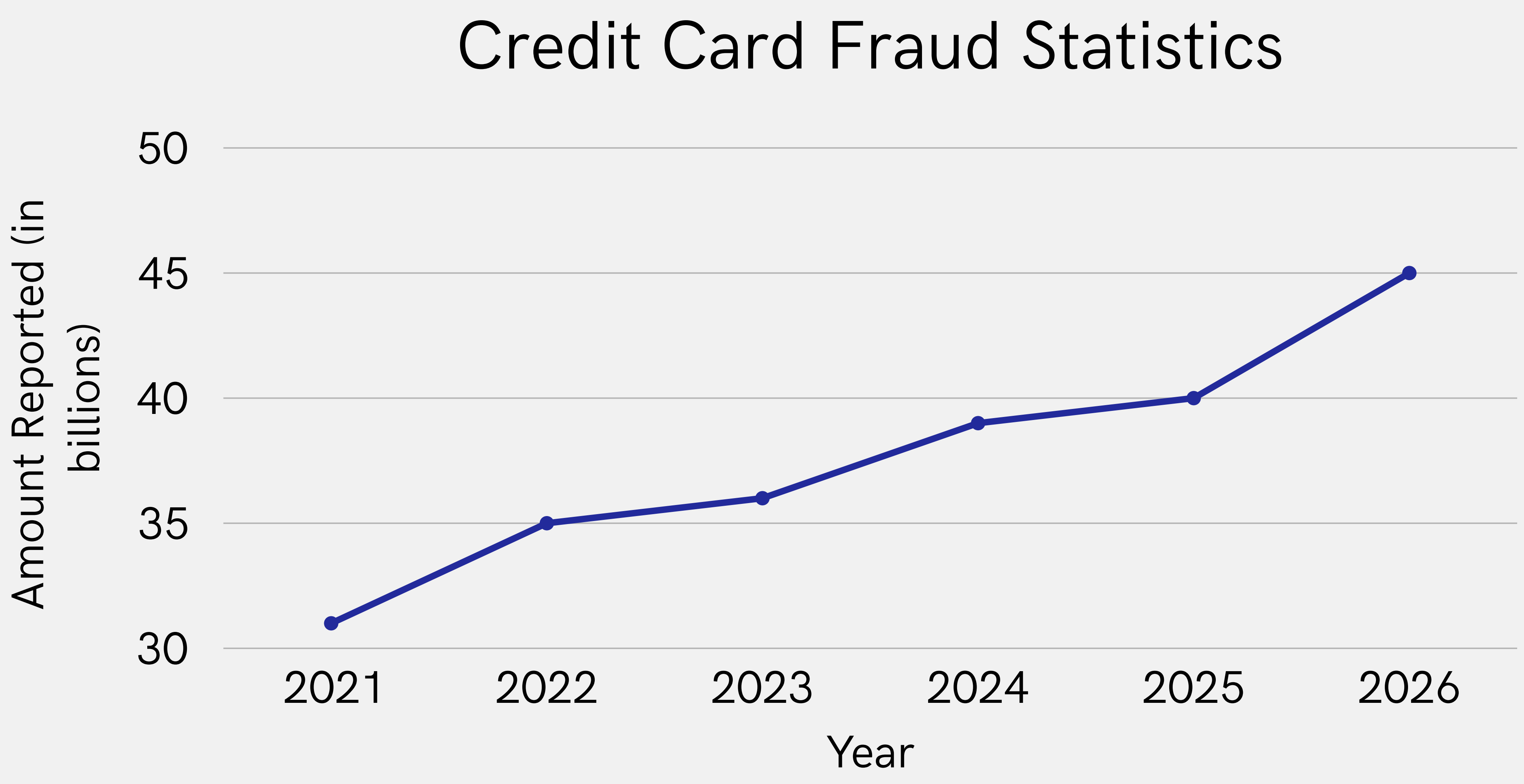
Project introduction • Fixing our data • Code Walkthrough / Model Results • Result Graphs • Conclusion

# 1. Explaining our Project

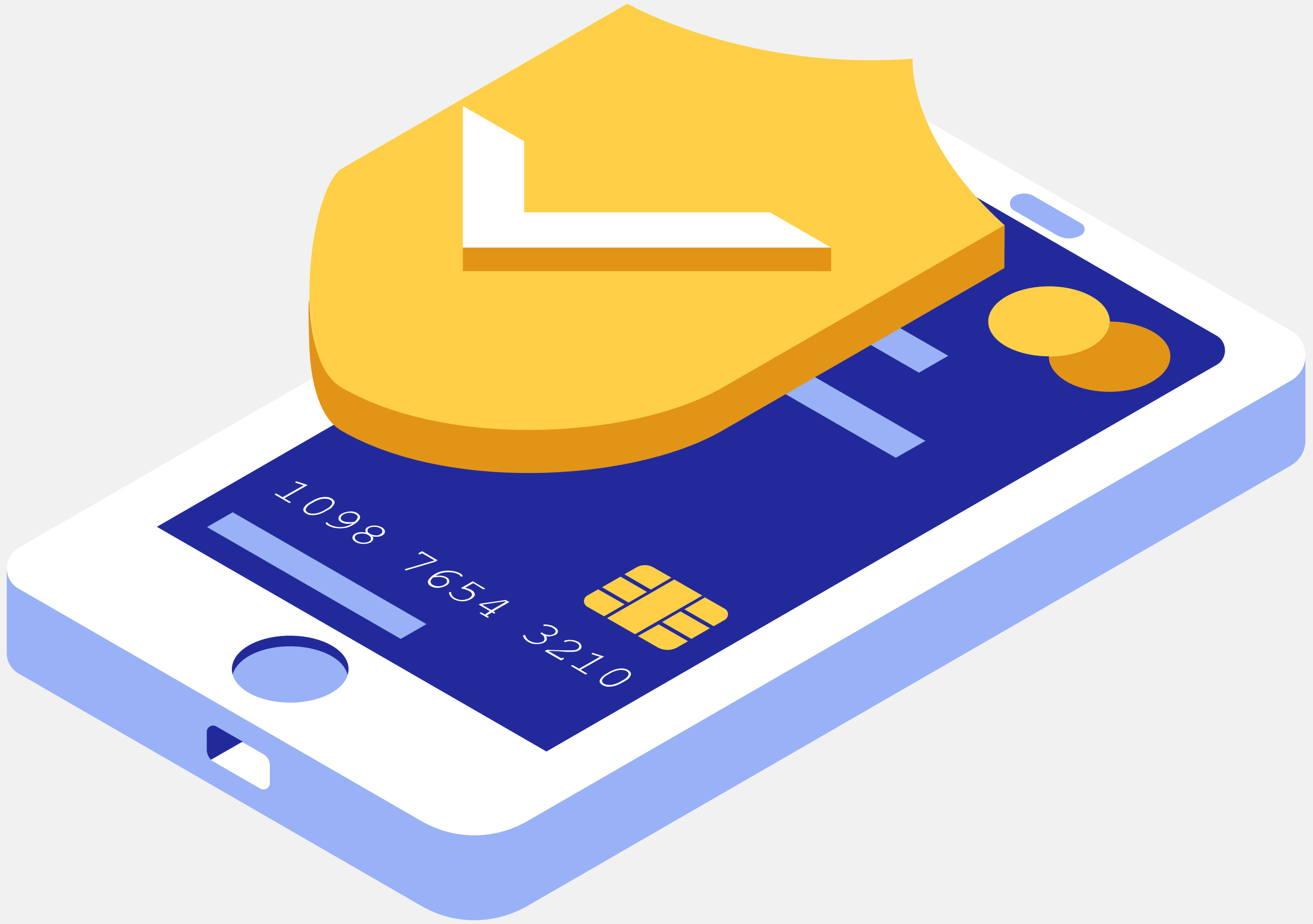
Why did we decide to do this topic? • Explaining the data

# NYC's Credit Fraud Problem

- In 2020, there was a record number of 67,000 identity fraud and credit card fraud cases reported in New York.
- This was an 85% increase from the amount of reported cases in 2019.
- Statistics show that the amount of money stolen will continue to increase per year




Statistics by Merchant Cost Consulting



## Where we collected our data

- All data comes from a large CSV file provided by Kaggle
- CSV contains **280,000** different transactions from September 2013 in Europe.

**Credit Card Fraud Detection**

Anonymized credit card transactions labeled as fraudulent or genuine

k kaggle.com

## Explaining the CSV file

- All "V…" variables are the result of a PCA transformation to keep the confidentiality of the cards.
- Time: The amount of time passed from the transaction compared to the first transaction in the CSV file.
- Amount: The amount of money spent with the transaction
- Class:
  - 1 → The transaction was **fraudulent**
  - 0 → The transaction was **real**

```
"Time","V1","V2","V3","V4","V5","V6","V7","V8","V9","V10","V11","V12","V13","V14","V15","V16","V17","V18","V19","V20","V21","V22","V23","V24","V25","V26","V27","V28","Amount","Class"
0,-1.3598071336738,-0.0727811733098497,2.53634673796914,1.37815522427443,-0.338320769942518,0.462387777762292,0.239598554061257,0.0986979012610507,0.363786969611213,0.0907941719789316
0,1.19185711131486,0.26615071205963,0.16648011335321,0.448154078460911,0.0600176492822243,-0.0823608088155687,-0.0788029833323113,0.0851016549148104,-0.255425128109186,-0.16697441400
1,-1.35835406159823,-1.34016307473609,1.77320934263119,0.379779593034328,-0.503198133318193,1.80049938079263,0.791460956450422,0.247675786588991,-1.51465432260583,0.207642865216696,0
1,-0.966271711572087,-0.185226008082898,1.79299333957872,-0.863291275036453,-0.0103088796030823,1.24720316752486,0.23760893977178,0.377435874652262,-1.38702406270197,-0.0549519224713
```

# 2. Fixing our data

- The original CSV provided only had 500 out of the 300,000 transactions be fraudulent
- Only 0.172% of cases were fraudulent, and attempting to train any models with this data would return incredibly low accuracy, precision and recall scores.

```
'class_weight': [{0:1, 1:10}, 'balanced']
```

**‘class_weight’**:

- **Non-fraudulent (0) cases** were given a regular penalty amount of 1, meaning minimal penalties if misclassified
- **Fraud (1) cases** were given a weight of 10, which would penalize the model greatly if misclassified.

**‘balanced’**:

- **Non-fraud (0) cases** have a normal weight of 1
- **Fraud (1) cases** weights ≈ 500× higher to counter the imbalance of data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

**‘stratify=y’**:

- Without **stratify=y:** the train/test split is random, so a set might have *zero fraud cases* by chance.
- With **stratify=y:** keeps the same fraud ratio in train test guaranteeing that fraud cases show up.

# Checking for misleading metrics

```
dummy = DummyClassifier(strategy="constant", constant=0)
dummy.fit(X_train, y_train)
```

- Dummy Classifier predicts all data as not being fraudulent
- Proves that accuracy isn’t the most reliable way to check if the model is trained well or not. **F1 Scores** are much more reliable and important.

---

- Accuracy ≈ 99.83%
- Recall ≈ 00.00%
- Precision ≈ 00.00%

```
=== Dummy Classifier ===
Accuracy:  0.9983 (99.83%)
Precision: 0.0000 (0.00%)
Recall:    0.0000 (0.00%)
F1 Score:  0.0000 (0.00%)
```

# 3. Our Models / Code Walkthrough

We trained a:

- Random Forest Model
- Logistic Regression Model
- Voting Classifier Model (using the Logistic Regression and Random Forest Model)

Data was split with 80% Training and 20% Testing using train_test_split

**Used Libraries:** sklearn, numpy, pandas, GridSearchCV, matplotlib.pyplot

# Random Forest Model

## Usage process

- It is a classification model, which is the best use model for our final project and will return to us the best results for credit card fraud detection.
- Robust and learns/improves easily to noisy data, meaning that it'll be more accurate

```
model = RandomForestClassifier(
    bootstrap=True,
    ccp_alpha=0.0,
    class_weight={0: 1, 1: 10},
    criterion='gini',
    max_depth=15,
    max_features='sqrt',
    max_leaf_nodes=None,
    max_samples=None,
    min_impurity_decrease=0.0,
    min_samples_leaf=2,
    min_samples_split=5,
    min_weight_fraction_leaf=0.0,
    n_estimators=200,
    n_jobs=None,
    oob_score=False,
    random_state=42,
    verbose=0,
    warm_start=False
)
```

- We used **GridSearchCV** to find the best possible hyperparameters for the model to get the highest and best results.

## Results

**Accuracy** ≈ 99.96% (0.9996)

**Precision** ≈ 95.29% (0.9529)

**Recall** ≈ 82.65% (0.8265)

**F1 Score** ≈ 88.52% (0.8852)

```
=== Standalone Random Forest ===
Accuracy:   0.9996
Precision:  0.9529
Recall:     0.8265
F1 Score:   0.8852
```

# Logistic Regression Model

## Usage process

- It is a classification model, which is the best use model for our final project and will return to us the best results for credit card fraud detection.
- It is one of the best comparisons that we could get to our Random Forest Model, allowing us to compare the two results and see which model returns the best results.

```
log_reg = LogisticRegression(
    C=0.1,
    class_weight='balanced',
    dual=False,
    fit_intercept=True,
    intercept_scaling=1,
    l1_ratio=None,
    max_iter=500,
    multi_class='auto',
    n_jobs=None,
    penalty='l2',
    random_state=None,
    solver='liblinear',
    tol=0.0001,
    verbose=0,
    warm_start=False
)
```

- The Logistic Regression model will also be incredibly useful in getting our final results with the **Voting Classifier** model.
- Used GridSearchCV to optimize our hyperparameters for the best results.

## Results

**Accuracy** ≈ 97.56% (0.9756)

**Precision** ≈ 0.061% (0.0611)

**Recall** ≈ 91.84% (0.9184)

**F1 Score** ≈ 11.5% (0.1146)

Hyper-parameters were set to get the best Recall Score, sacrificing precision and F1.

```
=== Standalone Logistic Regression ===
Accuracy:   0.9756
Precision:  0.0611
Recall:     0.9184
F1 Score:   0.1146
```

# Voting Classifier Model

## Usage process

- **Combines multiple models** (Logistic Regression + Random Forest) to make a final prediction.
- Averages the predicted probabilities and chooses the most confident model to fit with.
- Benefits of using the best of both models to get the highest testing results.

```
voting_clf = VotingClassifier(
    estimators=[
        ('logreg', log_reg_model),
        ('rf', rf_model)
    ],
    voting='soft',
    weights=[0.2, 0.8],
    n_jobs=-1
)
```

## Results

**Accuracy** ≈ 99.96% (0.9996)

**Precision** ≈ 95.24% (0.9524)

**Recall** ≈ 81.63% (0.8163)

**F1 Score** ≈ 87.91% (0.8791)

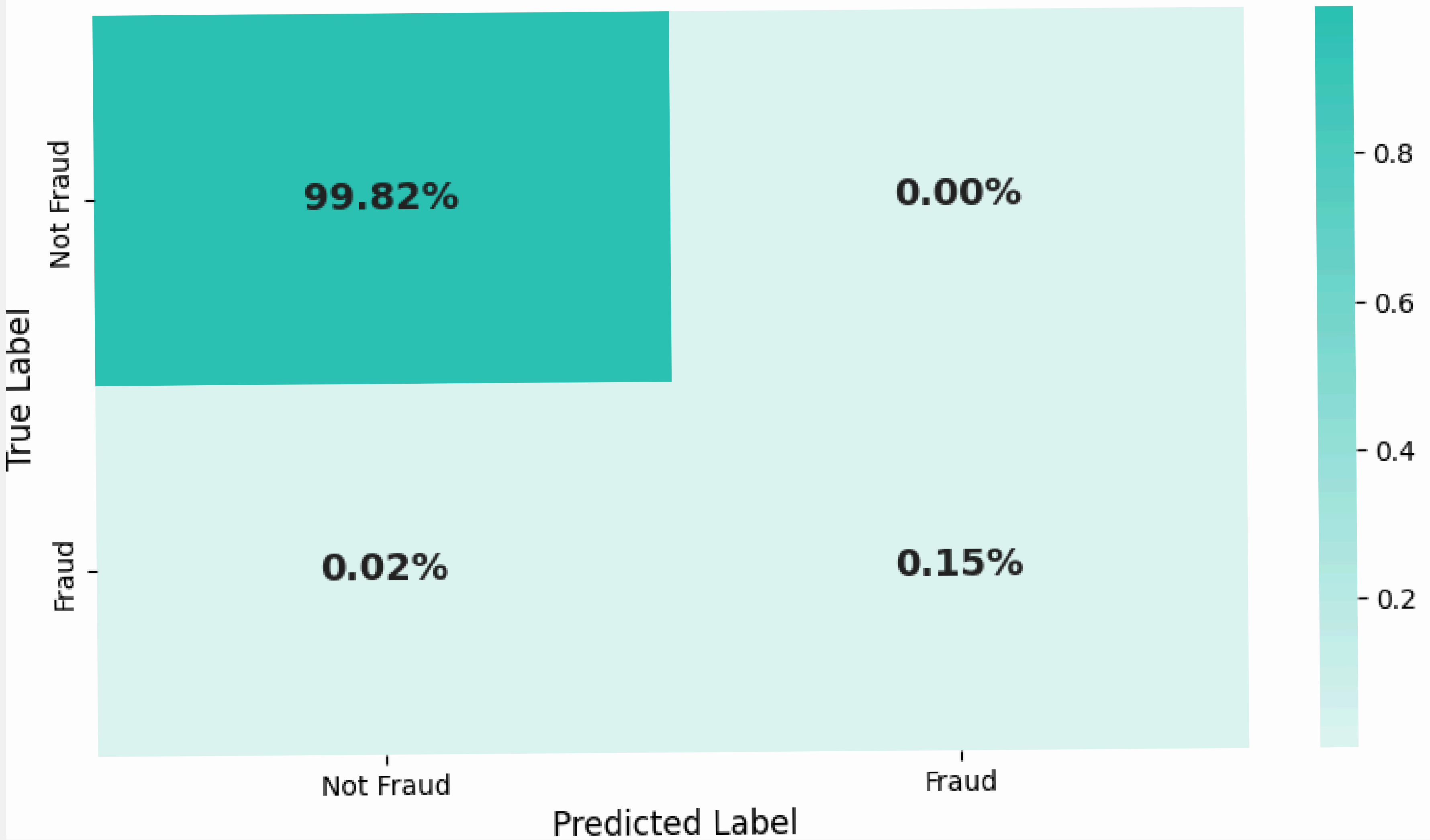
**these were our best results ever from all models!**

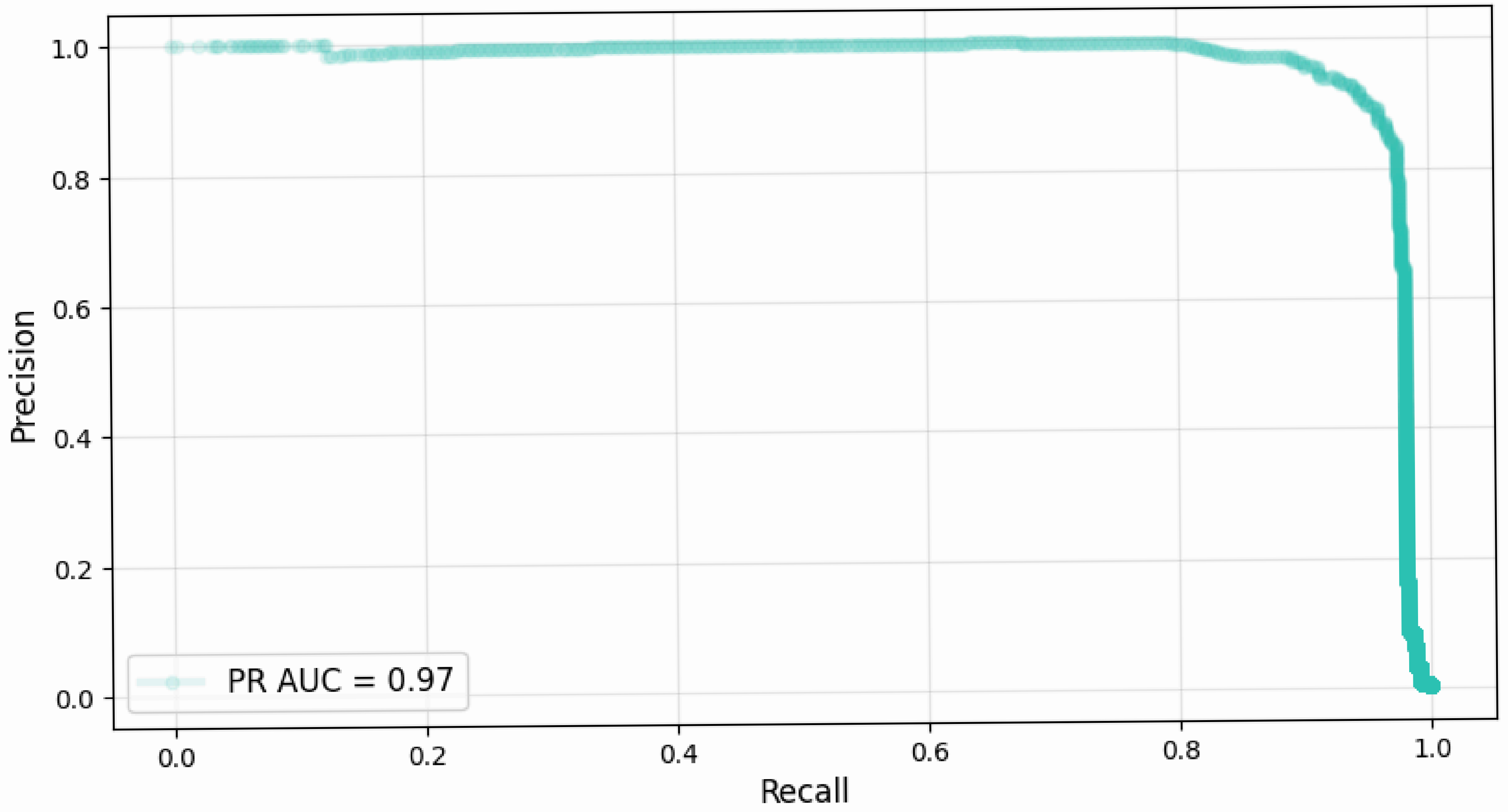
```
=== Voting Classifier (All Models, Soft Voting) ===
Accuracy:   0.9996
Precision:  0.9524
Recall:     0.8163
F1 Score:   0.8791
```

Metrics

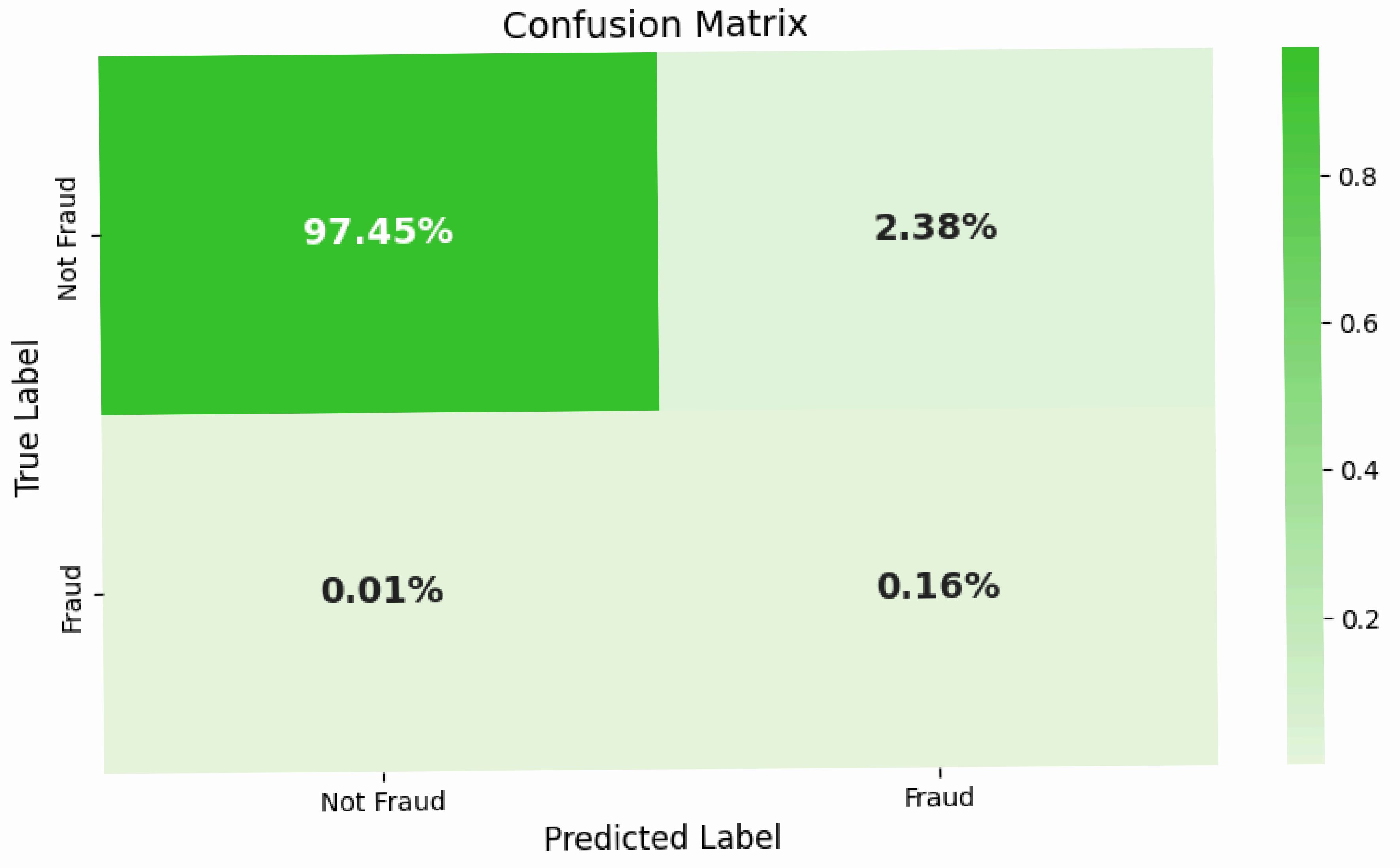
# 4. Result Graphs

# Random Forest Model

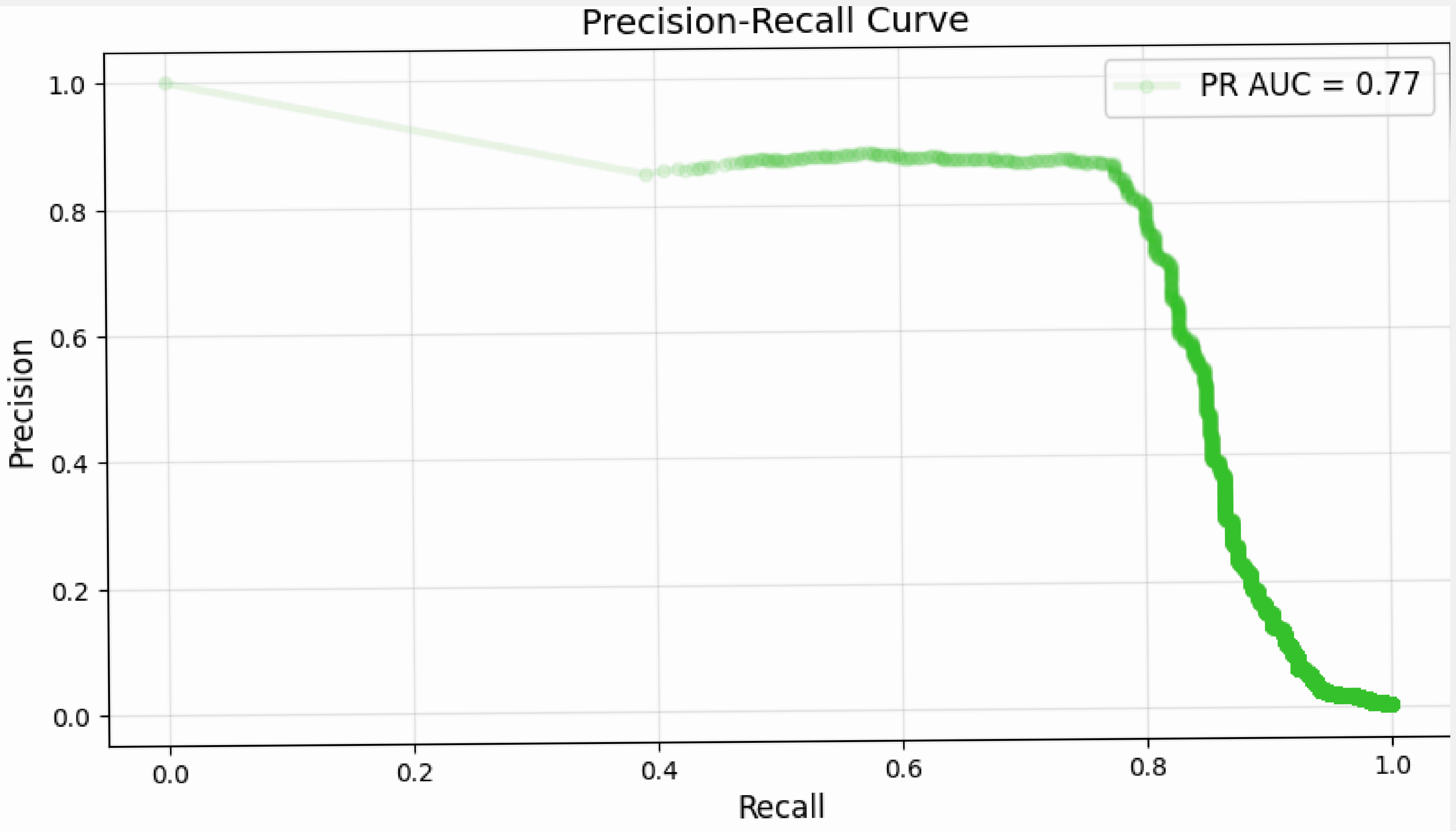


# Logistic Regression Model




Confusion Matrix




Precision-Recall Curve

# Voting Classifier Model