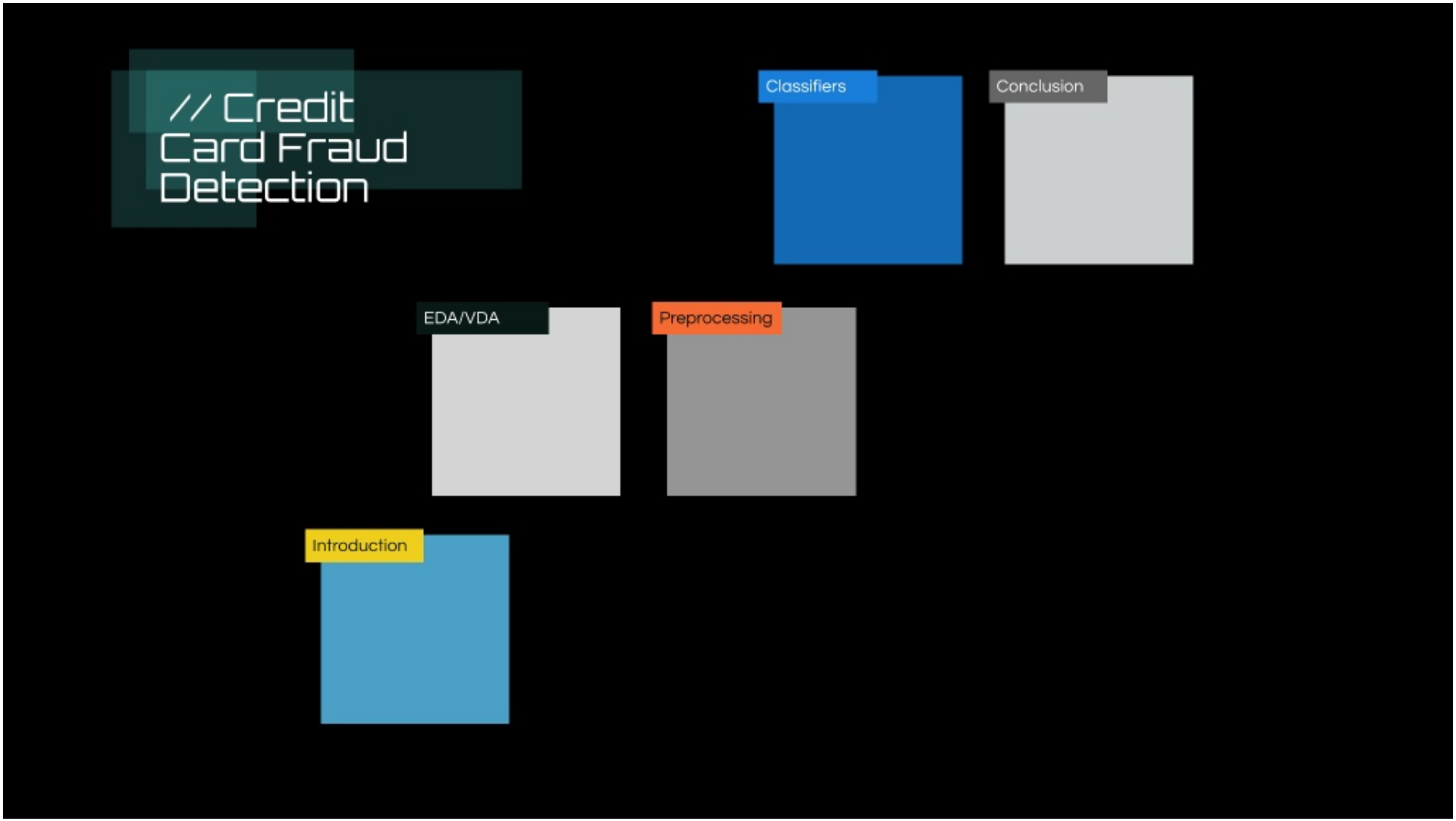

// Credit Card Fraud Detection
Classifiers
Conclusion
EDA/VDA
Preprocessing
Introduction

Introduction

This project is about the creation and evaluation of different classification models to identify whether the transaction is a normal or a fraudulent.

This data-set is published on Kaggle and it contains transactions made by credit cards in September 2013 by European cardholders. All the transactions in the data-set occurred in two days.

The features in the data-set are scaled and the names of the features are hidden due to privacy reasons.

Goals

Metrics

Goals

- Exploring the data-set.
- Dealing with imbalanced class distribution. (Under-sampling, Oversampling, Hybrid method)
- Creating and testing classification models
- Choosing the most accurate model.

Metrics

Accuracy score:

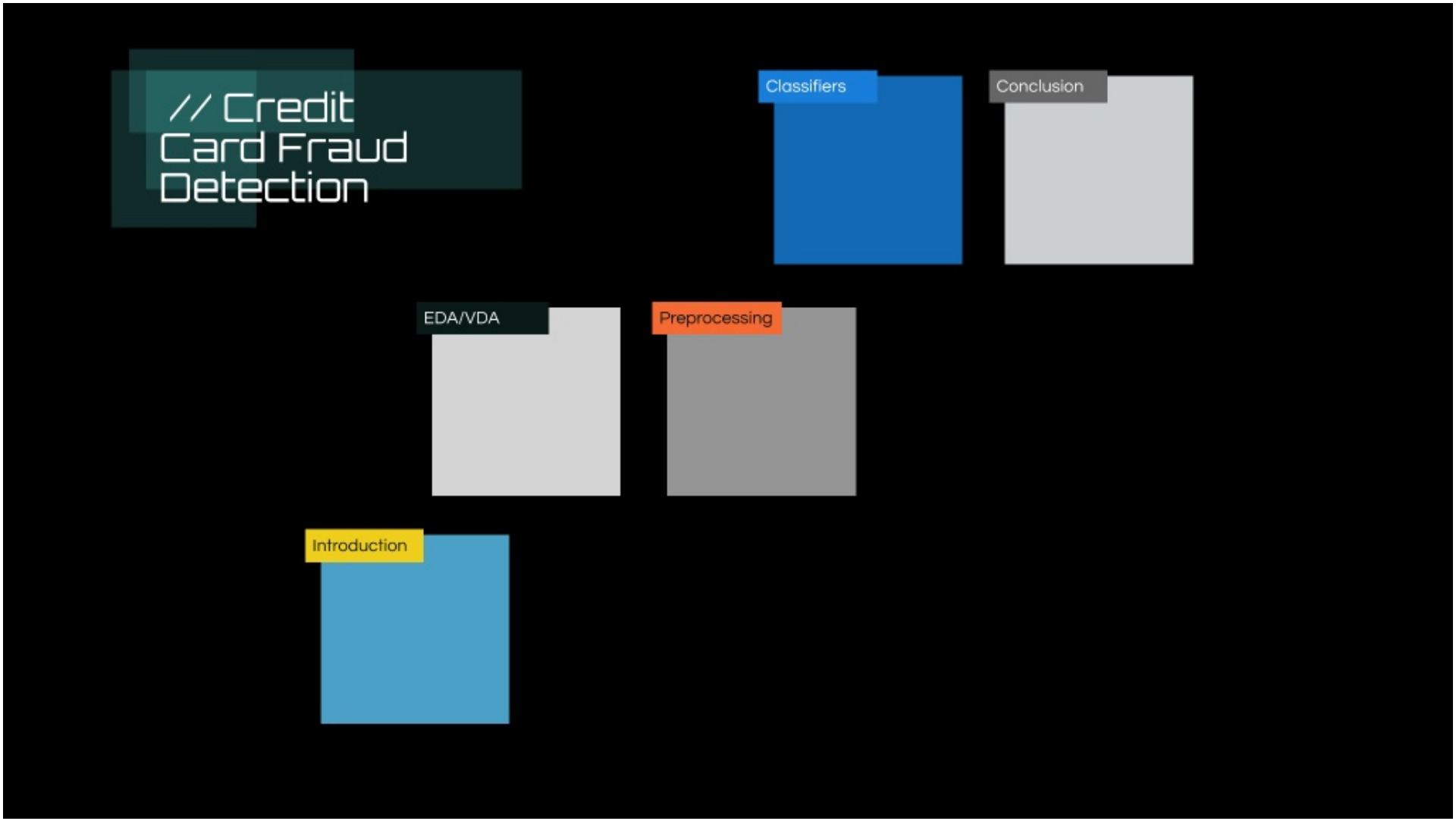
- When we use accuracy, we assign equal cost to false positives and false negatives. When that data set is imbalanced there is a great way to lower the cost. Models will predict that every instance belongs to the majority class, get accuracy of 99%. As we deal with a fraudulent transactions, the cost of failing to identify the fraud transaction is much higher than the cost of contacting customers to make sure that the transactions is not fraud.

ROC Curve:

With a large number of negative samples — precision is probably better. Precision is more focused in the positive class than in the negative class, it actually measures the probability of correct detection of positive values, while FPR and TPR (ROC metrics) measure the ability to distinguish between the classes.

Precision-Recall Curve:

When the majority is negative samples and not all positive samples are detected, the precision-recall will help to make a decision.


// Credit Card Fraud Detection
Classifiers
Conclusion
EDA/VDA
Preprocessing
Introduction

Explarotary Analysis

- The dataset has 31 columns and 284806 entries (transactions)
- The target variable is categorical
- There are no missing values
- 99.83% of transactions are **non-fraud**
- 0.17% of transactions are **fraud**

Note: except for the "Time" and "Amount" we dont know what the other columns are (due to privacy reasons).
We only know that the unknown columns have been scaled already.
"Time" columns represents the seconds elapsed since the first transaction.
"Amount" columns represents the amount of money being transacted

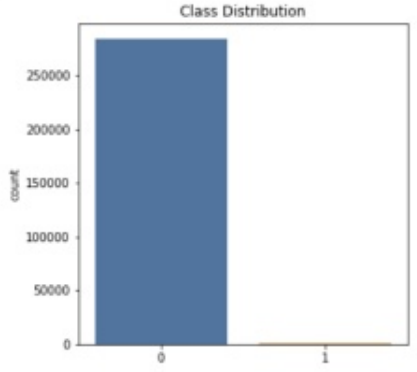
Class Imbalance

Correlation Matrix

Distributions

Class Imbalance

```
In [5]: # target variable distribution
        plt.figure(figsize = (5,5))
        _ = sns.barplot(x = df.Class.unique(), y = df.Class.value_counts())
        _.set(title = 'Class Distribution', ylabel = "count")
        plt.show()

        frauds = df.Class.value_counts(normalize = True).round(4)*100
        print(frauds)
```



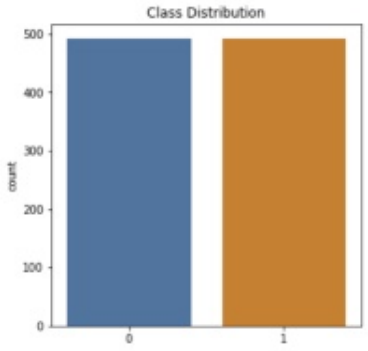

```
0    99.83
1     0.17
Name: Class, dtype: float64
```

Majority negative class

Manual Undersampling

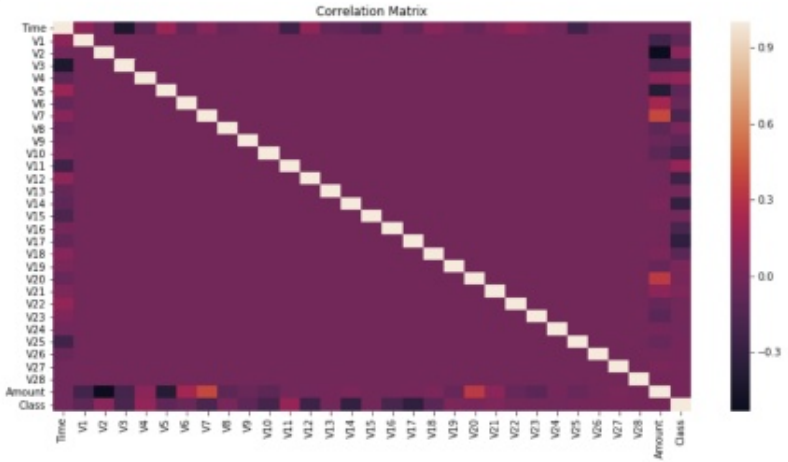
Manual Undersampling

In [7]:
```
# undersampling majority class manually
# shuffle the dataframe

df.sample(frac=1, random_state = 42)

X = df[df.Class == 1]
Y = df[df.Class == 0][:492] # we have 492 frauds out of 284,807 transactions

sampled_df = pd.concat([X, Y])
# check the class distribution
plt.figure(figsize = (5,5))
_ = sns.barplot(x = sampled_df.Class.unique(), y = sampled_df.Class.value_counts())
_.set(title = 'Class Distribution', ylabel = "count")
plt.show()
```



Correlation Matrix

```
In [6]: # correlation matrix
        plt.figure(figsize = (14,7))
        corr_mat = df.corr()
        _ = sns.heatmap(corr_mat)
        _.set(title = 'Correlation Matrix')
        plt.show()
```

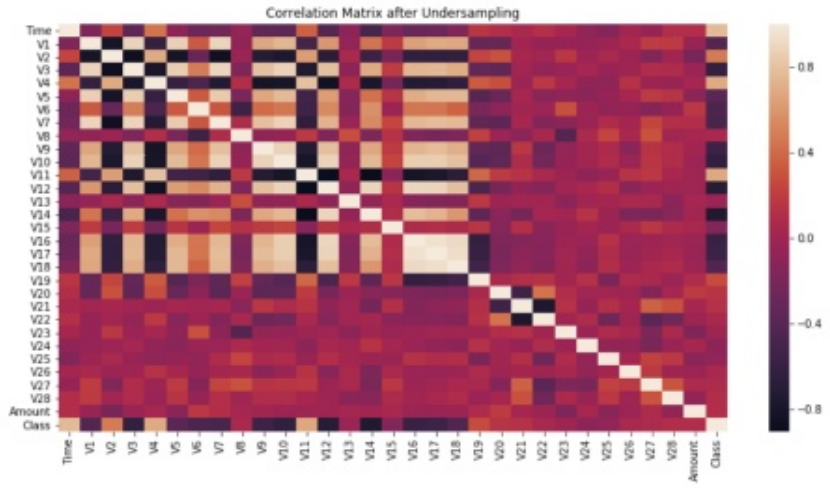


No observable correlation among features due to class imbalance

Manual Undersampling

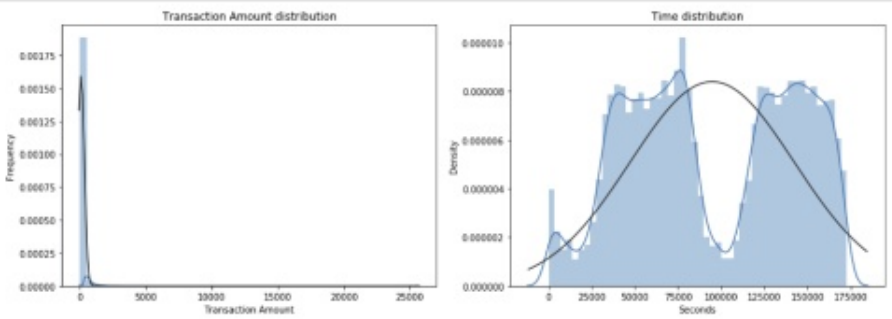
Correlation Matrix after Undersampling

```
In [8]: # correlation matrix
        plt.figure(figsize = (14,7))
        corr_mat = sampled_df.corr()
        _ = sns.heatmap(corr_mat)
        _.set(title = 'Correlation Matrix after Undersampling')
        plt.show()
```



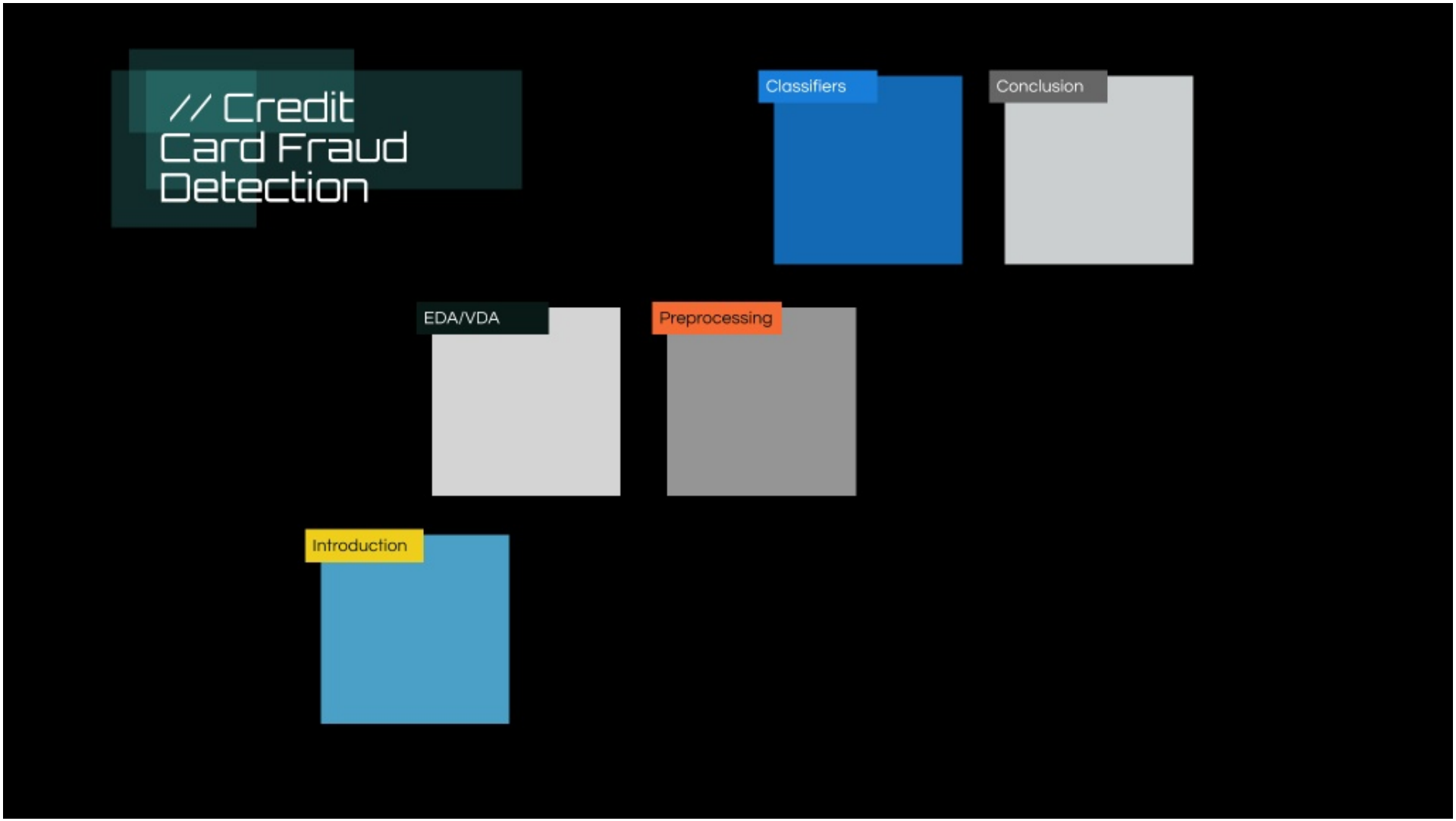
The higher the correlation between the features and class(either positive or negative), the higher the probability of fraud.

Distributions



By seeing the distributions we can have an idea how skewed are these features.

Both features have non-normal distributions. The transaction amount is heaviliy skewed towards small transactions and should be scaled.


// Credit Card Fraud Detection
Classifiers
Conclusion
EDA/VDA
Preprocessing
Introduction

Data preparation

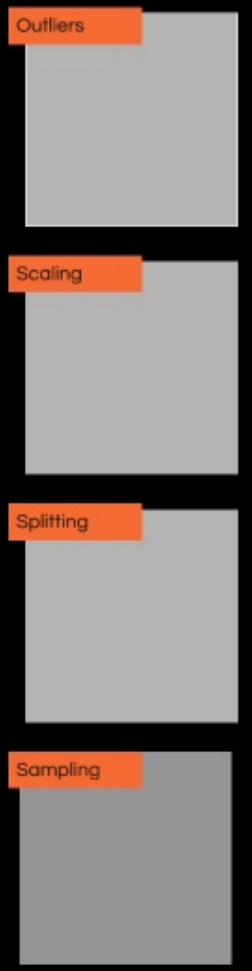
Defining Outliers:
Since we have 492 fraud transaction instances, better to hold each fraud case and remove the most of outliers from the non-fraudulent transactions.

The maximum fraud transaction amount is 2125.87. Therefore, by removing transaction amounts higher than 2150 we will clean the data from some of outliers.

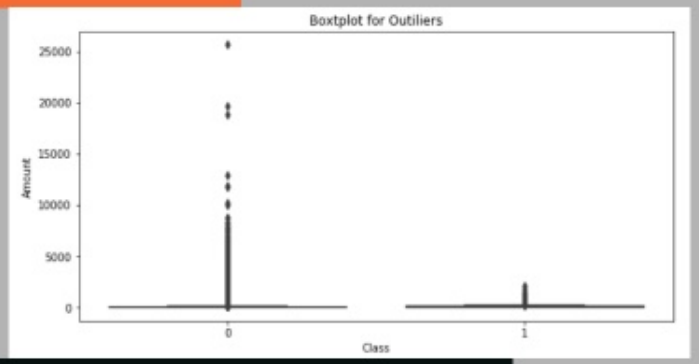
Scaling:
'Time' & 'Amount' columns well be scaled as the rest of the features.

Splitting:
Split dataset into train and test sets

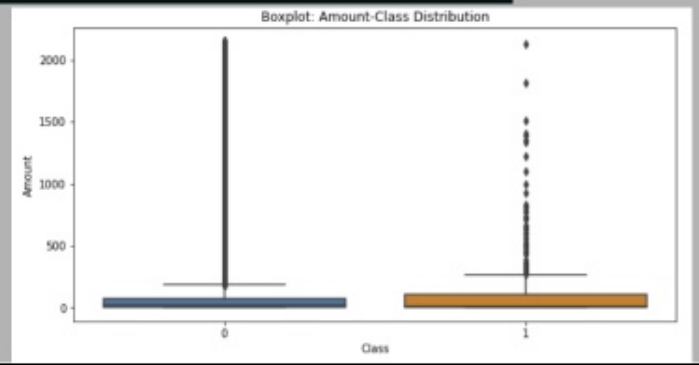
Sampling:
Undersampling or Hybrid Approach



Outliers



As the number of fraud transactions is very low, we will hold them even if there are some outliers.



Scaling

Scaling using sklearn StandardScaler()

In [51]:
```
# normalize Amount feature
scaler  = StandardScaler()
new_df['scaled_amount'] = scaler.fit_transform(new_df['Amount'].values.reshape(-1, 1))
new_df['scaled_time'] = scaler.fit_transform(new_df['Time'].values.reshape(-1,1))
new_df = new_df.drop(['Time', 'Amount'], axis = 1)
new_df.head(2)
```

Class	scaled_amount	scaled_time
0	0.386246	-1.996514
0	-0.444753	-1.996514

Splitting the data

```
X = new_df.drop('Class', axis = 1)
Y = new_df.Class

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
                                                    random_state = 42, stratify=new_df.Class)
```

After removing outliers and scaling the variables, I split the data into training and test subsets to fit and validate the models.

The "stratif" parameter makes a split so that the proportion of values in the sample produced will be the same as the proportion of values provided to parameter stratify.

Hybrid or Undersampling

Here we explore the various options to rebalance the training dataset. The methods can either:

- Over-sample the minority class
- Under-sample the majority class
- Combine over and under-sampling (Hybrid)
- Create ensemble balanced sets

Since the dataset is huge, oversampling the majority class will make it even larger and computationally costly

SMOTEENN sampling

RandomUnderSampler

SMOTEENN sampling

Imbalanced-learn has two hybrid modules :
SMOTEENN and SMOTETomek.

SMOTE ENN combines SMOTE (over sampling) and ENN or Edited Nearest Neighbours (Under Sampling) .

Since it tends to clean more noisy samples than SMOTETomek, it was my choice to create more accurate models.

```
# dealing with class imbalance in training set using SMOTEENN
smote_enn = SMOTEENN()
X_s, Y_s = smote_enn.fit_sample(X, Y)


print(X_train.shape)
print(X_test.shape)

count = np.bincount(Y_s)
print('number of 0\'s {} \nnumber of 1\'s {}'.format(
        count[0], count[1]))
```

```
(227375, 30)
(56844, 30)
number of 0's 264893
number of 1's 275269
```

Here, undersampling is applied to the original dataset rather training set.

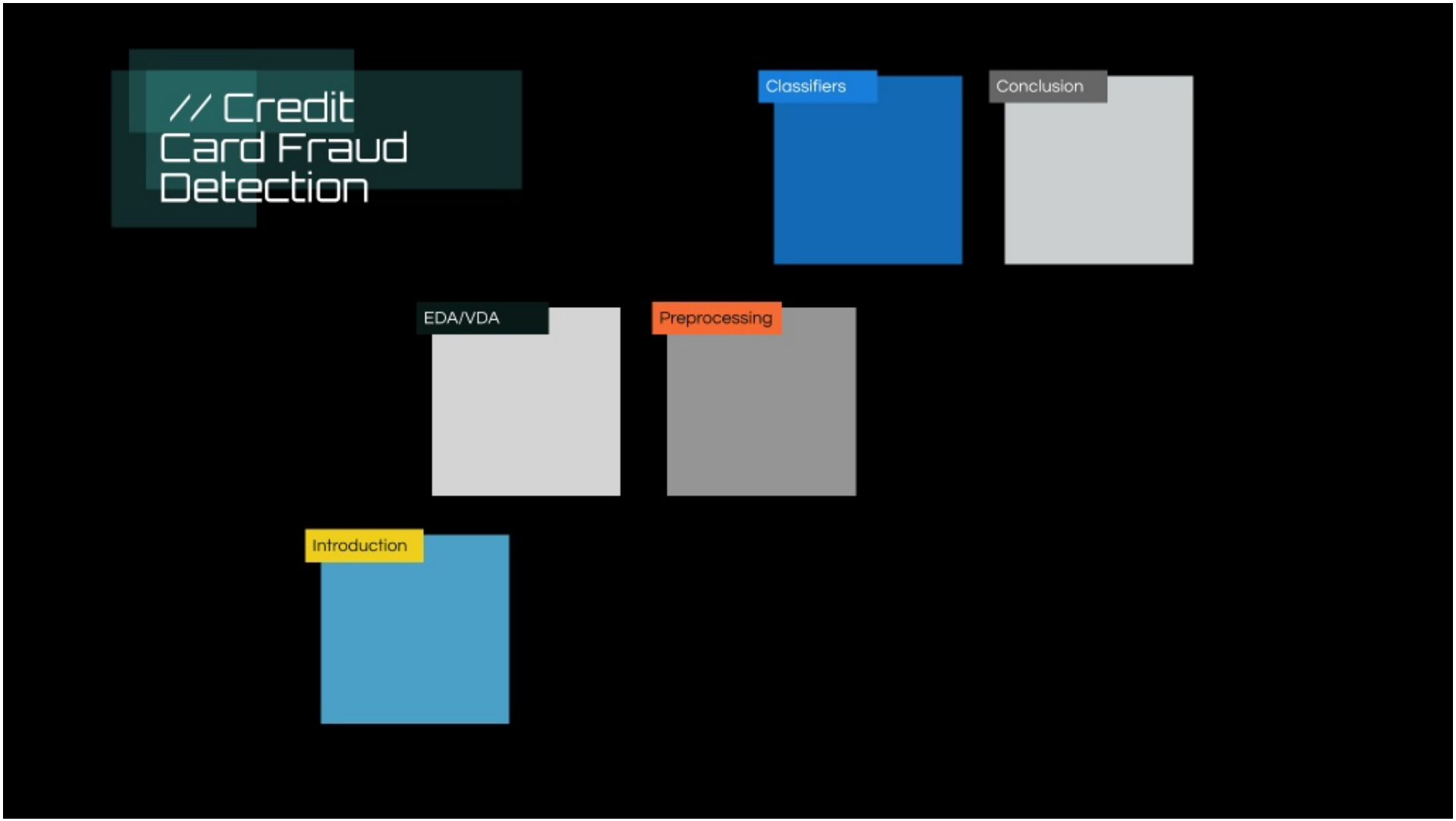
RandomUnderSampler

Though undersampling removes most of the data to reach the balance in class distribution, I compared these two sampling methods, and to my surprise I got more accurate results with RandomUnderSampler than SMOTEENN. And the running time of the models also decreased.

```
# dealing with class imbalance in training set using RandomUnderSampler
rus = RandomUnderSampler()
X_s, Y_s = rus.fit_sample(X, Y)


print(X_train.shape)
print(X_test.shape)

count = np.bincount(Y_s)
print('number of 0\'s {} \nnumber of 1\'s {}'.format(
        count[0], count[1]))
```

```
(227375, 30)
(56844, 30)
number of 0's 492
number of 1's 492
```

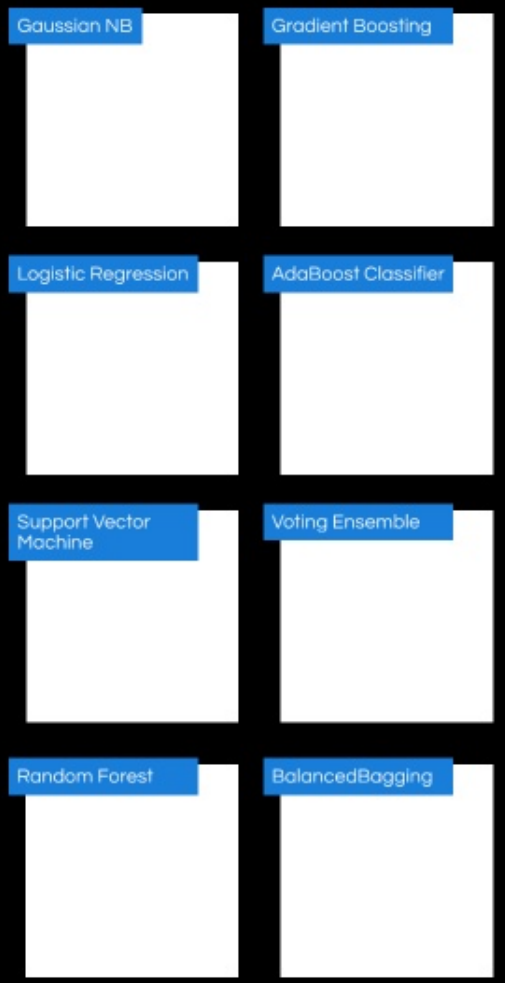
Here, undersampling is applied to the original dataset rather training set.


// Credit Card Fraud Detection
Classifiers
Conclusion
EDA/VDA
Preprocessing
Introduction

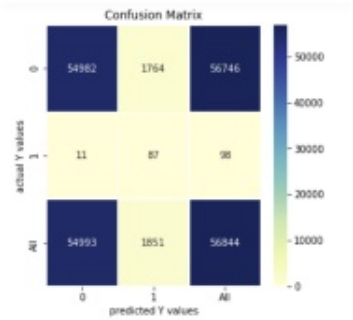
Classifiers

To predict the outcome variable I'll test 8 classification algorithms and compare their performance. For this specific problem (credit fraud) I want to minimize the recall. False Negative predictions mean that the model fails to identify True credit card frauds and labels them as non-fraud transactions.

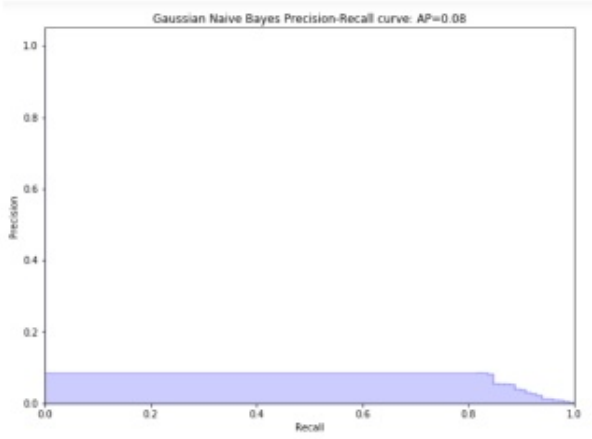
Metrics to focus on:
- ROC curve
- AUC score
- Recall score
- False Positive Rate
- Precision-Recall curve
- Average Precision (AP) score
- F1 score



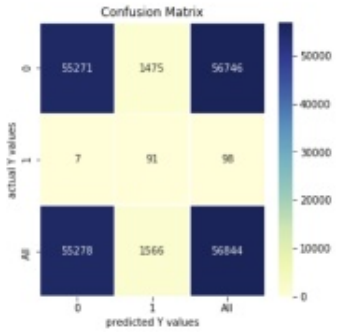
Gaussian Naive Bayes



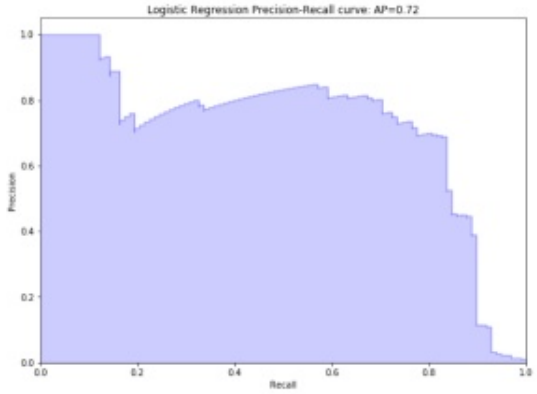
- Accuracy: 0.92 (std: 0.12)
- Recall: 0.89
- Precision: 0.05
- F1: 0.09
- AUC Score: 0.9725
- AP: 0.08



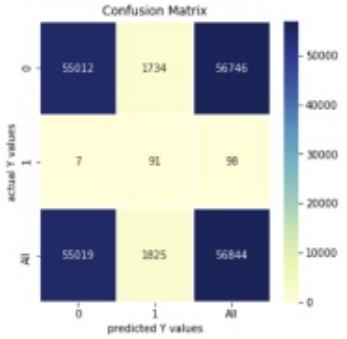
Logistic Regression (L2)



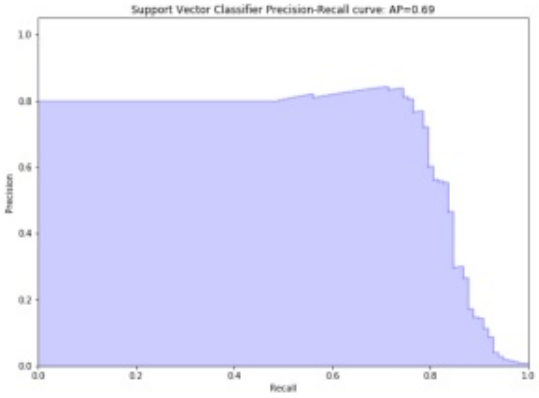
- Accuracy: 0.93 (std: 0.09)
- Recall: 0.93
- Precision: 0.06
- F1: 0.11
- AUC Score: 0.9924
- AP: 0.72



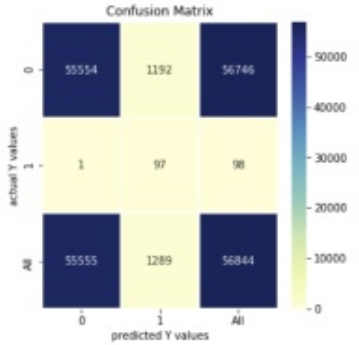
SVC



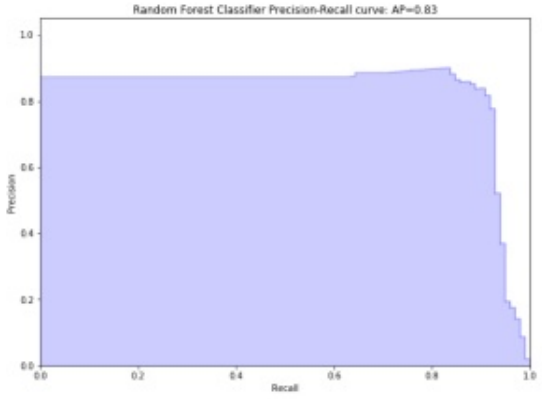
- Accuracy: 0.92 (std: 0.08)
- Recall: 0.93
- Precision: 0.05
- F1: 0.09
- AUC Score: 0.9905
- AP: 0.69



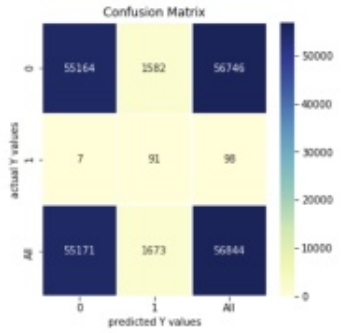
RandomForestClassifier



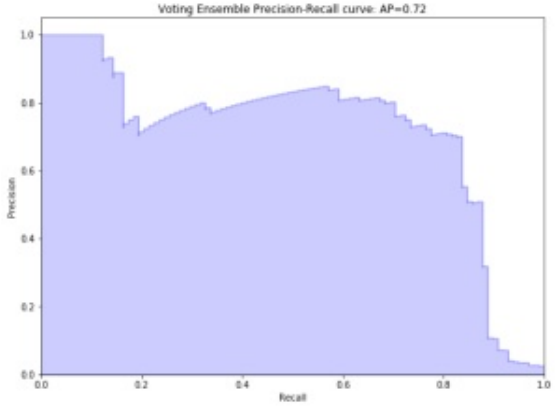
- Accuracy: 0.93 (std: 0.1)
- Recall: 0.99
- Precision: 0.08
- F1: 0.13
- AUC Score: 0.9986
- AP: 0.83



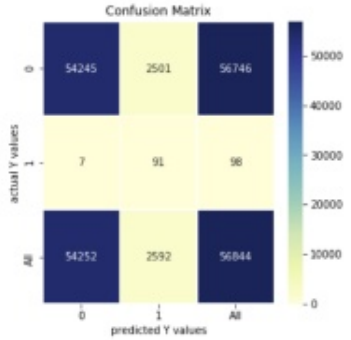
Voting Ensemble Classifier



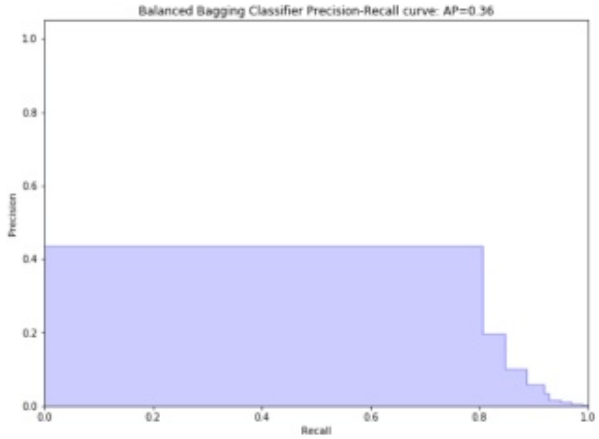
- Accuracy: 0.93 (std: 0.12)
- Recall: 0.93
- Precision: 0.05
- F1: 0.1
- AUC Score: 0.9953
- AP: 0.72



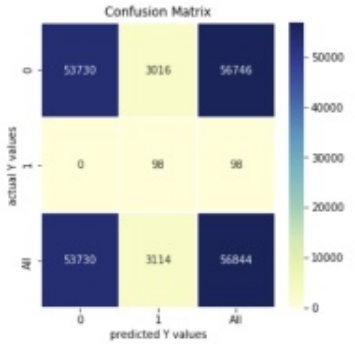
BalancedBaggingClassifier



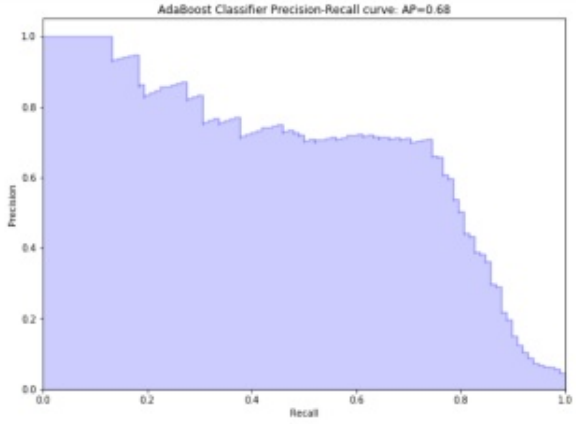
- Accuracy: 0.96 (std: 0.01)
- Recall: 0.93
- Precision: 0.04
- F1: 0.07
- AUC Score: 0.9791
- AP: 0.36



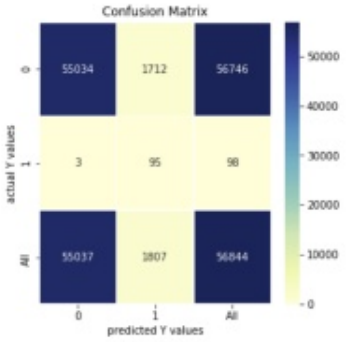
AdaBoost Classifier



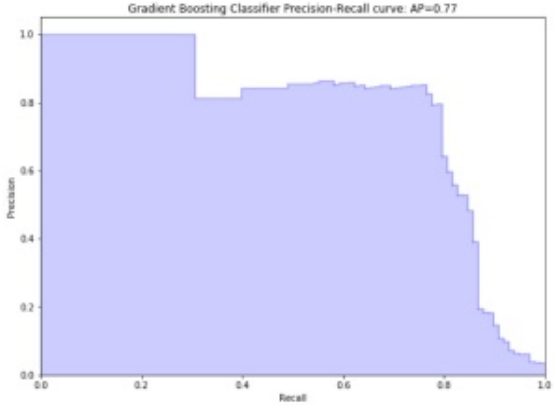
- Accuracy: 0.82 (std: 0.55)
- Recall: 1
- Precision: 0.03
- F1: 0.06
- AUC Score: 0.9973
- AP: 0.68

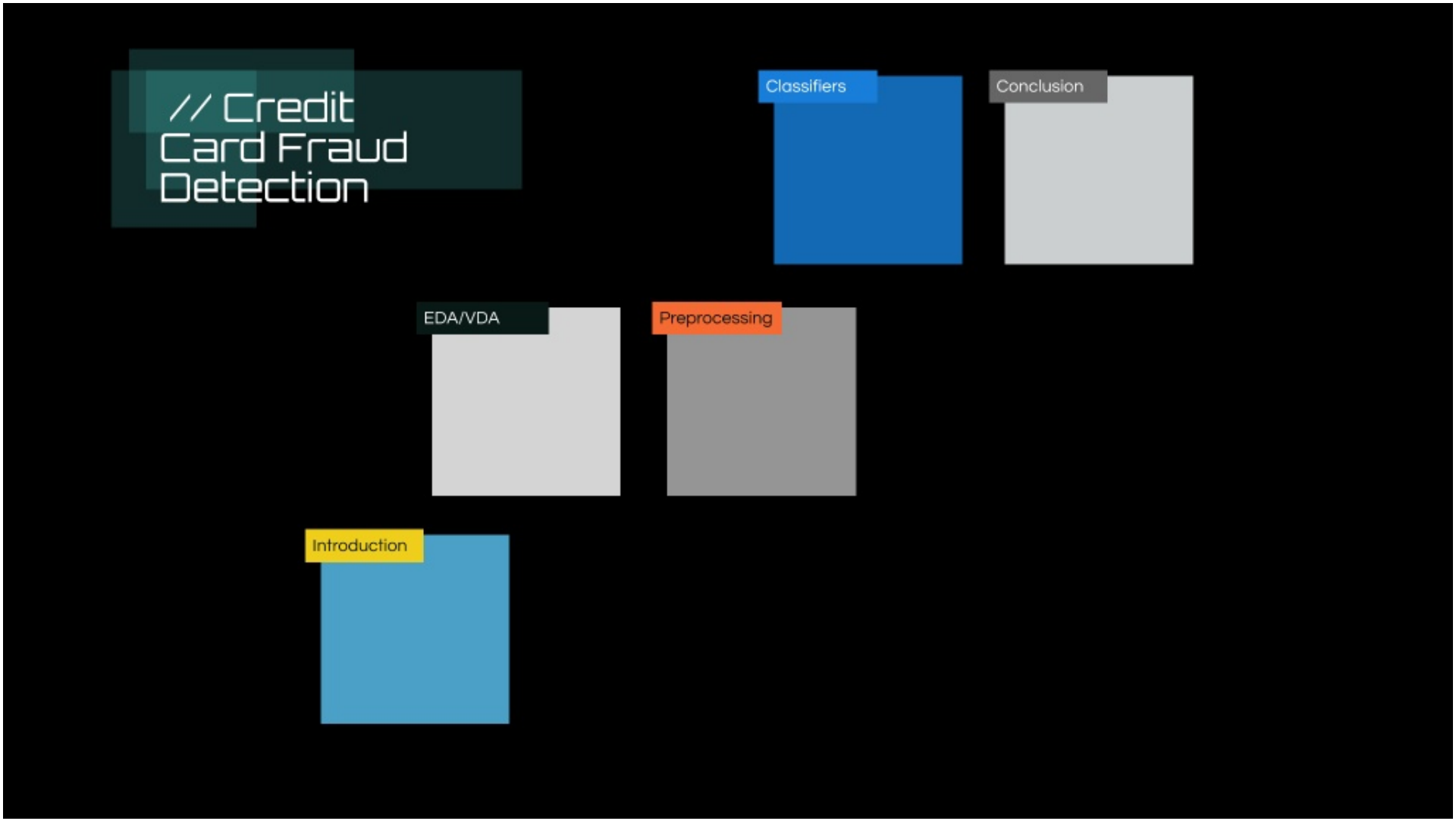


GradientBoostingClassifier



- Accuracy: 0.91 (std: 0.13)
- Recall: 0.96
- Precision: 0.05
- F1: 0.1
- AUC Score: 0.9969
- AP: 0.77



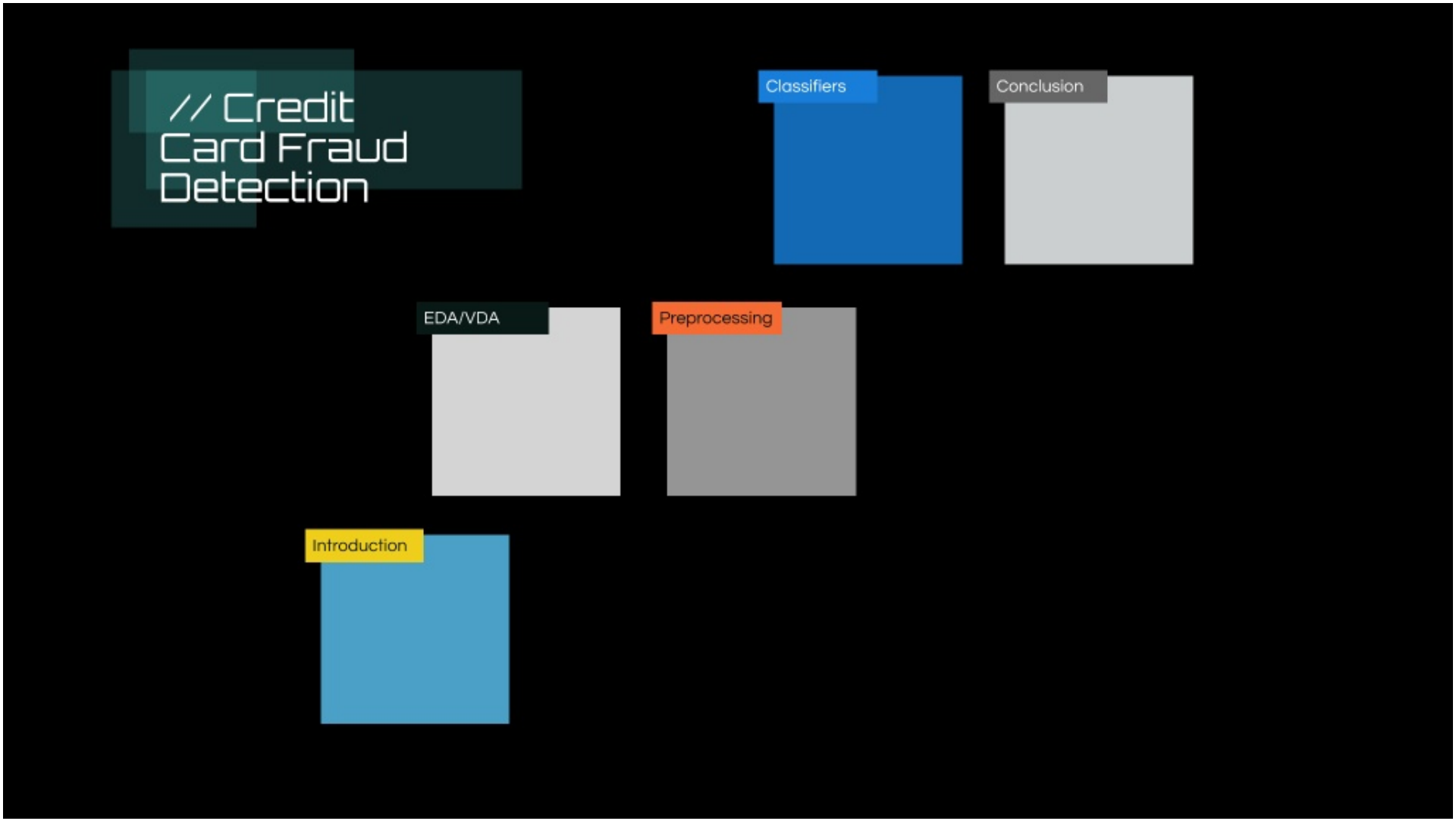

// Credit Card Fraud Detection
Classifiers
Conclusion
EDA/VDA
Preprocessing
Introduction

Conclusion

AUC score is representing the probability that a classifier will rank a randomly chosen positive observation higher than a randomly chosen negative observation, and thus it is a useful metric even for datasets with highly unbalanced classes.
Moreover, as the positive class is smaller and the ability to detect correctly positive samples is our main focus (correct detection of non-fraud examples is less important to the problem) we should use precision and recall.

In our case, Precision score is the best to reflect how poor is the model's detection ability. The higher Precision score, the better a model detects positive samples.

As a most accurate model, I would choose the one which is able to catch all frauds (highest Recall score) with the lowest number of False Positives (highest Precision score).
For this specific task, from 8 classifiers only RandomForestClassifier has the highest AUC, Precision, F1, AP and Recall scores.


// Credit Card Fraud Detection
Classifiers
Conclusion
EDA/VDA
Preprocessing
Introduction