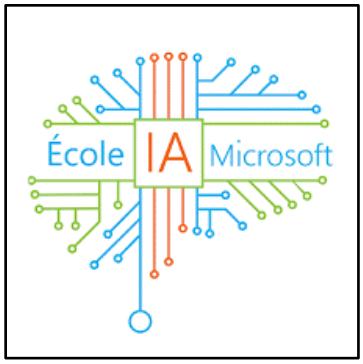


Amélioration d'un cas de machine learning

Challenge Data : Comment démasquer les fraudeurs ?

par BNP Paribas PF

Titre professionnel « Développeur en intelligence artificielle » de niveau 6 enregistré au RNCP sous le n°34757

Passage par voie de formation

Parcours de 19 mois achevé le 10 mai 2023

Promotion SIMPLON – LANNION 2022 – 2023

Thomas JONCOURT

Table des matières

Contexte **3**

Objectif **3**

Base de données **3**

Description du benchmark **4**

Métrique d'évaluation *4*

Benchmark *5*

Pré-processing **5**

Nettoyage du texte *5*

One hot encoding *6*

Gestion de la taille des données à entraînés *6*

Choix de l'algorithme de machine learning **7**

Gestion du déséquilibres des classes **8**

Résultats **9**

Conclusion **11**

Contexte

La fraude est un problème majeur de plus en plus préoccupant pour les institutions financières du monde entier. Les criminels utilisent une grande variété de méthodes pour attaquer des organisations comme la BNP Paribas, quels que soient les systèmes, les canaux, les process ou les produits.

Le développement de méthodes de détection de la fraude est stratégique et essentiel pour les établissements bancaires. Les fraudeurs s'avèrent toujours très créatifs et ingénieux pour normaliser leurs comportements et les rendre difficilement identifiables. Une contrainte s'ajoute à cette problématique, la faible occurrence de la fraude dans la population.

Lien du challenge : https://challengedata.ens.fr/challenges/104

Objectif

L'objectif de ce challenge est de trouver la meilleure méthode pour transformer et agréger les données relatives au panier client d'un des partenaires de la BNP pour détecter les cas de fraude.

En utilisant ces données panier, les fraudeurs pourront être détectés, et ainsi refusés dans le futur.

Base de données

La base contient une liste d'achats effectués sur le site partenaire. Les informations décrivent exclusivement le contenu du panier.

Pour chaque observation de la base, il y a 147 colonnes dont 144 peuvent être regroupées en 6 catégories :

- Item : Catégorie du bien de l'item, exemple : Computer
- cash_price : Prix de l'item, exemple : 850
- make : Fabriquant de l'Item, exemple : Apple
- model : Description du modèle de l'item, exemple : Apple Iphone XX
- goods_code : Code-barre de l'item, exemple : 2378284364
- Nbr_of_pro_purchas : Nombre de produits pour l'item, exemple : 2

Le panier se décompose au maximum en 24 items. Par exemple, si un panier contient 3 items alors toutes les informations relatives à ces 3 items seront renseignées dans les
colonnes item1, item2, item3, cash_price1, cash_price_2, cash_price3, make1, make 2, make3, model1, model2, model3, goods_code1, goods_code2, goods_code3, Nbr _of_prod_purchas1, Nbr_of_prod_purchas2 et Nbr_of_prod_purchas3. Les variables restantes (celles avec un indice > 3) seront vides .

Description du benchmark

Métrique d'évaluation

L'objectif est d'identifier une opération frauduleuse dans la population en prédisant un risque/probabilité de fraude. Par conséquent, la métrique à utiliser est l'aire sous la courbe Précision-Rappel, appelé également PR-AUC.

Cette métrique est appropriée pour évaluer correctement la performance d'un modèle sur la classe minoritaire dans le cadre d'une classification fortement déséquilibrée.

Le meilleur modèle correspondra à celui avec la valeur de PR-AUC la plus élevée.

Pour ce challenge, la PR-AUC sera estimée par la moyenne pondérées des précisions à chaque seuil avec le poids associé étant la variation du rappel entre le seuil précédent et le nouveau seuil :

Cette implémentation correspond à la métrique average_precision_score de scikit-learn.

Benchmark

Le benchmark intègre plusieurs étapes de pré-processing et utilise un modèle de Machine Learning optimisé pour prédire le risque de fraude. Les détails de ces étapes ne sont pas précisés par la BNP.

PR-AUC2=0,14PR-AUC2=0,14

Pré-processing

Nettoyage du texte

Il y a dans le jeu quelques coquille dans certains noms de marques ou de catégories. Ils sont corrigés au cas par cas avec une fonction créé manuellement.

```python
def clean(x):
  if type(x)==str:
    x = x.replace("& ","")
    x = x.replace(", ","")
    x = x.replace(" S ","S ")
    x = x.replace('TELEPHONESFAX','TELEPHONES FAX')
    x = x.replace('CARPETSRUGS','CARPETS RUGS')
    x = x.replace('BAGSWALLETS','BAGS WALLETS')
    x = x.replace('!','')
    x = x.replace('BABYBJ RN','BABYBJÖRN')
    x = x.replace('DR BROWN S','DR BROWNS')
    x = x.replace('KIEHL S','KIEHLS')
    x = x.replace('®','')
    x = x.replace('.','')
    x = x.replace('2','')
    x = x.replace(" + ","")
  return x
```

```python
for i in range(1,25):
  train[f"item{i}"] = train[f"item{i}"].apply(clean)
  train[f"make{i}"] = train[f"make{i}"].apply(clean)
  train[f"model{i}"] = train[f"model{i}"].apply(clean)
  test[f"item{i}"] = test[f"item{i}"].apply(clean)
  test[f"make{i}"] = test[f"make{i}"].apply(clean)
  train[f"model{i}"] = train[f"model{i}"].apply(clean)
```

One hot encoding

Avec ce jeu de données, le plus complexe est de transformer les informations contenues dans les colonnes item{i},make{i} et model{i} (i de 1 à 24) qui contiennent des données textuelles. Ces données doivent être regroupées par catégories et transformées en valeurs numériques. On ne peut pas dans ce cas de figure utilisé une librairie clé en main comme scikit-learn car il faut agréger les valeurs des 24 colonnes de chaque type de colonne (exemple : item) et les « additionner » ensemble.

J'ai donc préféré écrire un script manuellement pour mieux contrôler le résultat du one hot encoding.

Exemple avec les colonnes item{i} :

```
train_item_price = pd.DataFrame(columns=items,index=train.index).fillna(0)
train_item_count = pd.DataFrame(columns=items,index=train.index).fillna(0)
test_item_price = pd.DataFrame(columns=items,index=test.index).fillna(0)
test_item_count = pd.DataFrame(columns=items,index=test.index).fillna(0)
```

```
for row in train.index:
  for i in range(1,25):
    item = train[f"item{i}"][row]
    if item in items:
     train_item_price[item][row] += train[f"cash_price{i}"][row] * train[f"Nbr_of_prod_purchas{i}"][row]
     train_item_count[item][row] += train[f"Nbr_of_prod_purchas{i}"][row]

for row in test.index:
  for i in range(1,25):
    item = test[f"item{i}"][row]
    if item in items:
     test_item_price[item][row] += test[f"cash_price{i}"][row] * test[f"Nbr_of_prod_purchas{i}"][row]
     test_item_count[item][row] += test[f"Nbr_of_prod_purchas{i}"][row]
```

2 types de matrices sont créés : une avec le nombre d'occurrence de chaque item/marque dans le panier et une avec le prix cumulé des produits de chaque item/marque dans le panier

Gestion de la taille des données à entraînés

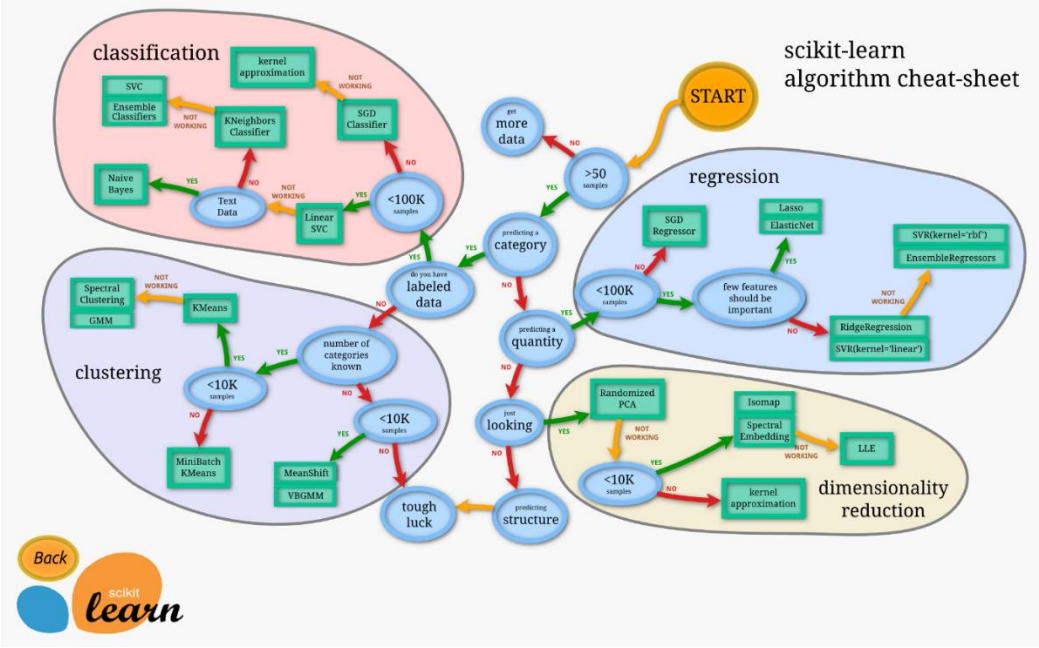
Il y 184 items, 811 marques, 9678 modèles et 14880 code-barres différents dans ce jeu de données et 92790 échantillons. Afin de réduire la taille finale du jeu d'entrainement, les colonnes contenants les code-barres seront écartés. Ensuite les résultats des phases de one hot encoding pour les autres catégories seront

sauvegardés sous formes de matrice creuse afin d'optimiser la mémoire RAM et la puissance de calcul nécessaire à l'entrainement à l'aide de la librairie scipy.

```
sps.save_npz(f'drive/MyDrive/Challenges/bnp_dataset/new/matrice/train_item_price.npz', csr_matrix(train_item_price.fillna(0)))
sps.save_npz(f'drive/MyDrive/Challenges/bnp_dataset/new/matrice/test_item_price.npz', csr_matrix(test_item_price.fillna(0)))
sps.save_npz(f'drive/MyDrive/Challenges/bnp_dataset/new/matrice/train_item_count.npz', csr_matrix(train_item_count.fillna(0)))
sps.save_npz(f'drive/MyDrive/Challenges/bnp_dataset/new/matrice/test_item_count.npz', csr_matrix(test_item_count.fillna(0)))
```

Choix de l'algorithme de machine learning

Pour m'orienter dans le choix du modèle de machine learning, je me suis aidé de la carte proposée par scikit-learn suivante :



A noter que pour pouvoir évaluer le modèle à l'aide de la métrique average_precision_score, il est nécessaire de choisir un algorithme qui permette de prédire une probabilité pour chaque classe. J'utiliserais donc en priorité les modèles « Ensemble Classifiers » de scikit-learn.

Gestion du déséquilibres des classes

Dans ce jeu de données, il y a une problématique que je n'avais encore jamais rencontré ; le jeu de données est déséquilibré (seulement 1.4% de fraude). J'essayerais donc d'utiliser trois méthodes pour ré-équilibrer ce dataset :

- L'undersampling : Il s'agit ici simplement de retirer aléatoirement des instances de la classe majoritaire afin de ré-équilibrer les proportions. On perd toutefois de l'information, et il y a donc un risque d'underfitting. Cette méthode est préconisée lorsque l'on dispose d'un très grand nombre d'observations **(à minima > 10K) comme c'est le cas pour notre dataset**.
- L'oversampling : Il s'agit ici de dupliquer aléatoirement certaines instances des classes minoritaires, rendant ainsi leur signal plus puissant. Il y a toutefois ici un risque d'overffiting. Cette méthode est préconisée lorsque l'on dispose d'un nombre limité d'observations (< 10K), ou bien si le temps de calcul n'est pas un problème.
- La méthode des class weights permets de prendre en compte le caractère biaisé de la distribution du dataset et de créer un modèle pénalisé. Il s'agit ici de simplement attribuer des poids différents aux différentes classes de notre dataset, en donnant un poids plus important aux classes minoritaires, afin d'influencer le modèle lors de son entraînement. Nous pénalisons ainsi plus fortement une erreur de classification d'une classe minoritaire par rapport à une erreur de classification d'une classe majoritaire.

Il est à noter que la plupart des modèles disponibles dans la librairie scikit-learn ont un paramètre "class_weight" qui est par défaut initialiser à None, donnant alors des poids égaux pour toutes les classes. En mettant ce paramètre à "balanced", nous pouvons indiquer à scikit-learn de calculer automatiquement les poids pour chaque classe. Il est aussi possible de donner à scikit-learn un dictionnaire où les poids ont été entrés manuellement.

Résultats

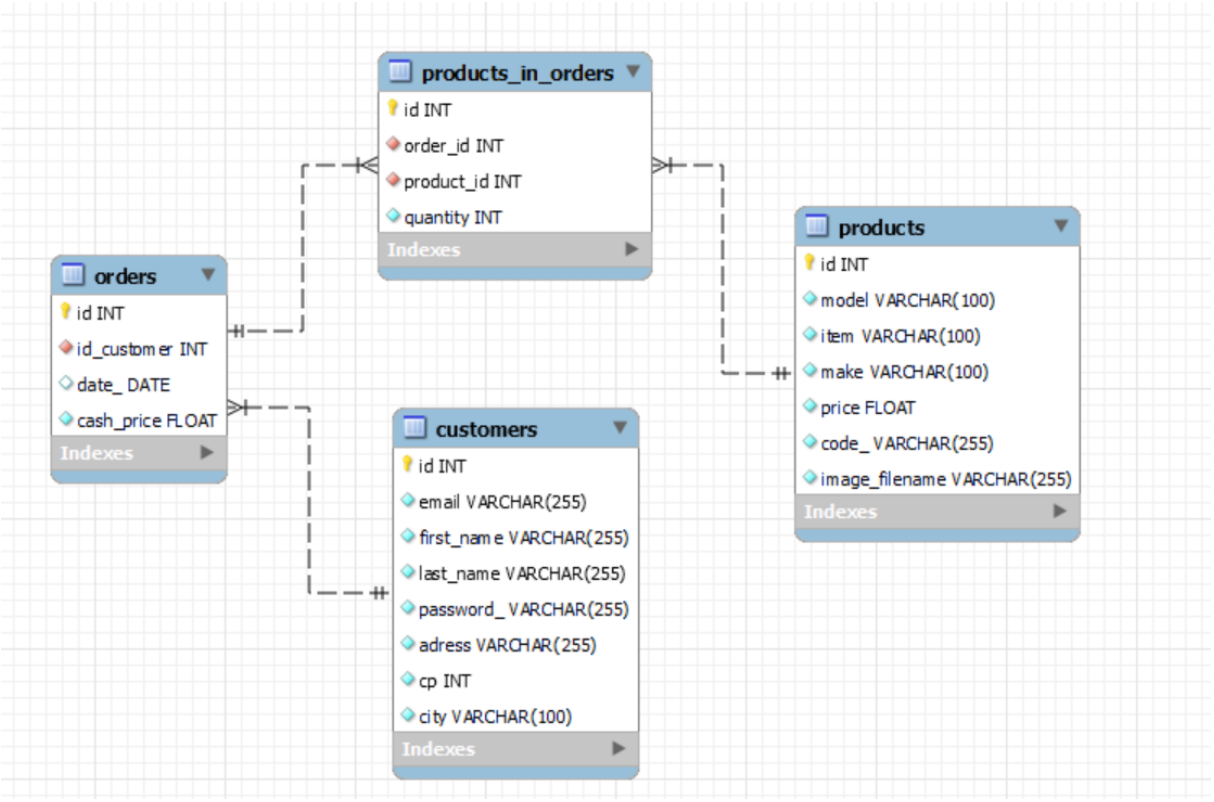
	class_weight = None	class_weight = « balanced »
RandomForestClassifier	0.184	0.120
GradientBoostingClassifier	0.126	
HistGradientBoostingClassifier	0.171	0.141
AdaBoostClassifier	0.090	
BaggingClassifier	0.159	
ExtraTreesClassifier	0.165	0.142

29	**24 janvier 2023 13:00**	**ThomasJoncourt**	**0,1745**
30	27 février 2023 19:40	totof	0,1734
31	7 avril 2023 10:39	Squirrel & AntoineDP	0,1731
32	7 avril 2023 17:18	tlecornec & cleroy	0,1727
33	13 avril 2023 10:40	Tom2	0,1700
34	20 février 2023 14:02	VictorHoffmann	0,1692
35	28 mars 2023 14:20	AdamEmz	0,1689
36	15 février 2023 16:40	raoulaur@gmail.com	0,1681
37	9 avril 2023 17:37	SimonLeR & hugo.leveque	0,1676
38	10 mars 2023 14:46	VictorHoffmann2	0,1675
39	28 mars 2023 14:45	Khobby	0,1668
40	27 mars 2023 18:04	Pierre_Joly	0,1652
41	14 février 2023 17:12	neodelphis	0,1649
42	19 février 2023 02:00	tontontondavid	0,1626
43	29 mars 2023 00:26	Lokhy	0,1599
44	9 avril 2023 20:53	h-anhluong & hamyluong	0,1578
45	**-**	**benchmark**	**0,1574**
46	22 janvier 2023 20:48	Quentin	0,1569

PR-AUC2=0,14PR-AUC$_2$=0,1745

Création d'une base de données relationnelle

Pour adapter le jeu de données afin d'être utilisé par une application de boutique en ligne, je l'ai transformé en une base de données SQL.



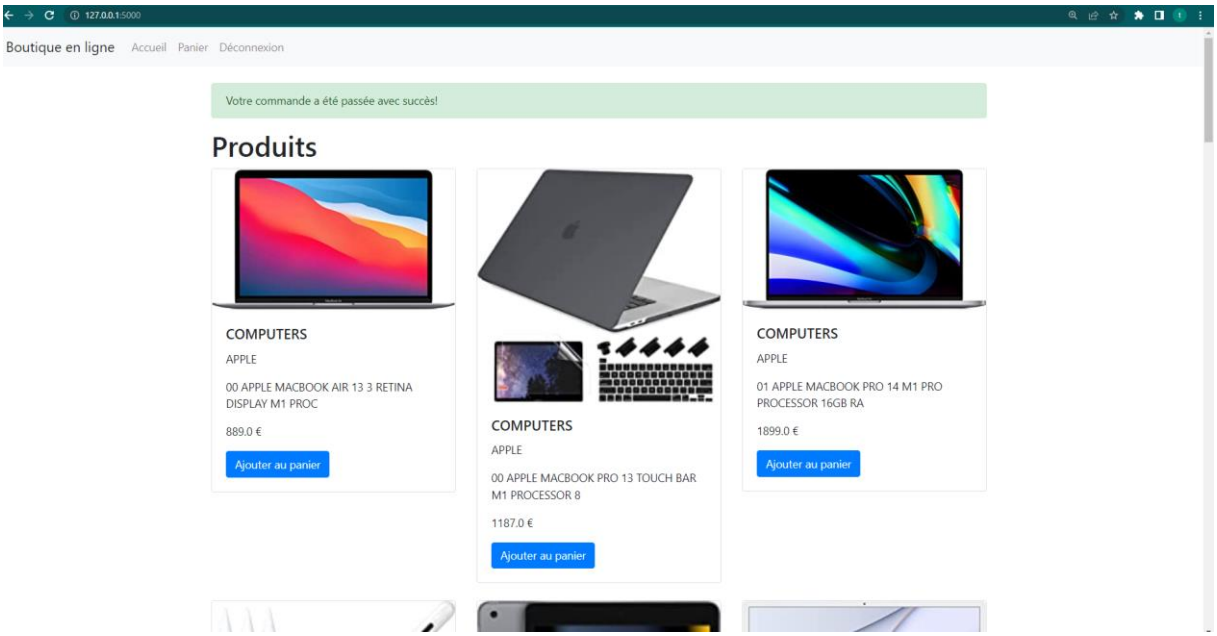
Le script SQL de création de la base de données est donné en Annexe 1.

Le script d'import des données dans la table «products» est donné en Annexe 2.

Application

Afin de simuler une utilisation pratique de notre modèle de détection de fraude, j'ai créé une application Flask qui sera relié à notre base de données SQL. Les informations de l'utilisateur et ses commandes seront stockées dans la base puis traité pour calculer la probabilité de fraude sur chaque commande.

Le script du fichier main.py de l'application est donné en annexe 3



Conclusion

Le score de précision moyenne obtenue est supérieur à celui du benchmark, j'ai donc réussi à proposer une solution plus performante sur la détection de fraude. Le classement de l'algorithme parmi tous ceux proposer dans le challenge est plutôt correct même s'il existe apparemment de bien meilleures solutions.

Les solutions pour essayer de rééquilibrer le dataset ne semble pas fonctionner. Je ne sais pas si c'est une mauvaise utilisation de ma part ou si ce dataset n'est tout simplement pas adapté au rééquilibrage. On peut supposer que dans notre cas précis, vu que le taux de fraude est une observation réelle, il ne doit pas être modifié pour ne pas s'éloigné de la réalité. Le rééquilibrage serait donc une solution adaptée à d'autre cas de machine Learning de classification où la proportion de chaque classe n'est pas associée à un phénomène réel comme des datasets de classification d'image par exemple.

Il est certainement possible d'obtenir de meilleurs résultats avec des modèles plus complexe comme des modèles de Deep Learning notamment.

Annexe

Annexe 1 : Script SQL de création de la base de données

```sql
drop database if exists bnp2;
-- Crée la base de données
CREATE DATABASE bnp2;
-- Utilise la base de données
USE bnp2;
-- Crée la table des produits
CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    model VARCHAR(100) NOT NULL,
    item VARCHAR(100) NOT NULL,
    make VARCHAR(100) NOT NULL,
    price FLOAT NOT NULL,
    code_ VARCHAR(255) NOT NULL,
    image_filename VARCHAR(255) NOT NULL
);
```

```sql
-- Crée la table des produits
CREATE TABLE customers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    password_ VARCHAR(255) NOT NULL,
    adress VARCHAR(255) NOT NULL,
    cp INT NOT NULL,
    city VARCHAR(100) NOT NULL
);
-- Crée la table des commandes
CREATE TABLE orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_customer INT NOT NULL,
    date_ DATE,
    cash_price FLOAT NOT NULL,
    FOREIGN KEY (id_customer) REFERENCES customers(id)
);
```

```sql
-- Crée la table des éléments de commande (order_items)
CREATE TABLE products_in_orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    FOREIGN KEY (order_id) REFERENCES orders(id),
    FOREIGN KEY (product_id) REFERENCES products(id)
);
```

Annexe 2 : Script d'import des données dans la table « products »

```python
train = pd.read_csv("/content/drive/MyDrive/Challenges/bnp_dataset/X_train.csv")

models = []
for i in range(1,25):
  models += train[f"model{i}"].values.tolist()

  df_models = pd.DataFrame({"model":pd.Series(models).value_counts().index,"count":pd.Series(models).value_counts().values})

df_models["make"] = None
df_models["item"] = None
df_models["price"] = None
df_models["good_code"] = None

for row in df_models.index:
  model = df_models["model"][row]
  mask = train[[f"model{i}" for i in range(1,25)]].isin([model]).any(axis=1)
  for i in range(1,25):
    if train[mask].iloc[0][f"model{i}"] == model:
      df_models["make"][row] = train[mask].iloc[0][f'make{i}']
      df_models["item"][row] = train[mask].iloc[0][f'item{i}']
      df_models["price"][row] = train[mask].iloc[0][f'cash_price{i}']
      df_models["good_code"][row] = train[mask].iloc[0][f'goods_code{i}']
      break

df_models["code_"] = df_models["good_code"]
df_models["image_filename"] = df_models["model"] + ".jpg"
df_models = df_models[["model", "item", "make", "price", "code_", "image_filename"]]

# Connect to the MySQL database
cnx = mysql.connector.connect(user='root', password='root', host='localhost', database='bnp2')


sql = "INSERT INTO products (model, item, make, price, code_, image_filename) VALUES (%s, %s, %s, %s, %s, %s)"
val = [(df_models["model"][row], df_models["item"][row], df_models["make"][row], str(df_models["price"][row]), df_models["code_"][row], df_models["image_filename"][row]) for row in df_models.iloc[0:20].index]

# Create a cursor object
cursor = cnx.cursor()

cursor.executemany(sql, val)

cnx.commit()
```

Annexe 3 : Script de fichier main.py de l'application Flask

```python
from flask import Flask, render_template, request, redirect, url_for, flash, session
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, login_user, logout_user, login_required, current_user
from models import db, Customers, Products, Orders, ProductsInOrder
from werkzeug.security import check_password_hash
from werkzeug.security import generate_password_hash

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root:root@localhost/bnp2'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.secret_key = 'your_secret_key'

db.init_app(app)

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

@login_manager.user_loader
def load_user(user_id):
    return Customers.query.get(int(user_id))

# Routes à créer ici (index, register, login, etc.)

@app.route('/')
def index():
    products = Products.query.all()
    return render_template('index.html', products=products)
```

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    if request.method == 'POST':
        email = request.form['email']
        first_name = request.form['first_name']
        last_name = request.form['last_name']
        password = request.form['password']
        adress = request.form['adress']
        cp = request.form['cp']
        city = request.form['city']

        # Vérifiez si l'utilisateur existe déjà
        user = Customers.query.filter_by(email=email).first()
        if user:
            flash('Cet email est déjà utilisé. Veuillez en utiliser un autre.', 'danger')
            return redirect(url_for('register'))

        hashed_password = generate_password_hash(password)
        new_customer = Customers(email=email, first_name=first_name, last_name=last_name, password_=hashed_password, adress=adress, cp=cp, city=city)
        db.session.add(new_customer)
        db.session.commit()

        flash('Inscription réussie! Vous pouvez maintenant vous connecter.', 'success')
        return redirect(url_for('login'))

    return render_template('register.html')
```

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    next_page = request.args.get('next', url_for('index'))

    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        remember_me = request.form.get('remember_me')

        user = Customers.query.filter_by(email=email).first()

        if user and check_password_hash(user.password_, password):
            login_user(user, remember=remember_me)
            session['cart'] = {}  # Initialise le panier
            return redirect(next_page)
        else:
            flash('Identifiants incorrects. Veuillez réessayer.', 'danger')

    return render_template('login.html', next=next_page)

@app.route('/logout')
def logout():
    logout_user()
    return redirect(url_for('index'))
```

```python
@app.route('/cart', methods=['GET', 'POST'])
#@login_required
def cart():
    if request.method == 'POST':
        # Créez la commande pour le client actuel
        order = Orders(id_customer=current_user.id, cash_price=0)
        db.session.add(order)
        db.session.commit()

        # Parcourez les éléments du panier et ajoutez-les à la commande
        cart_items = request.form.getlist('product_id')
        cart_quantities = request.form.getlist('quantity')
        total_price = 0

        for i, product_id in enumerate(cart_items):
            product = Products.query.get(product_id)
            quantity = int(cart_quantities[i])
            total_price += product.price * quantity

            # Ajoutez l'élément de commande à la base de données
            order_item = ProductsInOrder(order_id=order.id, product_id=product_id, quantity=quantity)
            db.session.add(order_item)

        # Mettez à jour le prix total de la commande
        order.cash_price = total_price
        db.session.commit()

        flash("Votre commande a été passée avec succès!", "success")
        return redirect(url_for('index'))

    cart = session.get('cart') or {}
    cart_items = []
    for product_id, quantity in cart.items():
        product = Products.query.get(product_id)
        cart_items.append({'product': product, 'quantity': quantity})

    return render_template('cart.html', cart_items=cart_items)

if __name__ == '__main__':
    app.run(debug=True)
```