

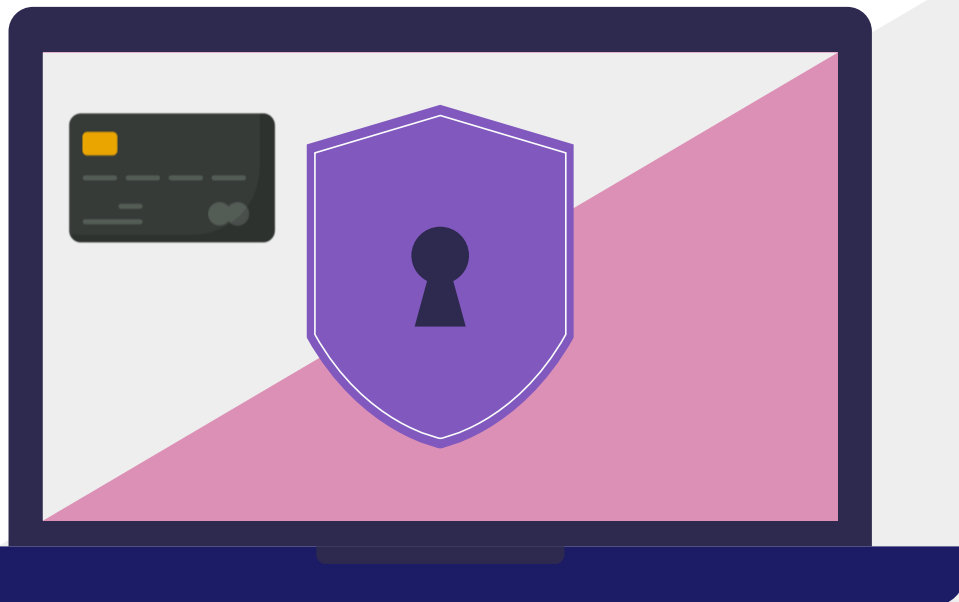
# Credit Card Fraud Detection

○ Encadré par :

- Mr BELMEKKI Abdelhamid
- Mr KAMAL IDRISSE Hamza
- Mme HANIN Charifa

○ Organisé par :

- AIT WAHMANE Elhoussaine
- IGAJANE Abdellah
- LACHEGUR Amine
- LAGHFIRI Oussama
- MOUISSET Hamza



# SOMMAIRE

## INTRODUCTION

Contexte général du sujet

1

## TESTS D'ENTRAINEMENT

Test des algorithmes

3

## Application WEB

Structure et Principe de fonctionnement

2

## CONCLUSION

4



1



# INTRODUCTION



# INTRODUCTION

De nos jours, la fraude par carte de crédit constitue un défi majeur dans le secteur des transactions financières, en raison de la hausse considérable des paiements par carte, représentant environ 51% de toutes les transactions selon le Federal Bank Reserve de San Francisco. L'avènement des technologies numériques et la popularité croissante des paiements par carte de crédit ont ouvert de nouvelles opportunités aux fraudeurs pour exploiter les failles du système.



# STATISTIQUES

Selon un rapport de l'équipe security.org, les statistiques suivantes sont établies:



1

65% des titulaires de carte de crédit en Amérique ont été victimes de fraude.

2

En 2022, 44% des utilisateurs de cartes de crédit ont déclaré avoir au moins deux débits frauduleux contre 35% en 2021.

3

L'accusation médiane est équivalente à environ 12 milliards de dollars de tentatives de transactions frauduleuses.

4

Les personnes qui stockent leur informations de la carte dans leur navigateur étaient plus susceptibles d'être victimes.

# TYPES DE FRAUDES



## Vol de carte

Le fraudeur prend possession de la carte et peut effectuer des transactions frauduleuses.

## Piratage des comptes de messagerie

Les banques envoient des e-mails sur les informations des clients que les pirates peuvent exploiter.

## SCAM

Les victimes reçoivent des appels ou des e-mails par des arnaques qui prétendent d'être des employés de banque

## Hameçonnage (Phishing)

Les clients reçoivent des liens ou messages malveillants.

## Ecrémage (Skimming)

Les guichets équipés par des dispositifs qui copient les informations de la carte bancaire.

# CONTEXTE GENERALE DU SUJET

**APPLICATION  
WEB**



Interface pour les  
administrateurs

Entrainement du modèle

**ENTRAINEMENT**



**Test**



Tester et analyser les  
transactions

Détecter les fraudes et  
afficher les résultats

**RESULTATS**



2



**APPLICATION WEB**



- **Page D'accueil :**

# STRUCTURE



CREDIT  
CARD  
FRAUD  
DETECTION

[HOME](#)

[ABOUT](#)

[CONTACT](#)

[LOG IN](#)

## Credit Card Fraud Detection

Protecting Your Finances

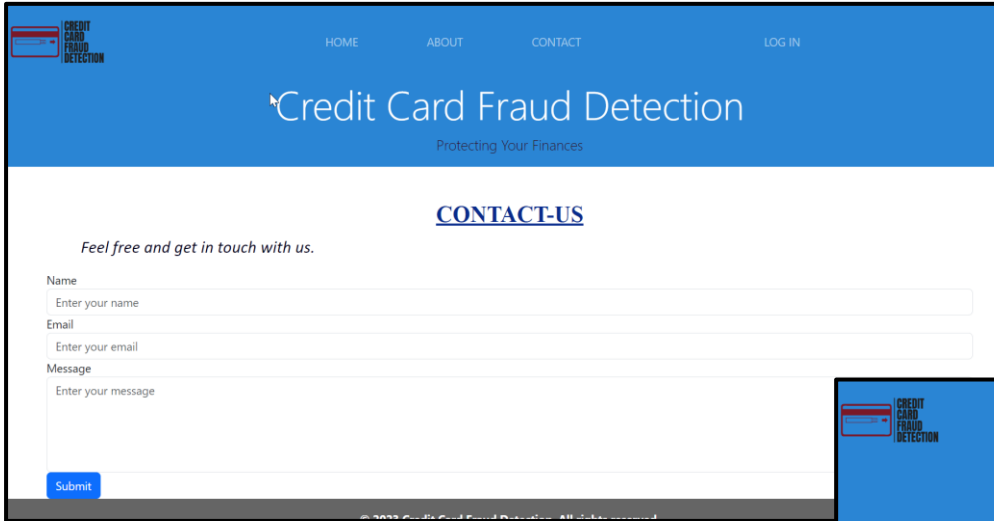


### Credit Card Fraud Detection

Using the Machine Learning Classification Algorithms to detect Credit Card Fraudulent Activities

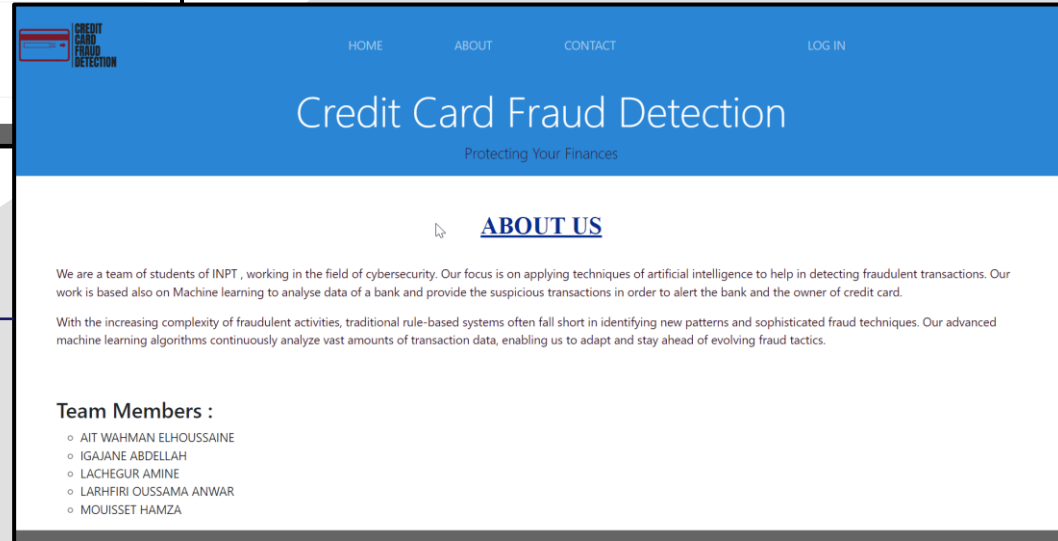
**Securing your account, is our priority**

# STRUCTURE



The screenshot shows the 'Contact' page of the 'Credit Card Fraud Detection' website. The page has a blue header with the logo and navigation links: HOME, ABOUT, CONTACT, and LOG IN. The main content area is white and features the title 'Credit Card Fraud Detection' with the tagline 'Protecting Your Finances'. Below this is a section titled 'CONTACT-US' with the text 'Feel free and get in touch with us.' followed by a form with three input fields: 'Name' (with placeholder 'Enter your name'), 'Email' (with placeholder 'Enter your email'), and 'Message' (with placeholder 'Enter your message'). A blue 'Submit' button is at the bottom left of the form.

→ **Contact**



The screenshot shows the 'About Us' page of the 'Credit Card Fraud Detection' website. The page has a blue header with the logo and navigation links: HOME, ABOUT, CONTACT, and LOG IN. The main content area is white and features the title 'Credit Card Fraud Detection' with the tagline 'Protecting Your Finances'. Below this is a section titled 'ABOUT US' with a paragraph of text: 'We are a team of students of INPT, working in the field of cybersecurity. Our focus is on applying techniques of artificial intelligence to help in detecting fraudulent transactions. Our work is based also on Machine learning to analyse data of a bank and provide the suspicious transactions in order to alert the bank and the owner of credit card.' followed by another paragraph: 'With the increasing complexity of fraudulent activities, traditional rule-based systems often fall short in identifying new patterns and sophisticated fraud techniques. Our advanced machine learning algorithms continuously analyze vast amounts of transaction data, enabling us to adapt and stay ahead of evolving fraud tactics.' Below the text is a section titled 'Team Members :' followed by a list of five team members: AIT WAHMAN ELHOUSSEINE, IGAJANE ABDELLAH, LACHEGUR AMINE, LARHFIRI OUSSAMA ANWAR, and MOUISSET HAMZA.

**À propos de nous** →

# STRUCTURE

**Page d'authentification**

Login to continue

Email / Username

Password

☐ Remember me

[Not a member? Click to sign up](#)

LOGIN

Sign Up

Username

Email

Password

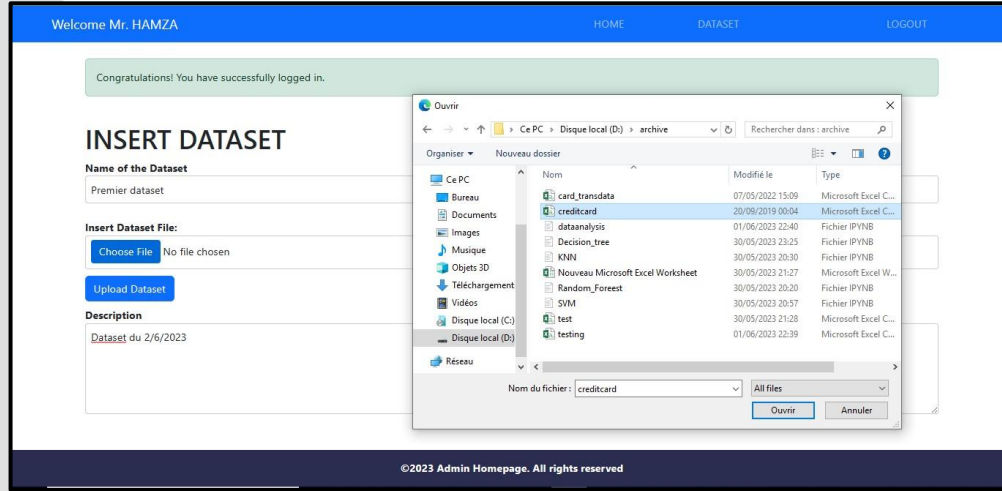
Retype password

[Already a member? Click to login](#)

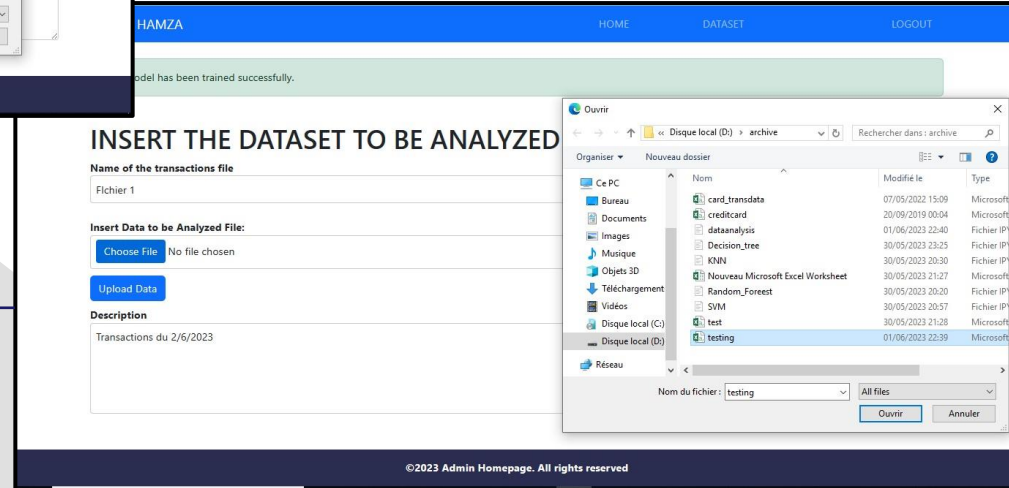
SIGN UP

**Page de création de compte**

# STRUCTURE



Insertion de dataset



Insertion des transactions à analyser

# DATASET

DATASET qu'on va utiliser :

```
data=pd.read_csv('creditcard.csv')  
data.head(100)
```

✓ 6.6s

Python

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-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	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	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	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-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	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	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	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	64.0	-0.658305	0.406791	2.037461	-0.291298	0.147910	-0.350857	0.945373	-0.172560	0.025133	...	-0.156096	-0.238805	0.089877	0.421195	-0.352487	0.074783	-0.094192	-0.092493	54.99	0
96	64.0	0.959602	0.370711	0.888613	2.343244	0.352491	1.365515	-0.277771	0.516053	-0.700929	...	-0.155547	-0.403239	0.356504	-0.696881	-0.198718	-0.220268	0.068546	0.020797	7.55	0
97	67.0	-0.653445	0.160225	1.592256	1.296832	0.997175	-0.343000	0.469937	-0.132470	-0.197794	...	0.038363	0.336449	-0.014883	0.102959	-0.265322	-0.348637	0.011238	-0.049478	19.85	0
98	67.0	-1.494668	0.837241	2.628211	3.145414	-0.609098	0.258495	-0.012189	0.102136	-0.286164	...	-0.140047	0.355044	0.332720	0.718193	-0.219366	0.118927	-0.317486	-0.340783	28.28	0
99	68.0	1.232996	0.189454	0.491040	0.633673	-0.511574	-0.990609	0.066240	-0.196940	0.075921	...	-0.251566	-0.770139	0.125998	0.369627	0.205598	0.094062	-0.033138	0.020990	15.99	0

100 rows × 31 columns

data.describe()

✓ 1.5s

Python

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	...	2.848070e+05	2.848070e+05	2.848070e+05	2.
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15	...	1.654067e-16	-3.568593e-16	2.578648e-16	4
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	...	7.345240e-01	7.257016e-01	6.244603e-01	6
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	...	-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	...	-2.283949e-01	-5.423504e-01	-1.618463e-01	-3
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	...	-2.945017e-02	6.781943e-03	-1.119293e-02	4
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	...	1.863772e-01	5.285536e-01	1.476421e-01	4
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	...	2.720284e+01	1.050309e+01	2.252841e+01	4.

8 rows x 31 columns

data.info()

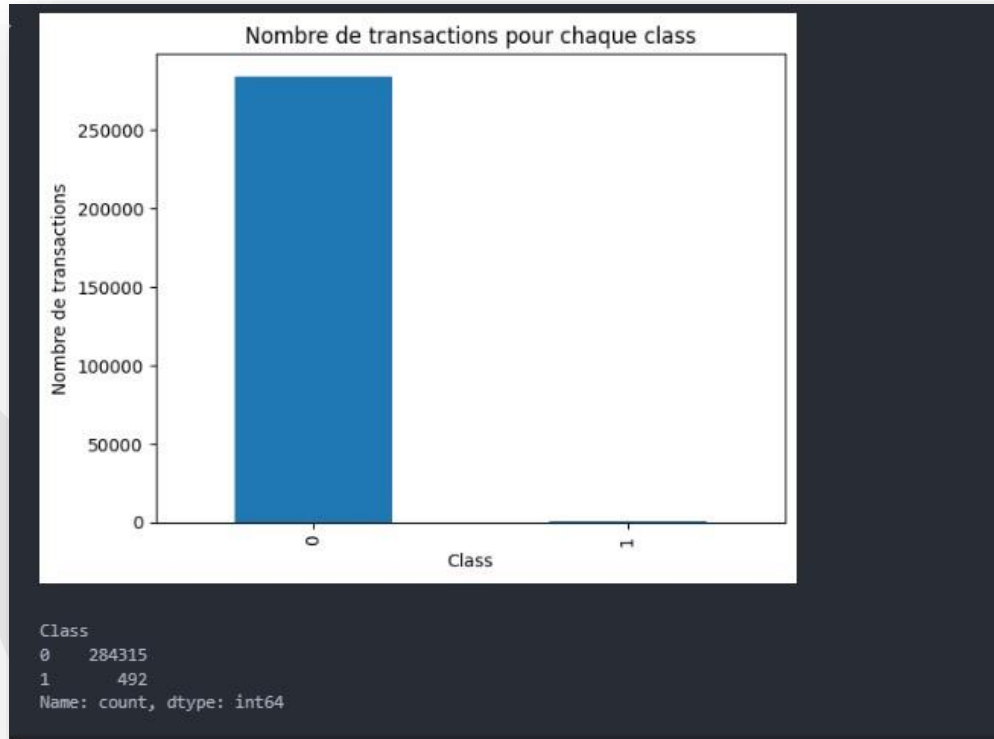
✓ 0.3s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
#   Column  Non-Null Count  Dtype
---  -
0    Time    284807 non-null  float64
1    V1       284807 non-null  float64
2    V2       284807 non-null  float64
3    V3       284807 non-null  float64
4    V4       284807 non-null  float64
5    V5       284807 non-null  float64
6    V6       284807 non-null  float64
7    V7       284807 non-null  float64
8    V8       284807 non-null  float64
9    V9       284807 non-null  float64
10   V10      284807 non-null  float64
11   V11      284807 non-null  float64
12   V12      284807 non-null  float64
13   V13      284807 non-null  float64
14   V14      284807 non-null  float64
15   V15      284807 non-null  float64
16   V16      284807 non-null  float64
17   V17      284807 non-null  float64
18   V18      284807 non-null  float64
19   V19      284807 non-null  float64
...
29   Amount  284807 non-null  float64
30   Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

## Description du dataset

# DATASET



Parmi 284807 transactions , il en existe 492 transactions qui sont frauduleuses.

# Principe de Fonctionnement

Pour détecter les transactions frauduleuses depuis le dataset, on va utiliser des multiples algorithmes d'apprentissage automatique et évaluer leurs performances.





3



**TESTS D'ENTRAINEMENT**

- **Régression Logistique**

# RESULTATS

```
xtrain,xtest=train_test_split(X, test_size=0.30, random_state=100)
ytrain,ytest=train_test_split(Y, test_size=0.30, random_state=100)
logit_model = LogisticRegression(penalty='l2')
log=logit_model.fit(xtrain,ytrain)
print("Coefficients:", logit_model.coef_)
print("Intercept:", logit_model.intercept_)
```

✓ 8.4s

```
Coefficients: [[-9.21661922e-05  3.30392207e-01 -4.67389995e-01 -5.96788897e-01
 1.60539690e-01  8.96095860e-02 -9.19942820e-02  1.62076635e-01
-2.43180483e-01 -4.42358475e-01 -3.05669410e-01 -1.06154685e-01
-8.13307091e-02 -3.14743142e-01 -8.19990144e-01 -2.45291092e-01
-3.48283699e-01 -6.60236838e-01 -8.13007828e-02  6.85685162e-02
 5.24677866e-02  1.91842379e-01  2.16205215e-01  5.24250940e-02
-2.55560228e-02 -2.02926644e-01  3.15539884e-02 -3.42039266e-02
 6.12273184e-02 -4.97653259e-03]]
```

```
Intercept: [-1.03862014]
```

```
c:\Python311\Lib\site-packages\sklearn\linear_model\logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

```
y_pred = logit_model.predict(xtest)
score = logit_model.score(xtest, ytest)
print(classification_report(ytest, y_pred))
print(accuracy_score(ytest, y_pred))
```

✓ 0.3s

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85301
1	0.71	0.65	0.68	142
accuracy			1.00	85443
macro avg	0.85	0.82	0.84	85443
weighted avg	1.00	1.00	1.00	85443

0.9989700736163291

**Prédiction**

- **Random Forrest**

# RESULTATS

```
xtrain,xtest=train_test_split(X, test_size=0.30, random_state=100)
ytrain,ytest=train_test_split(y, test_size=0.30, random_state=100)
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(xtrain, ytrain)
```

✓ 10m 56.8s

RandomForestClassifier  
RandomForestClassifier(random\_state=42)

```
y_pred = model.predict(xtest)
score = model.score(xtest, ytest)
print(classification_report(ytest, y_pred))
print(accuracy_score(ytest, y_pred))
```

✓ 5.5s

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85301
1	0.91	0.75	0.82	142
accuracy			1.00	85443
macro avg	0.95	0.88	0.91	85443
weighted avg	1.00	1.00	1.00	85443

0.9994616293903538

**Prédiction**

- **KNN**

# RESULTATS

```
X_train, X_test = train_test_split( x, test_size=0.3, random_state=100)
y_train, y_test = train_test_split( y, test_size=0.3, random_state=100)
model = KNeighborsClassifier()
model.fit(X_train, y_train)
```

✓ 0.6s

```
* KNeighborsClassifier
KNeighborsClassifier()
```

```
y_pred = model.predict(X_test)
score = model.score(X_test, y_test)
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

✓ 3m 7.0s

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85301
1	1.00	0.07	0.13	142
accuracy			1.00	85443
macro avg	1.00	0.54	0.57	85443
weighted avg	1.00	1.00	1.00	85443

0.9984551104244935

**Prédiction**

- SVM

# RESULTATS

```
xtrain,xtest=train_test_split(x, test_size=0.30, random_state=100)
ytrain,ytest=train_test_split(y, test_size=0.30, random_state=100)
model = SVC()
model.fit(xtrain, ytrain)
```

✓ 36.1s

• SVC

SVC()

```
y_pred = model.predict(xtest)
score = model.score(xtest, ytest)
print(classification_report(ytest, y_pred))
print(accuracy_score(ytest, y_pred))
```

✓ 26.7s

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	85301
1	0.00	0.00	0.00	142

accuracy			1.00	85443
----------	--	--	------	-------

macro avg	0.50	0.50	0.50	85443
-----------	------	------	------	-------

weighted avg	1.00	1.00	1.00	85443
--------------	------	------	------	-------

0.99833807333544

**Prédiction**

- **Decision Tree**

# RESULTATS

```
xtrain,xtest=train_test_split(x, test_size=0.30, random_state=100)
ytrain,ytest=train_test_split(y, test_size=0.30, random_state=100)
model = DecisionTreeClassifier()
model.fit(xtrain, ytrain)
```

✓ 50.3s

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred = model.predict(xtest)
score = model.score(xtest, ytest)
print(classification_report(ytest, y_pred))
print(accuracy_score(ytest, y_pred))
```

✓ 0.4s

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85301
1	0.67	0.75	0.70	142
accuracy			1.00	85443
macro avg	0.83	0.87	0.85	85443
weighted avg	1.00	1.00	1.00	85443

0.9989583699074237

**Prédiction**

# COMPARAISON

	Régression Logistique	Random Forrest	KNN	SVM	Decision Tree
<b>Précision</b>	99.89%	99.94%	99.84%	99.83%	99.89%
<b>Temps d'entraînement</b>	8.4 s	10 min 56 s	0.6 s	36.1 s	50.3 s
<b>Temps de test</b>	0.3 s	5.5 s	3 min 7 s	26.7 s	0.4 s

# RESULTATS

Welcome Mr. HAMZA123

[HOME](#)

[LOGOUT](#)

## Results

- 1 Fraudulant
- 0 Valid
- 1 Fraudulant
- 0 Valid
- 0 Valid
- 1 Fraudulant
- 0 Valid
- 1 Fraudulant
- 0 Valid
- 1 Fraudulant



5



# CONCLUSION

# CONCLUSION

Dans le cadre de notre projet de détection de fraudes par carte de crédit, nous avons développé une application web utilisant des algorithmes de Machine Learning tels que la régression logistique, le KNN, le SVM, le Random Forest et le Decision Tree. Notre système offre une interface conviviale permettant aux administrateurs de soumettre leurs transactions pour évaluation, avec un modèle entraîné pour effectuer des prédictions en temps réel. Bien que la détection de fraudes soit un défi en constante évolution, notre projet a été un succès en renforçant la sécurité des transactions et en protégeant les utilisateurs contre les activités frauduleuses.

# ANNEXE

- Equilibrage des données :

```
class_counts = {class_label: len(y[y == class_label]) for class_label in set(y)}  
labels = list(class_counts.keys())  
counts = list(class_counts.values())  
plt.bar(labels, counts)  
plt.xlabel('Class')  
plt.ylabel('Nombre de transactions')  
plt.title('Nombre de transactions pour chaque class')  
  
plt.show()
```

✓ 0.7s

```
oversample=SMOTE()  
X_sm, y =oversample.fit_resample(X,Y)
```

✓ 1.9s

# ANNEXE

