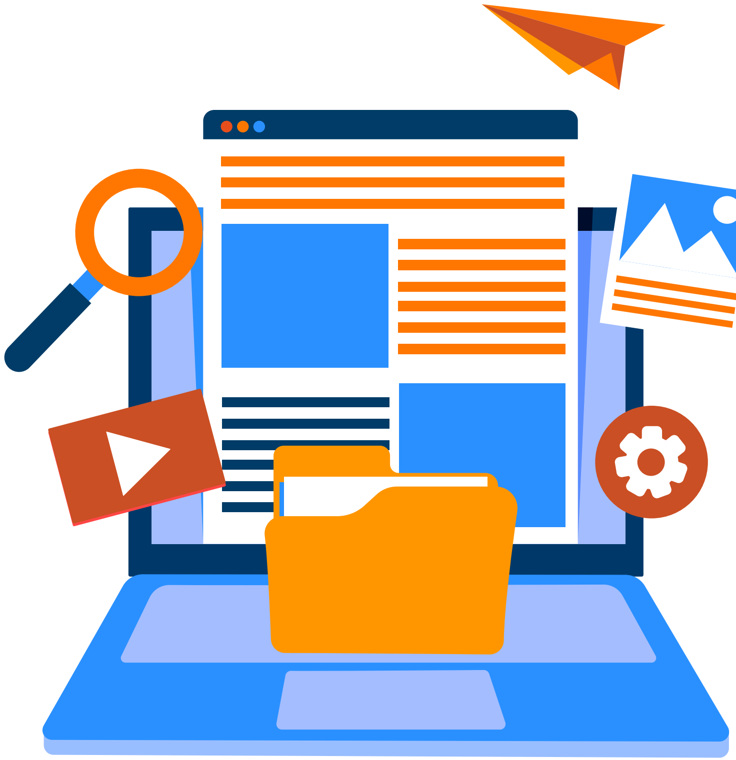
Credit Card Fraud Detection

Made by:
Aida Himmiche
Michael Asante



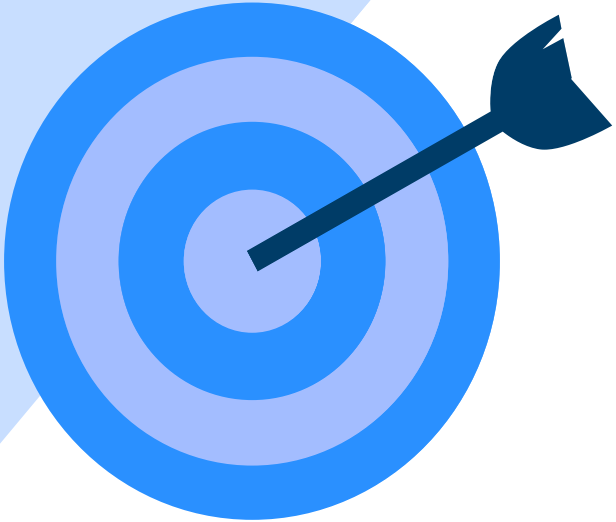
Introduction

The purpose of this project is to build an application which operates in the domain of banking or financial organizations, in order to detect fraud in a list of credit card transactions made by a customer.

This application was implemented using:

- **Supervised Machine Learning:** Classification with Random Forest.
- **Weka:** Dataset/model evaluation and analysis
- **Python:** Implementation of the data mining part, and the application backend.
- **Flask:** The Python webapp framework to integrate the ML code.

Table of contents



01

KDD Process

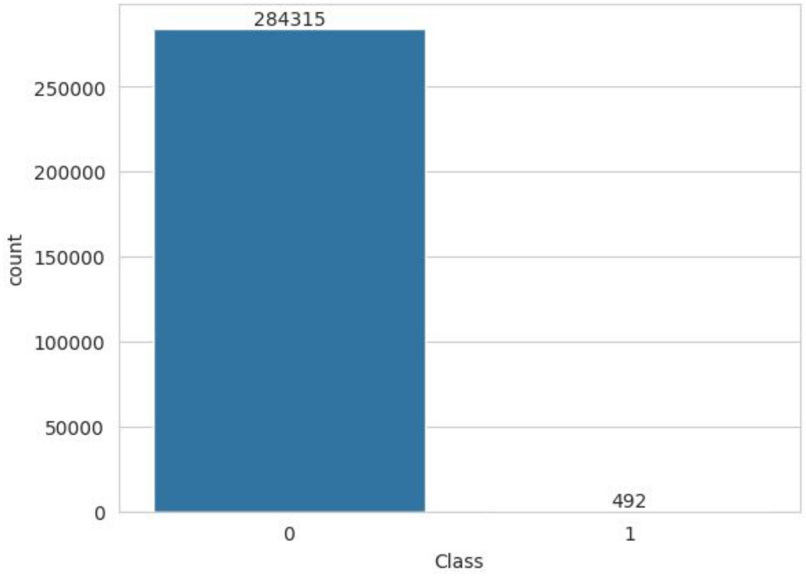
The Dataset

- The dataset used for this project contains 280 000+ transactions from a single credit card.

- It splits the columns into 31 features, 28 of which have been PCA transformed for confidentiality reasons. The rest are Time, Amount, and Class.

- All features contain numerical values .

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.215153	69.99	0



The Class Distribution

The dataset is extremely unbalanced:

284 315: normal class

491: fraud class

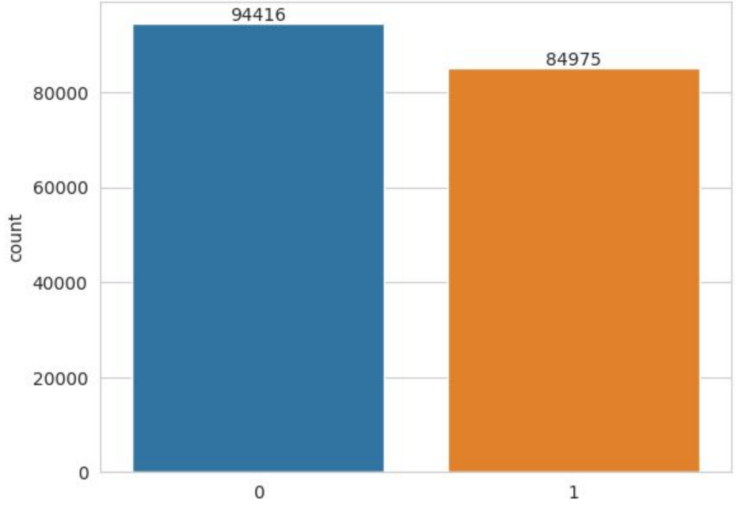
Preprocessing

The preprocessing was done in three simple steps:

- **Duplicates:** The number of duplicates found was 1081. Dropping the examples to a count of 283 726.

- **Missing values:** There were no null/missing values in this dataset.

- **Rebalancing:**
    - In Weka: using Resample and SMOTE
    - In Python: using RandomUnderSampler and SMOTE

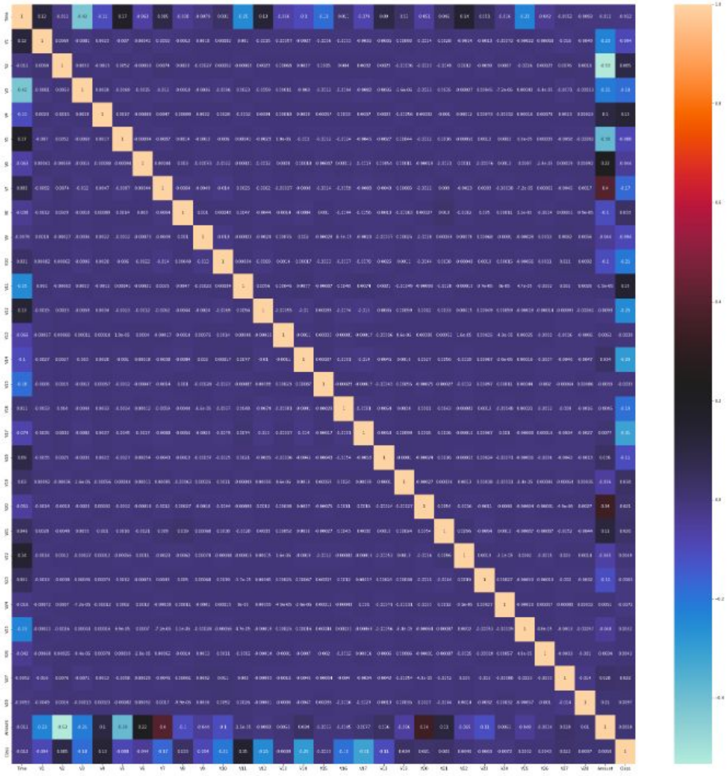


Data Mining: Attribute Selection

Looking at the correlation matrix, we can see that only about half the attribute have a significant correlation with the class.

Using Weka, we were able to try different algorithms for attribute selection namely:

- **CfsSubsetEval + BestFirst:** 8 features: [V3, V4, V10, V11, V12, V14, V16, V17]
- **CorrelationAttributeEval + Ranker:** Similar to the first; it placed the same selected attributes in the top 8 except V2, instead replacing it with V9 as it calculated the Pearson correlation to be higher.
- **PrincipalComponent + Ranker:** 19 Attributes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]



Cross- Validation

To make sure the model has learned patterns and not the data itself, it is necessary to test it against unseen data.

For this we chose the Holdout cross validation method with 66% for the train set and 34% for the test set.

Classification: Model Evaluation

The following evaluation measures were considered for this step

Accuracy: Correctly classified per total.
MAE: Errors between predictions and observations, all have the same weight.
RMSE: Errors between actual and predicted values, gives weight to larger errors.
RAE: Absolute different between actual and predicted values.
RRSE: How a model performs compared to a simple model. The best value is the closest to zero.
Precision: How many predictions of a class actually belong in the class.
Recall: How many instances were correctly classified per total instances of that class.
F-measure: Harmony measure between Precision and Recall, the higher the better.

Classification: Model Evaluation

Classifier	Evaluation Method	Feature Selection	Accuracy %	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error %	Root Relative Squared Error %	Precision	Recall	F-Measure
Random Forest	66% train – 34% test	-	99.831	0.012	0.048	2.451	9.801	0.998	0.998	0.998
C45 Pruned	66% train – 34% test	-	99.364	0.007	0.078	1.565	15.789	0.994	0.994	0.994
C45 Unpruned	66% train – 34% test	-	99.353	0.007	0.079	1.521	15.942	0.994	0.994	0.994
RandomTree	66% train – 34% test	-	98.937	0.011	0.103	2.154	20.764	0.989	0.989	0.989
logistic Function	66% train – 34% test	-	98.243	0.028	0.117	5.836	23.692	0.983	0.982	0.982
AdaBoost	66% train – 34% test	-	96.784	0.048	0.159	9.778	32.199	0.968	0.968	0.968
Naive Bayes	66% train – 34% test	-	94.176	0.058	0.236	11.841	47.718	0.943	0.942	0.941
ZeroR	66% train – 34% test	-	56.005	0.493	0.496	100	100	?	0.560	?
Random Forest	66% train – 34% test	Correlation AttributeEval+ Ranker	99.830	0.012	0.048	2.450	9.801	0.998	0.998	0.998
Random Forest	66% train – 34% test	CfSubsetEval + BestFirst	99.521	0.012	0.063	2.507	12.682	0.995	0.995	0.995
C45 pruned	66% train – 34% test	CorrelationAttributeEval + Ranker	99.364	0.007	0.0784	1.565	15.789	0.994	0.994	0.994
Random Tree	66% train – 34% test	CfSubsetEval + BestFirst	98.929	0.010	0.103	2.170	20.844	0.989	0.989	0.989
C45 pruned	66% train – 34% test	Principal Component + Ranker	98.586	0.016	0.115	3.355	23.304	0.986	0.986	0.986

Model Selection

Random Forest

This model performed the best in terms of accuracy, precision, recall, and F-measure, which are the measures decided to be most relevant for this task.

A Decision Tree based model is also an appropriate choice because of its readability and interpretability.

02

Implementation

Replicating the previous steps

Preprocessing:

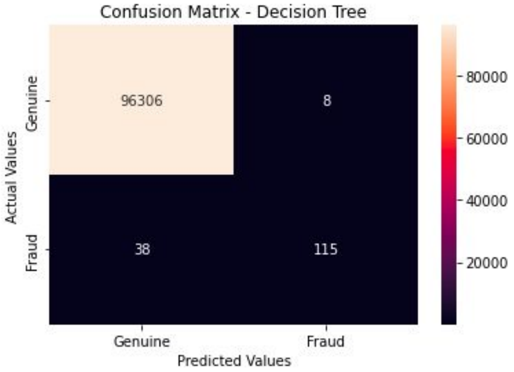
- **Duplicates:** data.drop_duplicates()
- **Rebalancing:**
  - SMOTE(sampling_strategy=0.3)
  - RandomUnderSampler(sampling_strategy=0.9)
- **Data split:** train_test_split(attribute_cols, class_col, split size, seed)

Feature Selection: We tried using the Random Forest based attribute selector “SelectFromModel” but the results were the same as CfsSubsetEval.

```python
# Split the data into training and testing sets
train_features,
test_features,
train_labels,
test_labels = train_test_split(features,
                               labels,
                               test_size = 0.34,
                               random_state = 42)
```

```
Training Features Shape: (187259, 30)
Training Labels Shape: (187259,)
Testing Features Shape: (96467, 30)
Testing Labels Shape: (96467,)
```

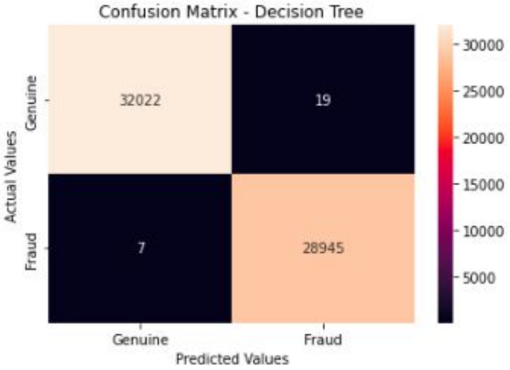
Random Forest Evaluation



	Metrics	Results
0	Accuracy	0.999523
1	Precision	0.934959
2	Recall	0.751634
3	F1_score	0.833333

Without Rebalancing

The python implementation produced a high accuracy of 99.95% most likely due to the higher examples of the majority class.



	Metrics	Results
0	Accuracy	0.999574
1	Precision	0.999344
2	Recall	0.999758
3	F1_score	0.999551

With Rebalancing

The same level of accuracy 99.95% but higher PRF scores. Valid this time.

03

Anomaly Detection

Unsupervised Learning

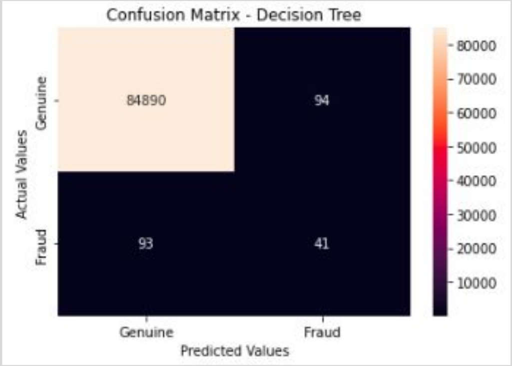
Isolation Forest

Orthogonal space splits + High anomaly score to fewest required “isolation” splits.

```
Errors:  187
Accuracy Score:
0.9978030498836908
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     84984
           1       0.30      0.31      0.30       134

    accuracy                           1.00     85118
   macro avg       0.65      0.65      0.65     85118
weighted avg       1.00      1.00      1.00     85118
```



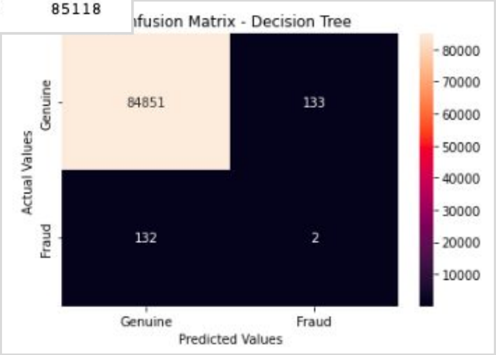
Local Outlier Factor (LOF)

Computes the local density deviation + Outliers are the points that have a substantially lower density than their neighbors.

```
Errors:  265
Accuracy Score:
0.9968866749688667
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     84984
           1       0.01      0.01      0.01       134

    accuracy                           1.00     85118
   macro avg       0.51      0.51      0.51     85118
weighted avg       1.00      1.00      1.00     85118
```



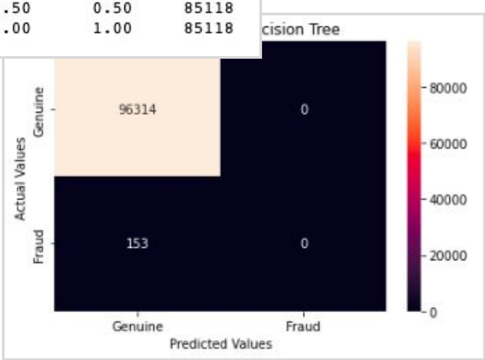
Unsupervised Learning

K-Means Clustering

Partitions N observations into K clusters in which each observation belongs to the cluster with the nearest mean.

Errors: 134
Accuracy Score:
0.9984257148899175
Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	84984
1	0.00	0.00	0.00	134
accuracy			1.00	85118
macro avg	0.50	0.50	0.50	85118
weighted avg	1.00	1.00	1.00	85118



One-Class Support Vector Machine (SVM)

The support vector machine algorithm finds a hyperplane in an N-dimensional space that distinctly classifies the data points using the largest possible margin.

The one class SVM uses a (smallest possible) hypersphere

Errors: 34054
Accuracy Score:
0.5999201109048615
Classification Report:

	precision	recall	f1-score	support
0	1.00	0.60	0.75	84984
1	0.00	0.41	0.00	134
accuracy			0.60	85118
macro avg	0.50	0.51	0.38	85118
weighted avg	1.00	0.60	0.75	85118

Reflections…

Even if Anomaly Detection may sound more appropriate for this kind of problem, the performance compared to supervised learning was not impressive.

This method could be useful in the case of non-availability of labeled data, notably the Isolation Forest model.

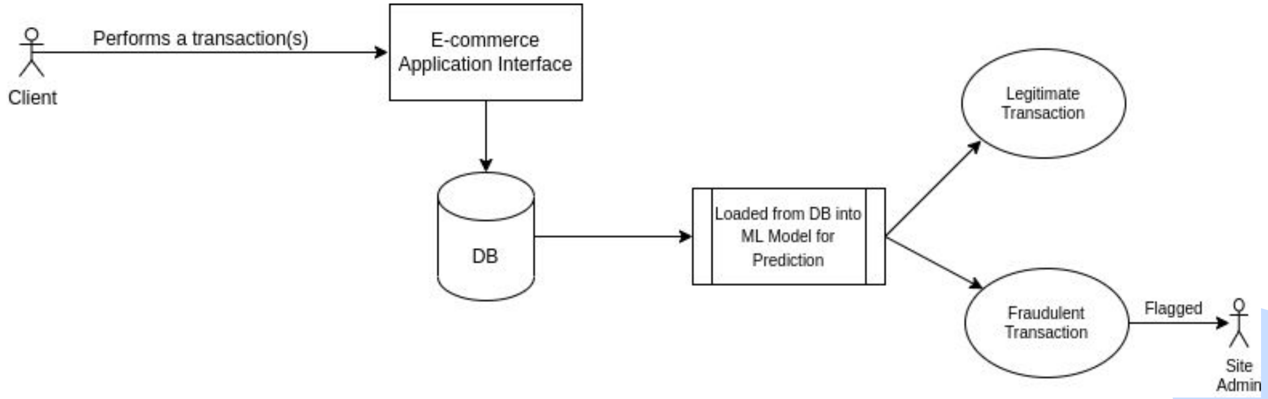
03

Application

Purpose of the Application

This application was designed to serve as a service used by financial organizations in order to determine whether the transactions of a customer are fraudulent or not.

We can illustrate a version of this goal in the following diagram:



Design of the Application

To simplify the scope of the web application, we described the functional and non-functional requirements as follows:

Functional:

- The user may input a list of transactions through the platform in a ".csv" format
- They may see their inputted transactions on the web page.
- They may use the model to predict the class of each transaction.

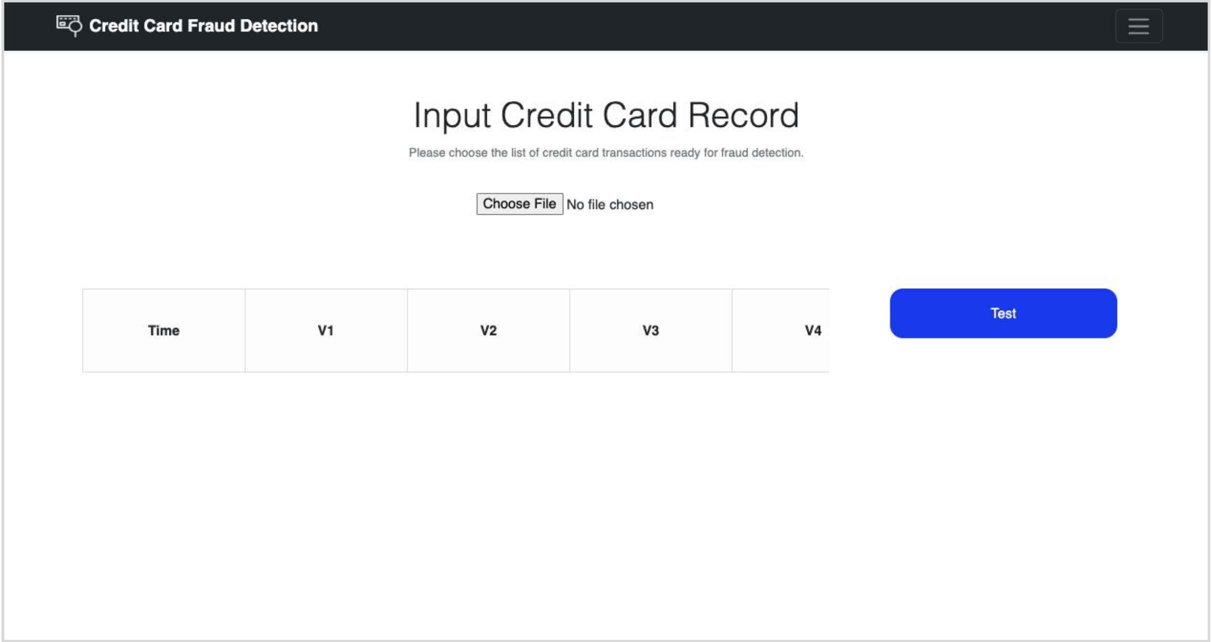
Non-functional:

- The app shall be easy to use
- The user shall not wait long for any of the functionalities above.
- The model must predict at best accuracy and reduce false positives/false negatives.

Testing the Application

The front page:

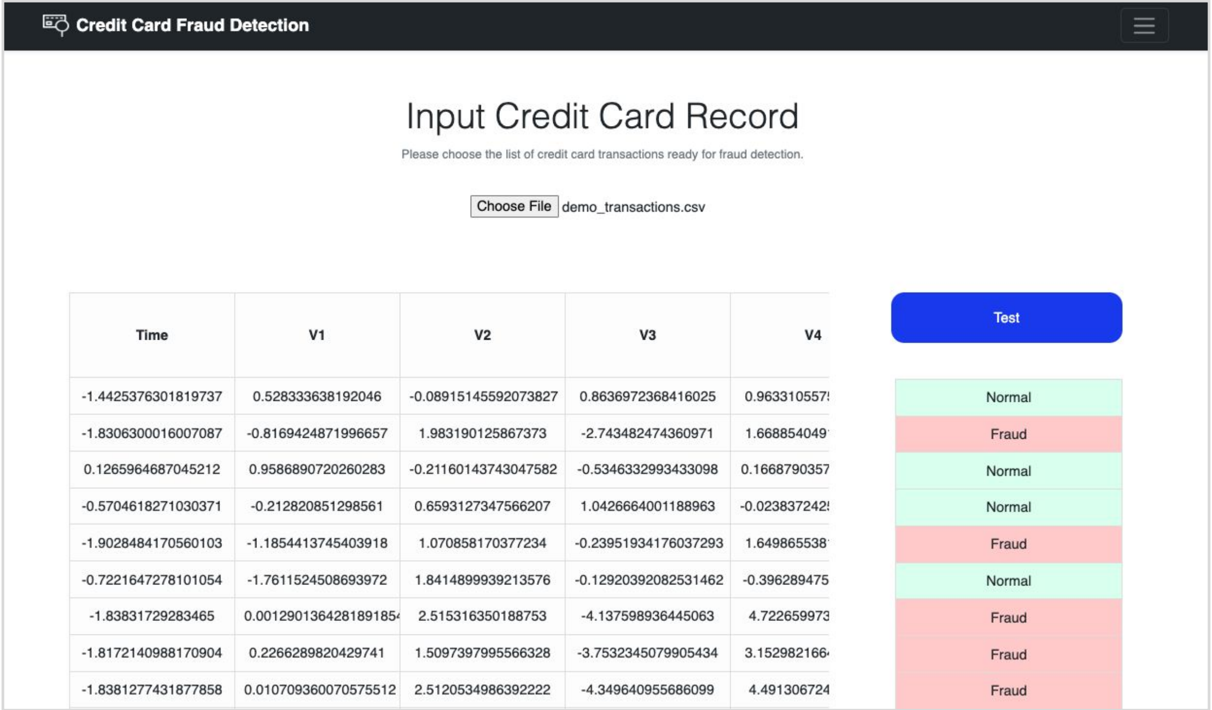
Allows a user to choose their preferred file. (List of transactions)



Testing the Application

The prediction results:

What the user sees after clicking the “Test” button.



Credit Card Fraud Detection

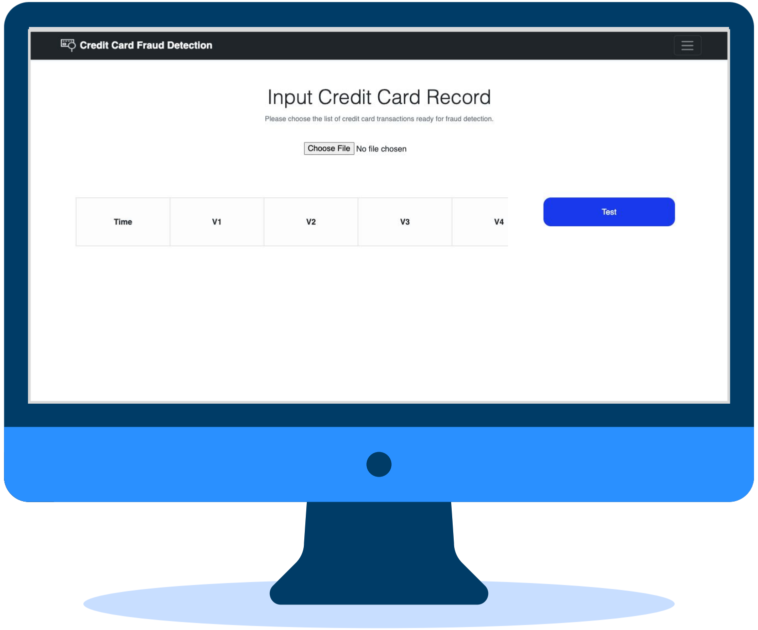
Input Credit Card Record

Please choose the list of credit card transactions ready for fraud detection.

Choose File demo_transactions.csv

Test

Time	V1	V2	V3	V4	
-1.4425376301819737	0.528333638192046	-0.08915145592073827	0.8636972368416025	0.963310557	Normal
-1.8306300016007087	-0.8169424871996657	1.983190125867373	-2.743482474360971	1.668854049	Fraud
0.1265964687045212	0.9586890720260283	-0.21160143743047582	-0.5346332993433098	0.1668790357	Normal
-0.5704618271030371	-0.212820851298561	0.6593127347566207	1.0426664001188963	-0.023837242	Normal
-1.9028484170560103	-1.1854413745403918	1.070858170377234	-0.23951934176037293	1.649865538	Fraud
-0.7221647278101054	-1.7611524508693972	1.8414899939213576	-0.12920392082531462	-0.396289475	Normal
-1.83831729283465	0.00129013642818918	2.515316350188753	-4.137598936445063	4.722659973	Fraud
-1.8172140988170904	0.2266289820429741	1.5097397995566328	-3.7532345079905434	3.152982166	Fraud
-1.8381277431877858	0.010709360070575512	2.5120534986392222	-4.349640955686099	4.491306724	Fraud


Credit Card Fraud Detection
Input Credit Card Record
Please choose the list of credit card transactions ready for fraud detection.
Choose File No file chosen
Time
V1
V2
V3
V4
Test

Live Demo!

Thanks!

Do you have any questions?