<코드 설명서 - 웹 (Django)>

| 웹 개발(Django) 소스 코드 설명서 | |
| --- | --- |
| 작성자 | 김동훈 |
| 작성 일자 | 2024/10/11 |
| 팀명 | Sniffers |

| 순서 | 작업 내용 | 페이지 |
| --- | --- | --- |
| 1 | manage.py | 3 ~ 4 |
| 2 | models.py | 5 ~ 6 |
| 3 | fraud/urls.py | 7 |
| 4 | serializers.py | 8 |
| 5 | signals.py | 9 ~ 11 |
| 6 | apps.py | 12 ~ 13 |
| 7 | index\_transactions.py | 14 ~ 15 |
| 8 | migrations/ | 16 |
| 0001\_initial.py | 16 ~ 17 |
| 0002\_delete\_frauddata.py | 18 ~ 19 |
| 0003\_frauddata.py | 20 ~ 21 |
| 0004\_auto\_20240913\_0347.py | 22 ~ 23 |
| 0005\_remove\_frauddata\_transaction\_date.py | 24 ~ 25 |
| 0006\_transaction.py | 26 ~ 27 |
| 0007\_transaction\_fraud\_prediction.py | 28 ~ 29 |
| 9 | database\_routers.py | 30 ~ 31 |
| 10 | settings.py | 32 ~ 34 |
| 11 | views.py | 35 |
| Index 함수 | 35 |
| test\_fraud 함수 | 36 ~ 39 |
| upload\_data 함수 | 40 ~ 41 |
| load\_model 함수 | 42 ~ 43 |
| create\_user\_index 함수 | 44 ~ 45 |
| dashboard 함수 | 46 ~ 48 |
| login\_view 함수 | 49 ~ 50 |
| logout\_view 함수 | 51 |
| signup 함수 | 52 ~ 53 |
| upload\_form\_sub 함수 | 54 ~ 58 |
| dashboard\_sub 함수 | 59 ~ 64 |

# <manage.py>

#!/usr/bin/env python3

"""Django's command-line utility for administrative tasks."""

import os

import sys

def main():

"""Run administrative tasks."""

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'sniffers.settings')

try:

from django.core.management import execute\_from\_command\_line

except ImportError as exc:

raise ImportError(

"Couldn't import Django. Are you sure it's installed and "

"available on your PYTHONPATH environment variable? Did you "

"forget to activate a virtual environment?"

) from exc

execute\_from\_command\_line(sys.argv)

if \_\_name\_\_ == '\_\_main\_\_':

main()

| 기능 | |
| --- | --- |
| Django 프로젝트의 명령어 인터페이스로, 프로젝트 실행 및 관리 작업을 담당한다. 서버 실행, 마이그레이션 적용, 관리 명령 실행 등이 포함된다. | |
| 코드 설명 | |
| execute\_from\_command\_line | Django 설정 파일을 환경 변수로 지정한 후, execute\_from\_command\_line 함수로 명령을 실행한다. |
| 입력 | |
| 명령어 입력 (python manage.py <command> 형식) | |
| 출력 | |
| 명령 실행 결과 (서버 실행 메시지, 마이그레이션 결과 등) | |
| 매개변수 | |
| 없음 | |
| 예외 처리 | |
| Django가 설치되지 않았거나, 환경 변수가 설정되지 않은 경우 ImportError 발생. | |
| 알고리즘 | |
| 환경 변수를 설정하고 명령어를 처리하는 간단한 흐름으로 구성. | |

# <models.py>

from django.db import models

class FraudData(models.Model):

transaction\_id = models.IntegerField(primary\_key=True)

amount = models.FloatField()

customer\_id = models.IntegerField()

merchant\_id = models.IntegerField()

lat = models.IntegerField(default = 0)

long = models.IntegerField(default = 0)

def \_\_str\_\_(self):

return str(self.transaction\_id)

from django.db import models

class Transaction(models.Model):

transaction\_id = models.CharField(max\_length=100)

amount = models.DecimalField(max\_digits=10, decimal\_places=2)

customer\_id = models.CharField(max\_length=100)

merchant\_id = models.CharField(max\_length=100)

lat = models.FloatField()

long = models.FloatField()

fraud\_prediction = models.FloatField(null=True)

def \_\_str\_\_(self):

return self.transaction\_id

| 기능 | |
| --- | --- |
| 거래 데이터를 저장하기 위한 모델을 정의하며, Fraud Detection System에서 중요한 역할을 한다. 주요 필드로는 거래 ID, 금액, 고객 ID, 상점 ID, 위도, 경도가 있다. | |
| 코드 설명 | |
| transaction\_id | 각 거래의 고유 식별자 |
| amount | 거래 금액 |
| customer\_id | 고객 식별자 |
| merchant\_id | 상점 식별자 |
| lat, long | 위도, 경도 |
| 입력 | |
| 각 필드에 맞는 데이터를 삽입하여 거래 정보를 저장 | |
| 출력 | |
| 거래 데이터 객체의 문자열 표현(\_\_str\_\_)로 transation\_id 반환 | |
| 매개변수 | |
| 없음 | |
| 예외 처리 | |
| Django ORM이 데이터 유효성을 관리 | |
| 알고리즘 | |
| Django ORM을 사용하여 데이터베이스에 데이터 저장 및 조회 | |

# <fraud/urls.py>

from django.urls import path

from . import views

urlpatterns = [

path('test-fraud/', views.test\_fraud, name='test\_fraud'),

path('upload/', views.upload\_form, name='upload\_form'),

path('dashboard/', views.dashboard, name='dashboard'), path('index/', views.index, name='index'), path('dashboard2/', views.dashboard2, name='dashboard2'), path('upload\_form2/', views.upload\_form2, name='upload\_form2'),

]

| 기능 | |
| --- | --- |
| 프로젝트에서 사용되는 URL 패턴을 정의하고, 각 URL 요청에 대해 어떤 뷰가 처리할지 지정한다. | |
| 코드 설명 | |
| test-fraud/ | 사기 탐지 테스트 API 호출 경로 |
| upload/ | 데이터 업로드 페이지 |
| dashboard/ | 대시보드 페이지 |
| 입력 | |
| 각 URL에 맞는 HTTP 요청 | |
| 출력 | |
| 뷰의 처리 결과를 반환 | |
| 매개변수 | |
| path | URL 패턴과 그에 대응하는 뷰 함수 |
| 예외 처리 | |
| 없는 URL로 접근 시 404 에러 발생 | |
| 알고리즘 | |
| URL과 뷰를 매핑하는 알고리즘 | |

# <serializers.py>

from rest\_framework import serializers

from .models import FraudData

class FraudDataSerializer(serializers.ModelSerializer):

class Meta:

model = FraudData

fields = '\_\_all\_\_'

| 기능 | |
| --- | --- |
| FraudData 모델을 직렬화 및 역직렬화하는 데 사용된다. 이를 통해 데이터베이스 모델을 JSON 형식으로 변환하여 API 응답으로 보낼 수 있고, API에서 받은 JSON 데이터를 데이터베이스 모델로 변환하여 저장할 수 있다. | |
| 코드 설명 | |
| FraudDataSerializer | FraudData 모델을 직렬화하는 클래스 |
| fields = ‘\_\_all\_\_’ | 모델의 모든 필드를 직렬화할 수 있도록 지정 |
| 입력 | |
| 모델 인스턴스 또는 JSON 데이터 | |
| 출력 | |
| 직렬화된 JSON 데이터 또는 모델 인스턴스 | |
| 매개변수 | |
| 없음 | |
| 예외 처리 | |
| 입력 데이터의 형식이 잘못되면 ValidationError 발생 | |
| 알고리즘 | |
| Django REST Framework의 기본 직렬화 알고리즘을 사용 | |
| 추가 설명 | |
| 이 파일은 REST API를 구현할 때 필수적인 역할을 한다. 모델 데이터를 직렬화하거나 역직려로하하여 API와 상호작용할 수 있게 한다. | |

# <signals.py>

from django.db import connection

from django.contrib.auth.models import User

from django.db.models.signals import post\_save, post\_delete

from django.dispatch import receiver

from django.db.utils import OperationalError

@receiver(post\_save, sender = User)

def create\_user\_database(sender, instance, created, \*\*kwargs):

if created:

db\_name = f'user\_{instance.username}\_db'

try:

with connection.cursor() as cursor:

cursor.execute(f"CREATE DATABASE {db\_name}")

print(f"Database {db\_name} created.")

except OperationalError:

print(f"Database {db\_name} already exists.")

@receiver(post\_delete, sender = User)

def delete\_user\_database(sender, instance, \*\*kwargs):

db\_name = f'user\_{instance.username}\_db'

with connection.cursor() as cursor:

cursor.execute(f"DROP DATABASE IF EXISTS {db\_name}")

print(f"Database {db\_name} deleted.")

from django.db.models.signals import post\_delete

from django.dispatch import receiver

from django.contrib.auth.models import User

from elasticsearch import Elasticsearch

@receiver(post\_delete, sender=User)

def delete\_elasticsearch\_index(sender, instance, \*\*kwargs):

es = Elasticsearch([{'host': '10.0.1.7', 'port': 9200}])

index\_name = f'user\_{instance.username}\_transactions'

try:

es.indices.delete(index=index\_name)

print(f"Successfully deleted index: {index\_name}")

except Exception as e:

print(f"Failed to delete index: {str(e)}")

| 기능 | |
| --- | --- |
| 데이터가 저장되거나 삭제될 때 특정 작업을 자동으로 수행하는 신호를 정의한다. 이를 통해 모델의 생명주기 동안 다양한 작업을 자동화할 수 있다. | |
| 사용자가 생성되거나 삭제될 때 자동으로 개인 데이터베이스 및 Elasticsearch 인덱스를 생성 또는 삭제하는 기능을 구현한다. | |
| 사용자 등록 시 자동으로 개인 데이터베이스를 만들고, 사용자가 삭제될 때 해당 데이터베이스와 Elasticsearch 인덱스를 삭제한다. | |
| 코드 설명 | |
| connection | Django에서 데이터베이스에 접근하는 객체이다. SQL 쿼리를 직접 실행할 때 사용된다. |
| User | Django 내장 사용자 모델로, 사용자가 생성되거나 삭제될 때를 처리한다. |
| post\_save, post\_delete | 모델의 인스턴스가 저자오디거나 삭제된 후에 신호를 보내는 장치이다. |
| receiver | 특정 신호(post\_save, post\_delete)를 수신하고 이를 처리할 함수를 지정한다. |
| OperationalError | 데이터베이스 연산 중 발생할 수 있는 오류를 처리한다. |
| create\_user\_database | 사용자가 처음 생성될 때 자동으로 호출되는 함수이다. |
| instance.username | 새로운 사용자마다 고유한 데이터베이스 이름을 생성한다. |
| cursor.execute | SQL 명령을 실행하여 새 데이터베이스를 생성한다. (데이터베이스가 이미 존재할 경우 OperationalError 예외를 처리한다.) |
| delete\_user\_database | 사용자가 삭제될 때 호출되는 함수이다. |
| DROP DATABASE IF EXISTS | 해당 사용자의 데이터베이스를 삭제한다. (데이터베이스가 존재하지 않는 경우에도 에러 없이 안전하게 처리된다.) |
| delete\_elasticsearch\_index | 사용자가 삭제될 때 해당 사용자의 Elasticsearch 인덱스를 삭제한다. |
| Elasticsearch | Elasticsearch 클러스터에 연결하기 위해 사용된다. |
| es.indices.delete | 해당 사용자의 인덱스를 삭제하는 명령어이다. (예외가 발생할 경우 오류 메시지를 출력한다.) |
| 입력 | |
| instance | 삭제되거나 생성된 사용자 모델 인스턴스 |
| 출력 | |
| 사용자 데이터베이스 및 Elasticsearch 인덱스 생성/삭제 결과를 콘솔에 출력 | |
| 매개변수 | |
| sender | 신호를 보낸 모델 클래스(여기서는 User) |
| instance | 생성되거나 삭제된 사용자 인스턴스 |
| created | 사용자 인스턴스가 새로 생성된 경우 True |
| 예외 처리 | |
| 데이터베이스 또는 인덱스 생성 중 이미 존재할 경우 OperationalError 및 기타 예외를 처리 | |
| 알고리즘 | |
| 사용자 생성 후 개인 데이터베이스 생성 | |
| 사용자 삭제 후 데이터베이스와 Elasticsearch 인덱스 삭제 | |
| 추가 설명 | |
| 이 코드는 각 사용자의 데이터와 Elasticsearch 인덱스를 개별적으로 관리하여 데이터 격리와 관리 용이성을 제공한다. | |

# <apps.py>

from django.apps import AppConfig

class FraudConfig(AppConfig):

default\_auto\_field = 'django.db.models.BigAutoField'

name = 'fraud'

def ready(self):

import fraud.signals

| 기능 | |
| --- | --- |
| Django 애플리케이션의 설정을 정의한다. 프로젝트의 fraud 애플리케이션을 관리하고, 앱이 시작될 때 필요한 초기 설정을 수행한다. | |
| 코드 설명 | |
| AppConfig | Django 앱의 설정을 정의하는 기본 클래스로, fraud 앱의 설정을 관리한다. |
| FraudConfig | fraud 애플리케이션의 설정 클래스 |
| default\_auto\_field | 기본적으로 모델의 자동 생성 필드를 BigAutoField로 설정하여, ID 필드에 대한 기본값을 설정한다. |
| name = ‘fraud’ | 앱의 이름을 ‘fraud’로 정의한다. |
| ready() | Django가 앱을 초기화할 때 실행되는 메서드이다.  signals.py 파일을 불러와 앱에서 정의된 신호를 활성화한다. 이를 통해 post\_save와 post\_delete 등의 이벤트가 올바르게 처리된다. |
| 입력 | |
| 없음 | |
| 출력 | |
| 없음 | |
| 매개변수 | |
| 없음 | |
| 예외 처리 | |
| ready() 메서드에서 signals.py를 찾을 수 없을 경우 ImportError가 발생할 수 있다. | |
| 알고리즘 | |
| Django 애플리케이션이 시작될 때 앱 설정을 초기화하고 signals.py를 로드하는 과정이 진행된다. | |
| 추가 설명 | |
| fraud 앱이 시작될 때 필요한 설정을 담당한다. 특히 ready() 메서드에서 신호(signals)를 불러오는 부분이 중요한 역할을 하며, 사용자 생성 및 삭제 시 데이터베이스와 Elasticsearch 인덱스를 처리하는 기능을 활성화한다. | |

# <index\_transactions.py>

# fraud/management/commands/index\_transactions.py

from django.core.management.base import BaseCommand

from fraud.models import Transaction

from elasticsearch import Elasticsearch

import pickle

class Command(BaseCommand):

help = 'Index transactions to Elasticsearch'

def handle(self, \*args, \*\*options):

es = Elasticsearch([{'host': '10.0.1.7', 'port': 9200}]) # Elasticsearch URL

# Load the model

with open('fraud/models/FDS\_model\_isolationForest.pkl', 'rb') as f:

model = pickle.load(f)

for transaction in Transaction.objects.all():

doc = {

'transaction\_id': transaction.transaction\_id,

'amount': transaction.amount,

'customer\_id': transaction.customer\_id,

'merchant\_id': transaction.merchant\_id,

'lat': transaction.lat,

'long': transaction.long,

'fraud\_prediction': transaction.fraud\_prediction,

}

es.index(index='transactions', id=transaction.id, body=doc)

self.stdout.write(self.style.SUCCESS('Successfully indexed transactions.'))

| 기능 | |
| --- | --- |
| Django 관리 명령어로, 데이터베이스에 저장된 거래 정보를 Elasticsearch에 색인화한다. 또한, 사기 탐지 모델을 불러와 거래 데이터에 적용한다. | |
| 코드 설명 | |
| from django.core.management.base BaseCommand | Django의 관리 명령어를 만들기 위한 기본 클래스이다. |
| from fraud.models import Transaction | fraud 애플리케이션의 거래 데이터를 나타내는 모델이다. |
| from elasticsearch import Elasticsearch | Elasticsearch 클러스터에 연결하기 위한 라이브러리이다. |
| import pickle | 사기 탐지 모델을 불러오기 위한 Python의 직렬화 도구이다. |
| es = Elasticsearch() | Elasticsearch 클러스터에 연결한다. |
| with open(‘.pkl’, ‘rb’) as f:  model = pickle.load(f) | IsolationForest 모델이 저장된 FDS\_model\_isolationForest.pkl 파일을 불러온다.  이 모델을 통해 거래 데이터의 사기 가능성을 평가할 수 있다. |
| 입력 | |
| 데이터베이스에 저장된 Transaction 객체들 | |
| 출력 | |
| 각 거래가 Elasticsearch에 색인화되며, 완료 후 성공 메시지를 출력 | |
| 매개변수 | |
| 없음 | |
| 예외 처리 | |
| 기본 예외 처리 없이 진행되며, Elasticsearch 연결 문제나 파일 불러오기 오류가 발생할 수 있다. | |
| 알고리즘 | |
| 데이터베이스에서 거래 데이터를 불러와서, 각 거래 정보를 Elasticsearch에 색인한다. | |
| 추가 설명 | |
| 이 명령어는 관리 콘솔에서 ‘python manage.py index\_transactions’ 명령으로 실행할 수 있으며, 거래 데이터가 Elasticsearch에 색인된 후 Kibana와 같은 도구에서 시각화할 수있다. | |

# <migrations/>

### <0001\_initial.py>

# Generated by Django 3.2.12 on 2024-09-12 06:36

from django.db import migrations, models

class Migration(migrations.Migration):

initial = True

dependencies = [

]

operations = [

migrations.CreateModel(

name='FraudData',

fields=[

('transaction\_id', models.IntegerField(primary\_key=True, serialize=False)),

('amount', models.FloatField()),

('customer\_id', models.IntegerField()),

('merchant\_id', models.IntegerField()),

('transaction\_data', models.DateTimeField()),

],

),

]

| 기능 | |
| --- | --- |
| FraudData 모델을 처음으로 데이터베이스에 생성하는 작업을 수행한다. Django 프로젝트를 처음 시작할 때 데이터베이스 스키마를 초기화하는 데 사용된다. | |
| 코드 설명 | |
| from django.db import migrations | Django의 데이터베이스 마이그레이션 시스템을 관리한다. |
| from django.db import models | Django ORM에서 사용되는 모델 필드를 정의하는 데 사용된다. |
| initial = True | 마이그레이션 파일이 프로젝트의 첫 번째 마이그레이션임을 나타낸다. |
| migrations.CreateModel | 새로운 모델(테이블)을 데이터베이스에 생성하는 작업을 수행한다. |
| name = ‘FraudData’,  fields = [...] | FraudData 모델은 각 거래 데이터를 나타내는 테이블이다. |
| 입력 | |
| 새로운 FraudData 모델 생성 요청 | |
| 출력 | |
| 데이터베이스에 FraudData 테이블이 생성된다. | |
| 매개변수 | |
| 모델 필드 정의 시 사용하는 다양한 매개변수 (예: primary\_key = True) | |
| 예외 처리 | |
| 이미 동일한 이름의 테이블이 존재하는 경우 마이그레이션 시 오류가 발생할 수 있다. | |
| 알고리즘 | |
| Django 마이그레이션 시스템을 통해 테이블과 필드가 정의된 대로 데이터베이스에 적용된다. | |
| 추가 설명 | |
| 이 마이그레이션 파일은 프로젝트 초기 설정 시 데이터베이스 스키마를 정의하는 중요한 작업을 수행하며, 거래 데이터를 저장하기 위한 기본 테이블을 생성한다. | |

### <0002\_delete\_frauddata.py>

# Generated by Django 3.2.12 on 2024-09-13 02:16

from django.db import migrations

class Migration(migrations.Migration):

dependencies = [

('fraud', '0001\_initial'),

]

operations = [

migrations.DeleteModel(

name='FraudData',

),

]

| 기능 | |
| --- | --- |
| 이전에 생성된 FraudData 모델을 삭제하는 작업을 수행한다. 데이터 모델이 변경되거나, 더 이상 필요 없는 경우 이를 데이터베이스에서 제거하는 데 사용된다. | |
| 코드 설명 | |
| from django.db import migrations | Django의 마이그레이션 시스템을 사용하여 데이터베이스 스키마를 관리한다. |
| dependencies = [(‘fraud’, ‘0001\_initial’),] | 0001\_initial 마이그레이션 파일이 이 마이그레이션 파일의 선행 조건으로 설저오디어 있다. 즉, 0001\_initial이 먼저 적용된 후 이 마이그레이션 파일이 실행된다. |
| migrations.DeleteModel(name = ‘FraudData’,),] | FraudData 모델을 데이터베이스에서 삭제한다. 이 작업을 통해 데이터베이스에서 FraudData 테이블이 삭제된다. |
| 입력 | |
| 삭제할 FraudData 모델 | |
| 출력 | |
| FraudData 테이블이 데이터베이스에서 삭제된다. | |
| 매개변수 | |
| name = ‘FraudData’ | 삭제할 모델의 이름을 지정한다. |
| 예외 처리 | |
| 삭제하려는 모델이 데이터베이스에 존재하지 않을 경우, 마이그레이션시 오류가 발생할 수 있다. | |
| 알고리즘 | |
| Django 마이그레이션 시스템은 DeleteModel 작업을 통해 데이터베이스에서 지정된 모델을 삭제한다. | |
| 추가 설명 | |
| 이 마이그레이션 파일은 FraudData 모델을 삭제하며, 이는 데이터베이스 스키마 변경 시 매우 중요한 작업이다. 모델이 더 이상 필요 없거나, 새로운 데이터 모델로 대체될 때 사용된다. | |

### <0003\_frauddata.py>

# Generated by Django 3.2.12 on 2024-09-13 03:33

from django.db import migrations, models

class Migration(migrations.Migration):

initial = True

dependencies = [

('fraud', '0002\_delete\_frauddata'),

]

operations = [

migrations.CreateModel(

name='FraudData',

fields=[

('transaction\_id', models.IntegerField(primary\_key=True, serialize=False)),

('amount', models.FloatField()),

('customer\_id', models.IntegerField()),

('merchant\_id', models.IntegerField()),

('transaction\_date', models.DateTimeField()),

],

),

]

| 기능 | |
| --- | --- |
| 삭제된 FraudData 모델을 다시 생성하는 작업을 수행한다. 이는 이전에 ‘0002\_delete\_frauddata’ 파일을 통해 삭제된 ‘FraudData’ 모델을 다시 데이터베이스에 복구하는 역할을 한다. | |
| 코드 설명 | |
| from django.db import migrations | 데이터베이스 스키마를 변경하고 관리하기 위한 Django 마이그레이션 시스템 |
| from django.db import models | Django ORM에서 사용되는 모델 필드 및 데이터베이스 스키마를 정의한다. |
| dependencies = [(‘fraud’, ‘0002\_delete\_frauddata’),] | 이 마이그레이션 파일은 ‘0002\_delete\_frauddata’ 이후에 실행되며, 삭제된 ‘FraudData’ 모델을 복구하기 위한 작업이다. |
| migrations.CreateModel(  name = ‘FraudData’,  fields = [...],) | ‘FraudData’라는 이름의 새로운 모델을 데이터베이스에 생성한다. |
| 입력 | |
| 새로운 ‘FraudData’ 모델 생성 요청 | |
| 출력 | |
| 데이터베이스에 ‘FraudData’ 모델 생성 요청 | |
| 매개변수 | |
| 각 필드에 대한 다양한 속성 (primary\_key, serialize, DateTimeField 등) | |
| 예외 처리 | |
| 동일한 이름의 테이블이 이미 존재할 경우, 마이그레이션 시 오류가 발생할 수 있다. | |
| 알고리즘 | |
| ‘CreateModel’ 작업을 통해 삭제된 ‘FraudData’ 테이블을 데이터베이스에 다시 생성한다. | |
| 추가 설명 | |
| 이 마이그레이션 파일은 ‘FraudData’ 모델을 다시 생성하며, 거래 데이터를 저장할 수 있는 구조를 복구한다. 이 파일은 데이터베이스 구조 변경 또는 실수로 삭제된 모델을 다시 복원할 때 유용하다. | |

### <0004\_auto\_20240913\_0347.py>

# Generated by Django 3.2.12 on 2024-09-13 03:47

from django.db import migrations, models

class Migration(migrations.Migration):

dependencies = [

('fraud', '0003\_frauddata'),

]

operations = [

migrations.AddField(

model\_name='frauddata',

name='lat',

field=models.IntegerField(default=0),

),

migrations.AddField(

model\_name='frauddata',

name='long',

field=models.IntegerField(default=0),

),

]

| 기능 | |
| --- | --- |
| ‘FraudData’ 모델에 두 개의 필드, ‘lat(위도)’와 ‘long(경도)’을 추가한다. 이로써 거래 데이터에 위치 정보를 추가하여 보다 세부적인 분석이 가능하게 합니다. | |
| 코드 설명 | |
| from django.db import migrations | 데이터베이스 스키마를 변경하고 관리하기 위한 Django 마이그레이션 시스템 |
| from django.db import models | Django ORM에서 사용되는 모델 필드 및 데이터베이스 스키마를 정의한다. |
| dependencies = [(‘fraud’, ‘0003\_frauddata’),] | 이 마이그레이션 파일은 ‘0003\_frauddata’ 이후에 실행된다. 즉, ‘FraudData’ 모델이 먼저 생성된 후에 위치 정보 필드를 추가한다. |
| migrations.AddField(  model\_name = ‘frauddata’,  name = ‘lat’,  field = models.IntegerField(default = 0),), … | ‘FraudData’ 모델에 새로운 필드를 추가하는 작업이다. 위도(lat)와 경도(long) 정보의 기본값은 0으로 설정된다. |
| 입력 | |
| 두 개의 새로운 필드(lat 및 long)를 추가하는 요청 | |
| 출력 | |
| 데이터베이스의 ‘FraudData’ 테이블에 위도와 경도 필드가 추가된다. | |
| 매개변수 | |
| ‘lat’ 및 ‘long’ 필드 | 기본값이 0인 정수형 필드 |
| 예외 처리 | |
| 이미 필드가 존재하는 경우 마이그레이션 오류가 발생할 수 있다. | |
| 알고리즘 | |
| ‘AddField’ 작업을 통해 데이터베이스에 새로운 필드를 추가한다. | |
| 추가 설명 | |
| 이 마이그레이션 파일은 ‘FraudData’ 모델에 위치 정보를 추가하여, 거래의 발생 위치를 저장할 수 있도록 구조를 확장한다. 이를 통해 거래 데이터를 분석할 때 위치 기반의 정보를 활용할 수 있다. | |

### <0005\_remove\_frauddata\_transaction\_date.py>

# Generated by Django 3.2.12 on 2024-09-13 03:50

from django.db import migrations

class Migration(migrations.Migration):

dependencies = [

('fraud', '0004\_auto\_20240913\_0347'),

]

operations = [

migrations.RemoveField(

model\_name='frauddata',

name='transaction\_date',

),

]

| 기능 | |
| --- | --- |
| ‘FraudData’ 모델에서 ‘transaction\_date’ 필드를 제거한다. 이 필드는 거래 날짜와 시간을 나타내는 필드였으며, 더 이상 필요하지 않거나 데이터 모델에서 제거하는 것이 적절하다고 판단된 경우 삭제된다. | |
| 코드 설명 | |
| from django.db import migrations | 데이터베이스 스키마를 관리하고 변경하는 Django 마이그레이션 시스템이다. |
| dependencies = [(‘fraud’, ‘0004\_auto\_20240913\_0347’),] | 이 마이그레이션 파일은 ‘0004\_auto\_20240913\_0347’ 이후에 실행된다. 즉, 위치 정보 필드가 추가된 후에 ‘transaction\_date’ 필드를 제거한다. |
| migrations.RemoveField(  model\_name = ‘frauddata’,  name = ‘transaction\_date’,), | ‘FraudData’ 모델에서 지정된 필드를 제거하는 작업이다. |
| 입력 | |
| ‘transaction\_date’ 필드를 제거하는 요청 | |
| 출력 | |
| ‘FraudData’ 테이블에서 ‘transaction\_date’ 필드가 제거된다. | |
| 매개변수 | |
| model\_name = ‘FraudData’ | 필드를 제거할 모델 |
| name = ‘transaction\_date’ | 제거할 필드 이름 |
| 예외 처리 | |
| 제거하려는 필드가 존재하지 않는 경우 마이그레이션 시 오류가 발생할 수 있다. | |
| 알고리즘 | |
| ‘RemoveField’ 작업을 통해 데이터베이스에서 해당 필드를 삭제한다. | |
| 추가 설명 | |
| 이 마이그레이션 파일은 데이터베이스 구조를 간소화하거나 변경할 때 사용되며, 거래 날짜 필드를 더 이상 사용하지 않기 때문에 이를 데이터베이스에서 제거한다. 데이터 분석이나 비즈니스 로직에서 해당 필드가 필요 없어진 상황에서 주로 사용된다. | |

<0006\_transaction.py>

# Generated by Django 3.2.12 on 2024-09-23 01:08

from django.db import migrations, models

class Migration(migrations.Migration):

dependencies = [

('fraud', '0005\_remove\_frauddata\_transaction\_date'),

]

operations = [

migrations.CreateModel(

name='Transaction',

fields=[

('id', models.BigAutoField(auto\_created=True, primary\_key=True, serialize=False, verbose\_name='ID')),

('transaction\_id', models.CharField(max\_length=100)),

('amount', models.DecimalField(decimal\_places=2, max\_digits=10)),

('customer\_id', models.CharField(max\_length=100)),

('merchant\_id', models.CharField(max\_length=100)),

('lat', models.FloatField()),

('long', models.FloatField()),

],

),

]

| 기능 | |
| --- | --- |
| 새로운 ‘Transaction’ 모델을 데이터베이스에 생성한다. 이 모델은 거래 정보를 보다 구체적으로 저장하며, 이전에 사용하던 ‘FraudData’ 모델과는 별개의 테이블로 새로운 구조를 정의한다. | |
| 코드 설명 | |
| from django.db import migrations | 데이터베이스 스키마를 변경하고 관리하기 위한 Django 마이그레이션 시스템 |
| from django.db import models | Django ORM에서 사용되는 모델 필드를 정의하는 모듈이다. |
| dependencies = [(‘fraud’, ‘0005\_remove\_frauddata\_transaction\_date’),] | 이 마이그레이션 파일은 ‘0005\_remove\_frauddata\_transaction\_date’ 이후에 실행되며, ‘FraudData’ 모델의 변경 후에 ‘Transaction’ 모델을 새로 생성하는 작업을 수행한다. |
| migrations.CreateModel(  name = ‘Transaction’,  fields = [...],) | 새로운 ‘Transaction’ 모델을 데이터베이스에 생성한다. |
| 입력 | |
| 새로운 ‘Transaction’ 모델을 생성하는 요청 | |
| 출력 | |
| 데이터베이스에 ‘Transaction’ 테이블이 생성된다. | |
| 매개변수 | |
| transaction\_id, customer\_id, merchant\_id | 최대 길이 100자의 문자열 |
| amount | 소수점 둘째 자리까지 저장할 수 있는 10자리 소수형 필드 |
| lat, long | 각각 위도와 경도를 저장하는 부동 소수점 필드 |
| 예외 처리 | |
| 테이블이 이미 존재하거나 다른 충돌이 발생할 경우 마이그레이션 시 오류가 발생할 수 있다. | |
| 알고리즘 | |
| Django의 ‘CreateModel’ 작업을 통해 데이터베이스에 새로운 테이블을 생성한다. | |
| 추가 설명 | |
| 이 마이그레이션 파일은 거래 정보를 보다 명확하고 구체적으로 저장할 수 있도록 ‘Transaction’ 모델을 생성한다. 이 모델은 거래 ID, 금액, 고객 ID, 상점 ID, 위도, 경도 정보를 포함하며, 이후 거래 데이터 분석 및 사기 탐지와 같은 작업에서 중요한 역할을 한다. | |

### <0007\_transaction\_fraud\_prediction.py>

# Generated by Django 3.2.12 on 2024-09-23 08:29

from django.db import migrations, models

class Migration(migrations.Migration):

dependencies = [

('fraud', '0006\_transaction'),

]

operations = [

migrations.AddField(

model\_name='transaction',

name='fraud\_prediction',

field=models.FloatField(null=True),

),

]

| 기능 | |
| --- | --- |
| ‘Transaction’ 모델에 새로운 ‘fraud\_prediction’ 필드를 추가한다. 이 필드는 거래 데이터에 대해 사기 여부를 예측한 값을 저장하는 데 사용된다. | |
| 코드 설명 | |
| from django.db import migrations | Django 마이그레이션 시스템을 사용해 데이터베이스 스키마를 변경하고 관리한다. |
| from django.db import models | Django ORM에서 모델 필드를 정의하는 데 사용된다. |
| dependencies = [(‘fraud’, ‘0006\_transaction’),] | 이 마이그레이션 파일은 ‘0006\_transaction’ 이후에 실행되며, 기존의 ‘Transaction’ 테이블에 새로운 필드를 추가한다. |
| migrations.AddField(  model\_name = ‘transaction’  name = ‘fraud\_prediction’,  field = models.FloatField(null = True),), | ‘Transaction’ 모델에 새로운 필드를 추가하는 작업이다.  ‘fraud\_prediction’은 사기 여부를 예측하는 부동 소수점 필드로, 예측 결과 값을 저장한다. 이 필드는 초기값으로 ‘null’을 허용한다. |
| 입력 | |
| ‘fraud\_prediction’ 필드를 ‘Transaction’ 모델에 추가하는 요청 | |
| 출력 | |
| 데이터베이스의 ‘Transaction’ 테이블에 새로운 ‘fraud\_prediction’ 필드가 추가된다. | |
| 매개변수 | |
| fraud\_prediction | 부동 소수점 필드(FloatField), 기본적으로 null 값을 허용하여 필드에 값이 없을 수 있도록 설정 |
| 예외 처리 | |
| 추가하려는 필드가 이미 존재하거나 데이터 충돌이 발생할 경우 마이그레이션 시 오류가 발생할 수 있다. | |
| 알고리즘 | |
| ‘AddField’ 작업을 통해 데이터베이스에 새로운 필드를 추가한다. | |
| 추가 설명 | |
| 이 마이그레이션 파일은 사기 탐지 시스템에서 중요한 역할을 하는 ‘fraud\_prediction’ 필드를 추가한다. 이 필드는 각 거래에 대해 사기 가능성을 예측한 결과를 저장하며, 이후 데이터 분석 및 시각화에서 중요한 정보를 제공한다. | |

### <database\_routers.py>

class UserDatabaseRouter:

'''

특정 사용자의 모든 데이터 베이스 기능을 컨트롤 하는 라우터

'''

def db\_for\_read(self, model, \*\*hints):

user = hints.get('user')

if user:

return f'user\_{user.username}\_db'

return 'default'

def db\_for\_write(self, model, \*\*hints):

user = hints.get('user')

if user:

return f'user\_{user.username}\_db'

return 'default'

def allow\_relation(self, obj1, obj2, \*\*hints):

if hasattr(obj1, 'user') and hasattr(obj2, 'user'):

if obj1.user == obj2.user:

return True

return None

def allow\_migrate(self, db, app\_label, model\_name = None, \*\*hints):

if db == 'default':

return True

return False

| 기능 | |
| --- | --- |
| 각 사용자에 대한 데이터베이스 기능을 관리한다. 현재 로그인한 사용자의 데이터베이스로 읽기, 쓰기 작업을 라우팅하며, 데이터베이스 간의 관계 및 마이그레이션을 제어하는 역할을 한다. | |
| 코드 설명 | |
| def db\_for\_read(self, model, \*\*hints):  … | 읽기 작업을 수행할 때, 현재 로그인한 사용자의 데이터베이스로 라우팅한다. |
| user = hints.get(‘user’) | ‘hints’로 전달된 ‘user’ 객체를 사용하여 사용자의 데이터베이스를 결정한다. 사용자가 존재하면 ‘user\_{username}\_db’로 라우팅되고, 그렇지 않으면 기본 데이터베이스(default)가 사용된다. |
| def db\_for\_write(self, model, \*\*hints):  … | 쓰기 작업을 수행할 때, 현재 로그인한 사용자의 데이터베이스로 라우팅한다. 이 로직은 읽기 작업과 유사하게 작동하며, 사용자의 데이터베이스로 쓰기 작업을 수행한다. |
| def allow\_relation(self, obj1, obj2, \*\*hints):  … | 두 객체가 동일한 사용자에게 속할 경우에만 관계를 허용한다. 즉, 각 사용자의 데이터베이스 간 관계를 제한하여 데이터 간 충돌을 방지한다. |
| def allow\_migrate(self, db, app\_label, model\_name = None, \*\*hints):  … | 마이그레이션은 기본 데이터베이스에서만 허용된다. 이를 통해 사용자별 데이터베이스에는 공통 테이블이 아닌 사용자 개별 데이터만 저장하도록 제한한다. |
| 입력 | |
| hints | 사용자의 정보를 포함하는 힌트. 예를 들어, 읽기 및 쓰기 작업에 사용자 데이터베이스를 사용하기 위한 ‘user’ 객체를 전달받는다. |
| 출력 | |
| 사용자의 데이터베이스로 읽기/쓰기 작업이 라우팅되고, 동일한 사용자 간의 데이터 관계만 허용됩니다. | |
| 매개변수 | |
| user | 현재 작업 중인 사용자를 가리키며, 각 사용자의 데이터베이스로 라우팅하는 데 사용된다. |
| 예외 처리 | |
| 사용자가 존재하지 않거나 잘못된 데이터베이스가 지정된 경우, 기본 데이터베이스(default)로 라우팅된다. | |
| 알고리즘 | |
| 읽기 및 쓰기 작업을 수행할 때, 현재 사용자의 데이터베이스로 라우팅하고, 사용자 간의 데이터 관계 및 마이그레이션을 제어한다. | |
| 추가 설명 | |
| 이 라우터는 각 사용자마다 고유한 데이터베이스를 사용할 수 있도록 설계되었다. 이를 통해 사용자 간의 데이터 격리를 보장하며, 사용자 개별 데이터베이스를 효율적으로 관리할 수 있다. | |

### <settings.py>

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

SECRET\_KEY = 'django-insecure-56qlwa9+y1kxo2yk&v$0rk^j#c763(x2gd8okcdfla05mx=gh@'

DEBUG = True

ALLOWED\_HOSTS = ['\*']

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'fraud',

'rest\_framework',

]

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

#'NAME': BASE\_DIR / 'db.sqlite3',

'NAME': 'transaction',

'USER': 'fds001',

'PASSWORD': 'fds001',

'HOST': '10.0.1.7',

'PORT': '3306'

}

}

EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL\_HOST = '10.0.1.8'

EMAIL\_PORT = 25

EMAIL\_USE\_TLS = False

DEFAULT\_FROM\_EMAIL = 'noreply@sniffers.site'

STATIC\_URL = '/static/'

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, "fraud", "static"),

]

DATABASE\_ROUTERS = ['sniffers.database\_routers.UserDatabaseRouter']

| 기능 | |
| --- | --- |
| Django 프로젝트의 전반적인 설정을 정의하는 파일로, 보안, 데이터베이스 연결, 정적 파일 경로, 애플리케이션 구성, 국제화 설정 등을 포함하고 있다. 이를 통해 애플리케이션의 다양한 환경 설정을 관리할 수 있다. | |
| 코드 설명 | |
|  |  |
| 입력 | |
| ‘settings.py’ 파일은 프로젝트의 여러 환경 설정을 정의하는 입력을 포함한다. 예를 들어, 데이터베이스 연결 정보, 정적 파일 경로, 이메일 서버 설정 등이 입력으로 포함된다. | |
| 출력 | |
| Django 애플리케이션은 ‘settings.py’ 파일의 설정을 기반으로 동작하며, 올바르게 구성된 경우 각 기능이 정상적으로 실행된다. 예를 들어, 정적 파일이 올바르게 제공되고, 데이터베이스 연결이 성공적으로 이루어진다. | |
| 매개변수 | |
| DEBUG | 개발 모드에서는 True, 운영 모드에서는 False로 설정한다. |
| ALLOWED\_HOSTS | 서버에 접근할 수 있는 허용된 호스트 목록을 정의한다. |
| DATABASES | 데이터베이스 연결에 필요한 정보이다. |
| EMAIL\_BACKEND | 이메일 전송을 위한 백엔드 설정이다. |
| 예외 처리 | |
| ALLOWED\_HOSTS에 정의되지 않은 호스트에서 접속 시, Django는 ‘DisallowedHost’ 예외를 발생시킨다. | |
| DATABASES 설정이 잘못되었을 경우, 데이터베이스 연결 오류가 발생할 수 있다. | |
| 알고리즘 | |
| 이 파일은 특별한 알고리즘을 포함하지 않으며, 프로젝트 전반에 걸친 환경 설정을 정의하는 역할을 한다. | |
| 추가 설명 | |
| ‘settings.py’ 파일은 Django 프로젝트의 핵심 설정 파일로, 이 파일을 통해 다양한 환경 변수와 애플리케이션 설정이 관리된다. 운영 환경과 개발 환경에 따라 적절한 설정이 필요하며, 특히 보안과 관련된 ‘SECRET\_KEY’, ‘DEBUG’, ‘ALLOWED\_HOSTS’ 등의 설정에 주의가 필요하다. | |

# <views.py>

### <index 함수>

def index(request):

return render(request, 'fraud/index.html')

| 기능 | |
| --- | --- |
| Fraud Detection System의 메인 페이지를 렌더링하는 함수이다. | |
| 코드 설명 | |
| render(request, ‘fraud/index.html’) | Django의 뷰 함수에서 템플릿을 렌더링하여 HTML 응답을 반환한다. 이 경우, ‘fraud/index.html’ 템플릿을 렌더링하여 사용자에게 보여준다. |
| request | Django의 ‘HttpRequest’ 객체로, 클라이언트로부터 받은 요청 정보를 담고 있다. |
| 입력 | |
| request | 사용자의 HTTP 요청 정보가 포함된 객체 |
| 출력 | |
| 메인 페이지(즉, ‘fraud/index.html’ 템플릿)가 사용자에게 렌더링된 HTML 응답으로 반환된다. | |
| 매개변수 | |
| request | 사용자로부터 받은 요청 객체 |
| 예외 처리 | |
| 이 함수는 특별한 예외 처리를 포함하지 않는다. 템플릿이 없을 경우 ‘TemplateDoesNotExist’ 예외가 발생할 수 있다. | |
| 알고리즘 | |
| 이 함수는 특정 알고리즘을 포함하지 않고, 주어진 템플릿을 렌더링하는 역할만 수행한다. | |
| 추가 설명 | |
| 이 함수는 Fraud Detection System의 메인 페이지를 호출할 때 사용된다. | |

### <test\_fraud 함수>

@login\_required

def test\_fraud(request):

model\_path = 'fraud/models/FDS\_model\_isolationForest.pkl'

try:

with open(model\_path, 'rb') as f:

model = pickle.load(f)

except FileNotFoundError:

return HttpResponse("Model file not found.")

transactions = Transaction.objects.all()

if not transactions:

return HttpResponse("No transaction data found.")

label\_encoders = {

'customer\_id': LabelEncoder(),

'merchant\_id': LabelEncoder()

}

predictions = []

customer\_ids = [transaction.customer\_id for transaction in transactions]

merchant\_ids = [transaction.merchant\_id for transaction in transactions]

label\_encoders['customer\_id'].fit(customer\_ids)

label\_encoders['merchant\_id'].fit(merchant\_ids)

for transaction in transactions:

transaction\_data = np.array([

0,

transaction.amount,

label\_encoders['customer\_id'].transform([transaction.customer\_id])[0],

label\_encoders['merchant\_id'].transform([transaction.merchant\_id])[0],

transaction.lat,

transaction.long

])

random\_features = [random.uniform(0, 1) for \_ in range(17)]

complete\_data = np.concatenate([transaction\_data, random\_features])

prediction = model.predict(complete\_data.reshape(1, -1))

predictions.append(prediction[0])

transaction.fraud\_prediction = prediction[0]

transaction.save()

complete\_data = np.concatenate([transaction\_data, random\_features, prediction])

context = {

'predictions': predictions,

}

return render(request, 'test\_fraud.html', context)

| 기능 | |
| --- | --- |
| 데이터베이스에 저장된 트랜잭션 데이터를 불러와 사기 여부를 예측하는 머신러닝 모델을 실행한다. 이 예측 결과는 데이터베이스에 저장되고, 최종적으로 사용자에게 예측 결과를 보여준다. | |
| 코드 설명 | |
| @login\_required | 사용자가 로그인되어 있어야만 ‘test\_fraud’ 함수에 접근할 수 있게 만드는 데코레이터이다. 로그인이 되어 있지 않으면 로그인 페이지로 리다이렉트된다. |
| model\_path | 사기 예측에 사용될 IsolationForest 모델 파일의 경로를 지정한다. |
| model = pickle.load(f) | 모델 파일을 바이너리로 읽어와 메모리로 로드한다. 만약 모델 파일이 존재하지 않으면 FileNotFoundError 예외가 발생하며, 이 경우 “Model file not found.” 메시지를 반환한다. |
| transactions = Transaction.objects.all() | 데이터베이스에서 모든 트랜잭션 데이터를 조회한다. 만약 트랜잭션 데이터가 없으면 “No transaction data found.” 메시지를 반환한다. |
| label\_encoders = {  ‘customer\_id’ : LabelEncoder(),  ‘merchant\_id’ : LabelEncoder()} | 문자열인 ‘customer\_id’와 ‘merchant\_id’ 값을 머신러닝 모델에서 사용할 수 있도록 숫자로 변환하는 인코더이다. |
| customer\_ids = [transaction.customer\_id for transaction in transactions]  merchant\_ids = [transaction.merchant\_id for transaction in transactions]  label\_encoders[‘customer\_id’].fit(customer\_ids)  label\_encoders[‘merchant\_id’].fit(merchant\_ids) | 각 ‘customer\_id’와 ‘merchant\_id’를 인코더에 맞춰 변환 준비를 완료한다. |
| for transaction in transactions:  transaction\_data = np.array([...]) | 각 트랜잭션 데이터에서 ‘amount’, ‘customer\_id’, ‘merchant\_id’, ‘lat’, ‘long’ 값을 추출하여 배열로 변환한다. |
| random\_features = [random.uniform(0, 1) for \_ in range(17)]  complete\_data = np.concatenate([transaction\_data, random\_features]) | 17개의 무작위 특성을 생성하고 트랜잭션 데이터와 결합하여 모델에 입력한다. |
| prediction = model.predict(complete\_data.reshape(1, -1))  predictions.append(prediction[0]) | IsolationForest 모델을 사용하여 사기 여부를 예측한다. 예측 결과는 ‘predictions’ 리스트에 추가된다. |
| transaction.fraud\_prediction = prediction[0]  transaction.save() | 예측된 사기 여부를 트랜잭션 객체의 ‘fraud\_prediction’ 필드에 저장하고, 데이터베이스에 저장한다. |
| context = {‘predictions’ : predictions,}  return render(request, ‘test\_fraud.html’, context) | 예측 결과를 ‘context’에 담아 ‘test\_fraud.html’ 템플릿을 렌더링한다. 사용자에게 예측 결과가 표시된다. |
| 입력 | |
| request | 사용자로부터 받은 HTTP 요청 객체 |
| 출력 | |
| ‘test\_fraud.html’ 템플릿에 사기 예측 결과를 전달하여 사용자에게 결과를 보여준다. | |
| 매개변수 | |
| request | 사용자로부터 받은 HTTP 요청 객체 |
| 예외 처리 | |
| 모델 파일이 없을 경우 FileNotFoundError 예외가 발생하며, “Model file not found.” 메시지를 반환한다. | |
| 트랜잭션 데이터가 없을 경우 “No transaction data found.” 메시지를 반환한다. | |
| 알고리즘 | |
| IsolationForest 알고리즘을 사용하여 트랜잭션 데이터의 사기 여부를 예측한다. 각 트랜잭션의 주요 특성에 추가적으로 무작위 특성을 결합하여 모델에 입력한 후 예측한다. | |
| 추가 설명 | |
| 이 함수는 Fraud Detection System의 핵심 기능 중 하나로, 트랜잭션 데이터의 사기 여부를 예측하여 데이터베이스에 저장하고 사용자에게 결과를 보여준다. | |

### <upload\_data 함수>

@login\_required

def upload\_data(request):

if request.method == 'POST':

uploaded\_file = request.FILES['file']

df = pd.read\_csv(uploaded\_file)

request.session['uploaded\_data'] = df.to\_json()

return redirect('dashboard2')

return render(request, 'fraud/upload\_form.html')

| 기능 | |
| --- | --- |
| 사용자가 업로드한 CSV 파일을 처리하고, 해당 데이터를 세션에 저장한 후 대시보드 페이지로 리다이렉트하는 역할을 한다. | |
| 코드 설명 | |
| @login\_required | 사용자가 로그인되어 있어야만 ‘test\_fraud’ 함수에 접근할 수 있게 만드는 데코레이터이다. 로그인이 되어 있지 않으면 로그인 페이지로 리다이렉트된다. |
| if request.method == ‘POST’: | POST 요청을 통해 파일 업로드가 이루어진 경우에만 이 로직을 실행한다. |
| uploaded\_file = request.FILES[‘file’] | 사용자가 업로드한 CSV 파일을 처리한다. |
| fd = pd.read\_csv(uploaded\_file) | pandas 라이브러리를 사용하여 업로드된 CSV 파일을 데이터프레임으로 변환한다. |
| request.session[‘uploaded\_data’] = df.to\_json() | 업로드된 데이터를 JSON 형식으로 세션에 저장한다. 이는 이후 대시보드 페이지에서 데이터를 사용할 수 있도록 하기 위함이다. |
| return redirect(‘dashboard2’) | 업로드된 데이터를 처리한 후, ‘dashboard2’ 페이지로 리다이렉트한다. 사용자는 대시보드 페이지로 이동하여 업로드한 데이터를 시각화할 수 있다. |
| return render(request, ‘fraud/upload\_form.html’) | 사용자가 파일을 업로드하지 않았을 때, ‘upload\_form.html’ 템플릿을 렌더링 하여 파일 업로드 폼을 보여준다. |
| 입력 | |
| request | 사용자의 HTTP 요청 객체. 주로 POST 요청을 통해 CSV 파일이 업로드된다. |
| 출력 | |
| 업로드된 데이터를 세션에 저장하고 대시보드 페이지로 리다이렉트한다. 사용자가 파일을 업로드하지 않았을 경우, 파일 업로드 폼을 렌더링한다. | |
| 매개변수 | |
| request | HTTP 요청 객체로, 업로드된 파일과 세션 데이터를 처리하는 데 사용된다. |
| 예외 처리 | |
| 별도의 예외 처리를 포함하지 않으나, 파일이 잘못되었거나 읽기에 실패할 경우 에러가 발생할 수 있다. | |
| 알고리즘 | |
| CSV 파일을 업로드한 후, 데이터를 pandas로 처리하여 JSON 형식으로 변환하고 세션에 저장한 후 대시보드 페이지로 리다이렉트하는 단순한 흐름을 따른다. | |
| 추가 설명 | |
| 이 함수는 사용자가 데이터를 업로드하고, 해당 데이터를 처리 및 저장하여 후속 처리와 대시보드 시각화를 위한 중요한 역할을 한다. | |

### <load\_model 함수>

def load\_model():

model\_path = 'fraud/models/FDS\_model\_isolationForest.pkl'

with open(model\_path, 'rb') as model\_file:

model = pickle.load(model\_file)

return model

| 기능 | |
| --- | --- |
| 이상 거래 예측에 사용되는 머신러닝 모델을 파일에서 로드하여 반환하는 역할을 한다. | |
| 코드 설명 | |
| model\_path = ‘...’ | 머신러닝 모델이 저장된 파일의 경로를 지정한다. 이 경로는 IsolationForest 모델을 저장한 경로로 설정되어 있다. |
| with open(model\_path, ‘rb’) as model\_file:  model = pickle.load(model\_file) | 모델 파일을 바이너리 읽기 모드(rb)로 열고, pickle을 사용해 파일에서 머신러닝 모델을 메모리로 불러온다. 이 과정에서 모델 파일은 직렬화된 데이터로 저장되어 있으며, 이 데이터를 복원하여 사용할 수 있도록 한다. |
| return model | 로드된 모델을 반환한다. 이 모델은 이후 트랜잭션 데이터를 기반으로 사기 여부를 예측하는 데 사용된다. |
| 입력 | |
| 없음 | |
| 출력 | |
| 로드된 머신러닝 모델 객체를 반환한다. | |
| 매개변수 | |
| 없음 | |
| 예외 처리 | |
| 모델 파일이 존재하지 않거나 손상되었을 경우 FileNotFoundError 또는 pickle.UnpicklingError와 같은 예외가 발생할 수 있다. | |
| 알고리즘 | |
| 모델 파일을 읽어서 메모리로 로드한 후 반환하는 단순한 흐름을 따른다. 모델 자체는 미리 학습된 IsolationForest 알고리즘이다. | |
| 추가 설명 | |
| 이 함수는 이상 거래 예측 시스템의 핵심적인 부분 중 하나로, 저장된 모델을 불러와 예측을 수행하는 데 사용된다. 함수가 실행되면 머신러닝 모델을 빠르게 로드하여 다른 함수나 메서드에서 사용할 수 있다. | |

### <create\_user\_index 함수>

def create\_user\_index(es, index\_name):

if not es.indices.exists(index=index\_name):

es.indices.create(index=index\_name, body={

"mappings": {

"properties": {

"amount": {"type": "float"},

"customer\_id": {"type": "float"},

"merchant\_id": {"type": "float"},

"lat": {"type": "float"},

"long": {"type": "float"},

"fraud\_prediction": {"type": "long"},

"timestamp": {"type": "date"},

"user\_id": {

"type": "text",

"fields": {

"keyword": {"type": "keyword", "ignore\_above": 256}

}

}

}

}

})

print(f"Index {index\_name} is successfully created")

| 기능 | |
| --- | --- |
| Elasticsearch에 사용자별 트랜잭션 데이터를 저장할 인덱스를 생성하는 역할을 한다. 인덱스가 존재하지 않을 경우 새로 생성하며, 이덱스의 매핑(properties)을 설정한다. | |
| 코드 설명 | |
| if not es.indices.exists(index = index\_name): | 주어진 인덱스(index\_name)가 Elasticsearch에 이미 존재하는지 확인한다. 인덱스가 존재하지 않으면 새로운 인덱스를 생성하는 다음 단계로 넘어간다. |
| es.indices.create(index = index\_name, body = {  “mappings”: {  “properties”: {  …  }}}) | 새로운 인덱스를 생성하며, ‘index\_name’에 해당하는 이름으로 인덱스를 만든다.  ‘mappings’ 부분에서는 인덱스에 저장될 데이터의 구조를 정의한다. |
| 입력 | |
| es | Elasticsearch 클라이언트 객체 |
| index\_name | 생성할 인덱스의 이름 |
| 출력 | |
| 새로운 인덱스가 Elasticsearch에 성공적으로 생성되면 콘솔에 성공 메시지가 출력된다. | |
| 매개변수 | |
| es | Elasticsearch와의 연결을 관리하는 객체 |
| index\_name | 생성할 인덱스의 이름. 각 사용자의 트랜잭션 데이터를 저장할 인덱스 이름이 된다. |
| 예외 처리 | |
| Elasticsearch 연결 문제나 권한 문제로 인해 인덱스 생성에 실패할 경우, elasticsearch.exceptions 관련 예외가 발생할 수 있다. | |
| 알고리즘 | |
| 주어진 인덱스가 존재하는지 확인하고, 존재하지 않을 경우 해당 인덱스를 생성하는 간단한 로직을 사용한다. 인덱스 생성 시 데이터 필드의 매핑을 명시적으로 설정하여 데이터가 올바른 타입으로 저장되도록 한다. | |
| 추가 설명 | |
| 이 함수는 사용자의 트랜잭션 데이터를 저장할 인덱스를 자동으로 생성하는 기능을 담당한다. 각 사용자의 인덱스는 이름을 통해 구분되며, 데이터의 구조(필드 타입 등)를 미리 정의하여 Elasticsearch에 효율적으로 저장하고 검색할 수 있도록 설계되어 있다. | |

### <dashboard 함수>

@login\_required

def dashboard(request):

db\_name = f'user\_{request.user.username}\_db'

if db\_name not in settings.DATABASES:

settings.DATABASES[db\_name] = {

'ENGINE': 'django.db.backends.mysql',

'NAME': db\_name,

'USER': 'fds001',

'PASSWORD': 'fds001',

'HOST': '10.0.1.7',

'PORT': '3306',

}

try:

with connections[db\_name].cursor() as cursor:

cursor.execute('SELECT amount, customer\_id, merchant\_id, lat, `long`, fraud\_prediction FROM fraud\_transaction')

transactions = cursor.fetchall()

transaction\_data = [{

'amount': row[0],

'customer\_id': row[1],

'merchant\_id': row[2],

'lat': row[3],

'long': row[4],

'fraud\_prediction': row[5],

} for row in transactions]

kibana\_url = f"http://211.188.51.137:5601/app/dashboards#/view/78e0ff70-7b56-11ef-9b66-13538d49ff90?\_a=(filters:!((meta:(alias:!n,disabled:!f,key:user\_id,negate:!f,params:(query:'{request.user.username}'),type:phrase)))"

return render(request, 'fraud/dashboard.html', {

'transactions': transaction\_data,

'kibana\_url': kibana\_url #

})

except OperationalError as e:

return HttpResponse(f"Database connection for {db\_name} failed: {str(e)}")

| 기능 | |
| --- | --- |
| 사용자별 데이터베이스에서 트랜잭션 데이터를 가져와 대시보드 페이지에 표시하는 역할을 한다. 또한, 사용자의 Kibana 대시보드를 볼 수 있는 URL을 생성하여 템플릿으로 전달한다. | |
| 코드 설명 | |
| db\_name = f’user\_{request.user.username}\_db’ | 현재 로그인한 사용자의 이름을 기반으로 사용자별 데이터베이스 이름을 동적으로 생성한다. 예를 들어, 사용자 이름이 ‘john’이라면 데이터베이스 이름은 ‘user\_john\_db’가 된다. |
| if db\_name not in settings.DATABASES:  settings.DATABASES[db\_name] = {...} | 데이터베이스 설정이 존재하지 않으면, 사용자별 데이터베이스 정보를 동적으로 추가한다. 이 설정은 MySQL 데이터베이스에 연결하기 위한 설정이다. |
| with connections[db\_name].cursor() as cursor:  cursor.execute(‘SELECT …’)  transactions = cursor.fetchall() | 해당 사용자 데이터베이스에 연결하여 ‘fraud\_transaction’ 테이블에서 트랜잭션 데이터를 조회한다. |
| transaction\_data = [{  …} for row in transactions] | 조회한 트랜잭션 데이터를 JSON 형식으로 변환하여, 템플릿으로 전달하기 위해 준비한다. 각 트랜잭션의 주요 필드를 포함한다. |
| kibana\_url = f”http://<IP addr>/app/dashboards#/view/<ID>\_a(filters:...,params:(query:’{request.user.username}’)) | 사용자의 ‘user\_id’를 기반으로 Kibana에서 해당 사용자의 데이터를 필터링할 수 있는 URL을 동적으로 생성한다. 이를 통해 사용자는 본인의 데이터만 볼 수 있게 된다. |
| return render(request, ‘fraud/dashboard.html’, {  ‘transactions’: transaction\_data,  ‘kibana\_url’: kibana\_url}) | 트랜잭션 데이터와 Kibana 대시보드 URL을 ‘dashboard.html 템플릿으로 전달하여 렌더링한다. |
| except OperationalError as e:  return HttpResponse(f”Database connection for {db\_name} failed: {str(e)}”) | 데이터베이스 연결 오류가 발생하면, 에러 메시지를 포함한 HTTP 응답을 반환한다. |
| 입력 | |
| request | HTTP 요청 객체. 사용자의 로그인 정보와 데이터베이스 연결 설정이 포함되어 있다. |
| 출력 | |
| 사용자별 트랜잭션 데이터를 포함한 대시보드 페이지가 렌더링된다. 또한, Kibana 대시보드의 사용자별 필터링된 URL도 제공된다. | |
| 매개변수 | |
| request | 사용자가 보낸 HTTP 요청 객체 |
| 예외 처리 | |
| 데이터베이스 연결에 실패한 경우 OperationalError가 발생하며, 실패 메시지를 포함한 응답을 반환한다. | |
| 알고리즘 | |
| 사용자 데이터베이스에 연결하여 데이터를 조회하고, 조회된 데이터를 JSON 형식으로 변환한다. 이후, 사용자별로 필터링된 Kibana 대시보드 URL을 생성하여 대시보드 페이지로 렌더링하는 흐름이다. | |
| 추가 설명 | |
| 이 함수는 사용자별 데이터베이스와 Kibana 대시보드를 통해 각 사용자가 자신의 트랜잭션 데이터를 확인할 수 있게 하는 중요한 역할을 한다. 데이터베이스 연결 및 Kibana 필터링을 동적으로 처리하여 각 사용자에게 맞춤형 대시보드를 제공한다. | |

### <login\_view 함수>

def login\_view(request):

if request.method == 'POST':

form = AuthenticationForm(request, data=request.POST)

if form.is\_valid():

username = form.cleaned\_data.get('username')

password = form.cleaned\_data.get('password')

user = authenticate(username=username, password=password)

if user is not None:

login(request, user)

messages.success(request, f'Welcome, {username}!')

return redirect('home')

else:

messages.error(request, 'Invalid username or password.')

else:

messages.error(request, 'Invalid username or password.')

else:

form = AuthenticationForm()

return render(request, 'fraud/login.html', {'form': form})

| 기능 | |
| --- | --- |
| 사용자의 로그인 요청을 처리한다. 사용자가 입력한 로그인 정보를 검증하고, 유효한 경우 사용자를 로그인시키고 환영 메시지를 출력한다. 그렇지 않은 경우 오류 메시지를 출력한다. | |
| 코드 설명 | |
| if request.method == ‘POST’: | 사용자가 로그인 폼을 제출했을 때(POST 요청)만 이 코드 블록이 실행된다. GET 요청일 경우 로그인 폼을 렌더링한다. |
| form = AuthenticationForm(request, data = request.POST) | Django가 제공하는 인증 폼으로, 사용자 입력 데이터를 검증한다. |
| if form.is\_valid(): | 입력된 사용자명과 비밀번호가 유효한지 검사한다. 유효하지 않은 경우, 오류 메시지를 출력한다. |
| username = form.cleaned\_data.get(‘username’)  password = form.cleaned\_data.get(‘password’)  user = authenticate(username = username, password = password) | 폼에서 입력된 ‘username’과 ‘password’ 값을 가져온다.  ‘authenticate()’는 제공된 자격 증명(아이디와 비밀번호)으로 사용자가 인증될 수 있는지 확인한다. 유효한 사용자라면 사용자 객체를 반환하고, 그렇지 않으면 None을 반환한다. |
| login(request, user)  return redirect(‘home’) | 사용자가 유효한 경우, 사용자 세션을 생성하여 로그인 처리한다.  로그인 성공 시, ‘home’ 페이지로 리다이렉트 한다. |
| form = AuthenticationForm() | GET 요청일 경우 빈 AuthenticationForm을 생성하여 로그인 페이지를 렌더링한다. |
| 입력 | |
| request | 사용자가 보낸 HTTP 요청 객체 |
| 출력 | |
| 로그인 성공 시 사용자에게 환영 메시지가 표시되고, ‘home’ 페이지로 리다이렉트된다. | |
| 로그인 실패 시 로그인 페이지에서 오류 메시지가 출력된다. | |
| 매개변수 | |
| request | 사용자가 보낸 HTTP 요청 객체 |
| 예외 처리 | |
| 인증 실패 시 오류 메시지를 출력한다. | |
| ‘form.is\_valid()’를 통해 사용자의 입력 데이터가 유효하지 않을 경우 적절한 메시지가 표시된다. | |
| 알고리즘 | |
| 사용자의 아이디와 비밀번호가 POST 요청을 통해 전달되면, 이를 검증하고 유효한 경우 로그인 처리를 진행한다. 인증에 실패하면 오류 메시지를 출력하고 로그인 페이지로 되돌아간다. | |
| 추가 설명 | |
| 이 함수는 Django의 기본 인증 폼을 사용하여 로그인 기능을 구현하고 있다. 로그인 성공 시 환영 메시지를 출력하며, 실패 시 적절한 오류 메시지를 사용자에게 보여준다. | |

### <logout\_view 함수>

def logout\_view(request):

logout(request)

messages.success(request, 'You have successfully logged out.')

return redirect('home')

| 기능 | |
| --- | --- |
| 사용자를 로그아웃시키고, 로그아웃 성공 메시지를 출력한 후 홈 페이지로 리다이렉트하는 역할을 한다. | |
| 코드 설명 | |
| logout(request)  return redirect(‘home’) | ‘logout()’은 Django의 내장 함수로, 현재 로그인한 사용자의 세션을 종료하고 로그아웃을 처리한다.  로그아웃 처리 후, 사용자를 홈 페이지로 리다이렉트한다. ‘redirect()’ 함수는 지정된 URL 패턴으로 사용자를 이동시킨다. |
| 입력 | |
| request | HTTP 요청 객체. 이 객체는 로그아웃 처리와 메시지 추가를 위한 정보를 제공한다. |
| 출력 | |
| 사용자가 로그아웃된 후 홈 페이지로 리다이렉트된다. 성공적으로 로그아웃되었음을 알리는 메시지가 표시된다. | |
| 매개변수 | |
| request | 로그아웃 요청을 처리하는 HTTP 요청 객체 |
| 예외 처리 | |
| 없음 | |
| 알고리즘 | |
| 로그아웃 요청을 받아 사용자의 세션을 종료하고, 성공 메시지를 설정한 후 홈 페이지로 리다이렉트하는 간단한 로직을 따른다. | |
| 추가 설명 | |
| 이 함수는 Django의 기본적인 로그아웃 처리 방식을 따르고 있으며, 성공적인 로그아웃을 사용자에게 알리기 위해 메시지 기능을 활용하고 있다. | |

### <signup 함수>

def signup(request):

if request.method == 'POST':

form = UserCreationForm(request.POST)

if form.is\_valid():

user = form.save()

login(request, user)

return redirect('index')

else:

form = UserCreationForm()

return render(request, 'fraud/signup.html', {'form': form})

| 기능 | |
| --- | --- |
| 사용자의 회원가입 요청을 처리한다. 사용자가 회원가입 폼을 제출하면 입력 데이터를 검증하고, 유효한 경우 사용자를 데이터베이스에 저장하고 자동으로 로그인시킨 후 홈 페이지로 리다이렉트한다. | |
| 코드 설명 | |
| if request.method == ‘POST’: | 사용자가 회원가입 폼을 제출할 때만 이 부분이 실행된다. |
| form = UserCreationForm(request.POST) | Django가 제공하는 기본 회원가입 폼으로, 사용자가 제출한 데이터를 받아 폼 객체를 생성한다. |
| if form.is\_valid(): | 사용자가 입력한 데이터가 유효한지 검증한다. 입력 데이터가 유효할 경우, 새로운 사용자 계정이 생성된다. |
| user = form.save() | 유효한 데이터를 바탕으로 새로운 사용자를 데이터베이스에 저장한다. |
| login(request, user) | 회원가입 후 새로 생성된 사용자로 자동으로 로그인 시킨다. |
| return redirect(‘index’) | 회원가입과 자동 로그인이 성공한 후, 홈 페이지(index 페이지)로 리다이렉트한다. |
| else:  form = UserCreationForm() | GET 요청이거나 폼이 유효하지 않을 경우, 빈 회원가입 폼을 생성한다. 이 폼은 회원가입 페이지에 렌더링 된다. |
| return render(request, ‘fraud/signup.html’, {‘form’: form}) | 회원가입 페이지(즉, signup.html 템플릿)를 렌더링하여 사용자에게 폼을 표시한다. 생성된 ‘form’ 객체를 템플릿에 전달한다. |
| 입력 | |
| request | HTTP 요청 객체로, 사용자가 입력한 회원가입 정보(POST 요청)를 포함한다. |
| 출력 | |
| 유효한 데이터가 입력되면 사용자가 회원가입하고 자동으로 로그인된 후, 홈 페이지로 리다이렉트된다. 폼이 유효하지 않을 경우 오류 메시지가 표시된 회원가입 페이지가 렌더링된다. | |
| 매개변수 | |
| request | 사용자가 보낸 HTTP 요청 객체 |
| 예외 처리 | |
| ‘form.is\_valid()’에서 폼 유효성 검사를 통해 잘못된 입력이 있을 경우 오류 메시지를 처리하고 다시 폼을 렌더링한다. | |
| 알고리즘 | |
| 사용자가 회원가입 요청을 제출하면, 이를 검증한 후 유효한 경우 새 사용자로 자동 로그인시키고 홈 페이지로 리다이렉트한다. 유효하지 않은 경우, 다시 회원가입 폼을 보여준다. | |
| 추가 설명 | |
| Django의 기본 회원가입 폼인 UserCreationForm을 사용하여 쉽게 회원가입 기능을 구현할 수 있다. 회원가입 후 자동으로 로그인을 처리하므로, 사용자 경험이 개선된다. | |

### <upload\_form\_sub 함수>

@login\_required

def upload\_form2(request):

if request.method == 'POST':

uploaded\_file = request.FILES['file']

df = pd.read\_csv(uploaded\_file)

db\_name = f'user\_{request.user.username}\_db'

create\_db\_sql = f"CREATE DATABASE IF NOT EXISTS `{db\_name}`"

if db\_name not in settings.DATABASES:

settings.DATABASES[db\_name] = {

'ENGINE': 'django.db.backends.mysql',

'NAME': db\_name,

'USER': 'fds001',

'PASSWORD': 'fds001',

'HOST': '10.0.1.7',

'PORT': '3306',

}

try:

with connection.cursor() as cursor:

cursor.execute(create\_db\_sql)

with connections[db\_name].cursor() as cursor:

cursor.execute('''

CREATE TABLE IF NOT EXISTS fraud\_transaction (

id INT AUTO\_INCREMENT PRIMARY KEY,

amount DECIMAL(10, 2),

customer\_id VARCHAR(255),

merchant\_id VARCHAR(255),

lat DECIMAL(10, 8),

`long` DECIMAL(11, 8),

fraud\_prediction DECIMAL(10, 8)

)

''')

index\_name = f'user\_{request.user.username}\_transactions'

customer\_id\_encoder = LabelEncoder()

merchant\_id\_encoder = LabelEncoder()

df['Customer ID'] = customer\_id\_encoder.fit\_transform(df['Customer ID'])

df['Merchant ID'] = merchant\_id\_encoder.fit\_transform(df['Merchant ID'])

model = load\_model()

for \_, row in df.iterrows():

try:

if not all(col in row for col in ['Amount', 'Customer ID', 'Merchant ID', 'Lat', 'Long']):

return HttpResponse("The format of the CSV file is not correct. Essential column is not exist")

except Exception as e:

return HttpResponse(f"The error occurs in data process stage: {str(e)}")

prediction\_input = np.concatenate([np.array([row['Amount'], row['Customer ID'], row['Merchant ID'], row['Lat'], row['Long']]), random\_features()])

prediction = model.predict(prediction\_input.reshape(1, -1))[0]

cursor.execute('''

INSERT INTO fraud\_transaction (amount, customer\_id, merchant\_id, lat, `long`, fraud\_prediction)

VALUES (%s, %s, %s, %s, %s, %s)

''', (row['Amount'], row['Customer ID'], row['Merchant ID'], row['Lat'], row['Long'], prediction))

try:

es.index(index=index\_name, body={

'amount': row['Amount'],

'customer\_id': row['Customer ID'],

'merchant\_id': row['Merchant ID'],

'lat': row['Lat'],

'long': row['Long'],

'fraud\_prediction': prediction,

'user\_id': request.user.username,

'timestamp': datetime.now(), })

print(f"Successfully indexed transaction to {index\_name}")

except Exception as e:

print(f"Failed to index data in Elasticsearch: {str(e)}")

request.session['uploaded\_data'] = df.to\_json()

return redirect('dashboard2')

except OperationalError as e:

return HttpResponse(f"Database connection for {db\_name} failed: {str(e)}")

return render(request, 'fraud/upload\_form2.html')

| 기능 | |
| --- | --- |
| 사용자가 업로드한 CSV 파일을 처리하여 사용자별 데이터베이스에 저장하고, 머신러닝 모델을 사용해 이상 거래 예측을 수행한 후 예측 결과를 데이터베이스와 Elasticsearch에 저장하는 역할을 한다. 처리 후 사용자는 대시보드 페이지로 리다이렉트된다. | |
| 코드 설명 | |
| if request.method == ‘POST’:  uploaded\_file = request.FILES[‘file’]  df = pd.read\_csv(uploaded\_file) | POST 요청을 처리하여 사용자가 업로드한 CSV 파일을 ‘request.FILES’에서 받아온 후, pandas의 ‘pd.read\_csv()’로 데이터를 데이터프레임으로 변환한다. |
| db\_name = f’user\_{request.user.username}\_db’  create\_db\_sql = f”CREATE DATABASE IF NOT EXISTS `{db\_name}`” | 사용자의 이름을 기반으로 사용자별 데이터베이스 이름을 동적으로 생성한다. 데이터베이스가 존재하지 않을 경우 ‘CREATE DATABASE’ SQL 명령을 사용해 데이터베이스를 생성한다. |
| if db\_name not in settings.DATABASES:  settings.DATABASES[db\_name] = {...} | Django 설정에 데이터베이스가 등록되지 않은 경우, 동적으로 사용자별 데이터베이스 설정을 추가한다. |
| with connection.cursor() as cursor:  cursor.execute(create\_db\_sql) | MySQL 데이터베이스에 연결하여 사용자별 데이터베이스를 생성하거나, 이미 존재하는 경우 데이터베이스를 선택한다. |
| with connections[db\_name].cursor() as cursor:  cursor.execute(‘’’  CREATE TABLE IF NOT EXISTS fraud\_transaction (...)’’’) | 트랜잭션 데이터를 저장할 테이블을 생성한다. 테이블에는 이상 거래 예측값을 저장할 필드가 포함된다. |
| customer\_id\_encoder = LabelEncoder()  merchant\_id\_encoder = LabelEncoder()  df[‘Customer ID’] = customer\_id\_encoder.fit\_transform(df[‘Customer ID’])  df[‘Merchant ID’] = merchant\_id\_encoder.fit\_transform(df[‘Merchant ID’]) | LabelEncoder를 사용해 고객 ID와 상점 ID를 숫자로 인코딩하여 머신러닝 모델에서 사용할 수 있도록 반환한다. |
| model = load\_model | 머신러닝 모델을 불러오는 ‘load\_model() 함수로 IsolationForest 모델을 로드한다. |
| prediction\_input = np.concatenate([np.array([...]), random\_features()])  prediction = model.predict(prediction\_input.reshape(1, -1))[0] | 트랜잭션 데이터를 입력으로 사용해 모델 예측을 수행한다. 예측된 결과는 데이터베이스에 저장된다. |
| cursor.execute(‘’’  INSERT INTO fraud\_transaction (...)  VALUES (%s, %s, %s, %s, %s, %s)  ‘’’, (..., prediction)) | 예측 결과와 트랜잭션 데이터를 데이터베이스에 저장한다. |
| es.index(index = index\_name, body = {…}) | 트랜잭션 데이터를 Elasticsearch에 인덱싱한다. 이를 통해 사용자가 Kibana 대시보드에서 트랜잭션 데이터를 시각화할 수 있다. |
| request.session[‘uploaded\_data’] = df.to\_json() | ‘request.session’은 Django에서 제공하는 세션 객체로, 사용자의 세션에 데이터를 저장할 수 있다. 이 세션 데이터는 사용자가 로그아웃하거나 세션이 만료되기 전까지 유지된다.  ‘df.to\_json()’은 pandas 데이터프레임을 JSON 형식으로 변환하는 메서드이다. CSV 파일에서 읽은 데이터를 JSON 형식으로 변환하여 세션에 저장한다. JSON 형식으로 저장하면 데이터의 구조를 유지한 채 쉽게 읽고 쓸 수 있다. |
| return redirect(‘dashboard2’) | 데이터 처리가 완료되면 ‘dashboard2’ 페이지로 리다이렉트하여 사용자에게 대시보드를 보여준다. |
| return render(request, ‘fraud/upload\_form2.html’) |  |
| 입력 | |
| request | 사용자가 업로드한 CSV 파일과 함께 전송된 HTTP 요청 객체. |
| 출력 | |
| 처리된 데이터를 데이터베이스와 Elasticsearch에 저장한 후, 사용자는 ‘dashboard2’ 페이지로 리다이렉트된다. | |
| 매개변수 | |
| request | HTTP 요청 객체로, 사용자 입력 및 파일 업로드 정보를 포함한다. |
| 예외 처리 | |
| 데이터베이스 연결에 실패할 경우 OperationalError 예외를 처리하여 적절한 오류 메시지를 반환한다. | |
| CSV 파일의 형식이 올바르지 않거나 필수 열이 누락된 경우 예외를 처리하여 사용자에게 경고 메시지를 반환한다. | |
| 알고리즘 | |
| 사용자가 업로드한 데이터를 처리하여 사용자별 데이터베이스에 저장한 후, 머신러닝 모델을 사용해 예측을 수행한다. 예측 결과는 데이터베이스와 Elasticsearch에 저장된다. | |
| 추가 설명 | |
| 이 함수는 사용자별로 데이터를 처리하고 저장하는 시스템의 핵심 기능 중 하나로, 사용자 경험을 맞춤형으로 제공하기 위해 데이터베이스와 Elasticsearch를 동적으로 관리한다. | |

### <dashboard\_sub 함수>

@login\_required

def dashboard2(request):

uploaded\_data = request.session.get('uploaded\_data')

if uploaded\_data:

df = pd.read\_json(uploaded\_data)

model = load\_model()

main\_features = ['Amount', 'Customer ID', 'Merchant ID', 'Lat', 'Long']

model\_features = model.feature\_names\_in\_

df = df.rename(columns={

'Amount': 'amt',

'Customer ID': 'customer\_id',

'Merchant ID': 'merchant\_id',

'Lat': 'lat',

'Long': 'long'

})

for feature in model\_features:

if feature not in df.columns:

df[feature] = np.random.rand(len(df))

X = df[model\_features]

fraud\_prediction = model.predict(X)

df['Fraud Prediction'] = fraud\_prediction

df['Transaction Type'] = np.where(df['Fraud Prediction'] == -1, 'Fraud', 'Normal')

lat = df['lat']

long = df['long']

fraud\_prediction = df['Fraud Prediction']

plt.figure(figsize=(10, 6))

colors = np.where(df['Fraud Prediction'] == 1, 'red', 'lavender')

plt.scatter(long, lat, c=colors, s=100)

plt.colorbar(label='Fraud Prediction')

plt.xlabel('Longitude')

plt.ylabel('Latitude')

plt.title('Fraud Prediction Map')

buffer = io.BytesIO()

plt.savefig(buffer, format='png')

buffer.seek(0)

image\_png = buffer.getvalue()

buffer.close()

graphic = base64.b64encode(image\_png).decode('utf-8')

fraud\_transactions = df[df['Fraud Prediction'] == 1][['amt', 'customer\_id', 'merchant\_id', 'lat', 'long']]

if not fraud\_transactions.empty:

try:

email\_html\_message = render\_to\_string('fraud/email\_template.html', {'fraud\_transactions': fraud\_transactions})

email\_plaintext\_message = strip\_tags(email\_html\_message)

send\_mail(

subject = '[Sniffers] Fraudulent Transaction Detected!',

message = email\_plaintext\_message,

from\_email = settings.DEFAULT\_FROM\_EMAIL,

recipient\_list = [request.user.email],

html\_message = email\_html\_message,

fail\_silently = False,

)

except Exception as e:

print(f"Failed to send email: {str(e)}")

if 'download' in request.GET:

response = HttpResponse(image\_png, content\_type='image/png')

response['Content-Disposition'] = 'attachment; filename="fraud\_prediction\_map.png"'

return response

elif 'download\_csv' in request.GET:

response = HttpResponse(content\_type='text/csv')

response['Content-Disposition'] = 'attachment; filename="fraud\_predictions.csv"'

fraud\_transactions.to\_csv(path\_or\_buf=response, index=False)

return response

return render(request, 'fraud/dashboard2.html', {

'graphic': graphic,

'fraud\_transactions': fraud\_transactions.to\_html(classes='table table-striped', index=False)

})

else:

return render(request, 'fraud/dashboard2.html', {'error': 'There is no data'})

| 기능 | |
| --- | --- |
| 사용자가 업로드한 데이터를 기반으로 이상 거래 예측을 수행하고, 그 결과를 시각화 및 HTML 테이블로 제공하는 역할을 한다. 또한, 이상 거래가 감지되면 이메일 알림을 발송하고, 시각화된 이미지를 다운로드할 수 있는 기능도 포함하고 있다. | |
| 코드 설명 | |
| uploaded\_data = request.session.get(‘uploaded\_data’) | 사용자가 업로드한 데이터를 세션에서 JSON 형식으로 가져온다. 데이터가 세션에 없으면 오류 메시지를 반환한다. |
| if uploaded\_data:  df = pd.read\_json(uploaded\_data) | 세션에서 가져온 JSON 데이터를 pandas 데이터프레임으로 변환한다. |
| model = load\_model() | IsolationForest 사기 예측 모델을 로드한다. |
| main\_features = [...]  model\_features = model.feature\_names\_in\_  df = df.rename(columns = { … }) | 모델에서 학습한 주요 특성들과 일치하도록 사용자의 데이터 컬럼 이름을 변경하고, 학습에 사용되지 않은 나머지 특성은 무작위 값으로 채워준다. |
| X = df[model\_features]  fraud\_prediction = model.predict(X)  df[‘Fraud Prediction’] = fraud\_prediction | 모델을 사용하여 사용자의 트랜잭션 데이터를 기반으로 사기 예측을 수행하고, 그 결과를 데이터프레임에 추가한다. |
| df[‘Transaction Type’] = np.where(df[‘Fraud Prediction’] == -1, ‘Fraud’, ‘Normal’) | 예측 결과에 따라 이상 거래(-1) 또는 정상 거래(1)를 구분한다. |
| plt.figure(figsize = (10, 6)) | 새로운 그래프를 생성하는 코드이다. ‘figsize = (10, 6)’는 그래프의 크기를 설정하며, 너비 10인치, 높이 6인치로 지정된다. |
| colors = np.where(df[‘Fraud Prediction’] == 1, ‘red’, ‘lavender’) | ‘df[‘Fraud Prediction’] == 1’은 예측 결과가 1인 경우, 즉 정상 거래일 때 해당된다.  이 조건이 참이면 ‘red’를 반환하고, 그렇지 않으면 ‘lavender’를 반환한다. 여기서 ‘red’는 이상 거래를 의미하고, ‘lavender’는 정상 거래를 의미한다.  결과적으로, ‘color’ 변수는 예측 결과에 따라 각 트랜잭션에 대해 적절한 색상을 지정한다. |
| plt.scatter(long, lat, c = color, s = 100) | ‘plt.scatter()’ 함수는 산점도를 그리는 함수이다.  ‘long(경도)’는 x축 값으로, ‘lat(위도)’는 y축 값으로 사용한다.  점의 색상을 예측 결과에 따라 설정된 ‘color’로 지정한다. 정상 거래는 ‘lavender’, 이상 거래는 ‘red’로 표시된다. |
| buffer = io.BytesIO() | 메모리상에서 파일처럼 데이터를 저장하고 다룰 수 있는 바이트 스트림 버퍼를 생성한다. 즉, 파일을 디스크에 저장하지 않고 메모리에서 직접 처리할 수 있게 해준다. |
| plt.savefig(buffer, format = ‘png’) | 그려진 그래프를 파일로 저장하는 함수이다. |
| buffer.seek(0) | 버퍼의 파일 포인터를 처음 위치로 이동시키는 함수이다.  Matplotlib 이미지가 버퍼에 저장되었지만, 이를 읽기 위해서는 파일 포인터를 처음 위치로 다시 이동해야 한다. |
| image\_png = buffer.getvalue() | 버퍼에 저장된 모든 데이터를 바이트 형식으로 가져온다. 여기서 PNG 형식의 이미지 데이터를 가져와 ‘image\_png’ 변수에 저장한다. |
| buffer.close() | 메모리에서 더 이상 필요 없는 버퍼를 닫아 리소스를 해제한다.  이 작업은 메모리 누수를 방지하기 위해 중요하다. |
| graphic = base64.b64encode(image\_png).decode(‘utf-8’) | ‘base64.b64encode()’는 바이너리 데이터를 Base64 형식으로 인코딩하는 함수이다.  ‘image\_png’는 PNG 이미지 데이터를 Base64로 인코딩하여, ‘graphic’ 변수에 저장한다.  ‘.decode(‘utf-8’)’은 Base64로 인코딩된 데이터를UTF-8 문자열로 변환한다.HTML에서 이미지를 표시할 때, Base64 형식의 문자열로 변환된 이미지를 사용할 수 있다. |
| fraud\_transactions = df[df[‘Fraud Prediction’] == 1][[ … ]] | ‘Fraud Prediction’ 열에서 값이 1인 행, 즉 이상 거래로 예측된 행만 필터링한다.  ‘df[ … ]’은 해당 조건에 맞는 데이터만 추출하여 새로운 데이터프레임을 생성한다. |
| if not fraud\_transactions.empty: | pandas 데이터프레임의 ‘empty’ 속성을 사용하여 데이터프레임이 비어 있는지 확인한다.  ‘not fraud\_transactions.empty’의 조건에 의해 데이터프레임에 이상 거래가 하나라도 있으면 True를 반환한다. |
| email\_html\_message = render\_to\_string(‘fraud/email\_template.html’, {‘fraud\_transactions’: fraud\_transactions})  email\_plaintext\_message = strip\_tags(email\_html\_message) | ‘render\_to\_string()’은 Django의 템플릿 엔진을 사용하여 HTML 템플릿을 렌더링한다.  ‘fraud/email\_template.html’은 이메일 본문을 위한 템플릿 파일 경로이다.  ‘{‘fraud\_transactions’: fraud\_transactions}’는 이상 거래 데이터프레임을 템플릿에 전달하여, 템플릿 안에서 이상 거래 목록을 출력할 수 있도록 한다.  ‘strip\_tags()’는 HTML 태그를 제거하고 텍스트 형식의 이메일 내용을 생성한다. 이는 일부 클라이언트가 HTML 형식 대신 일반 텍스트를 사용할 수 있기 때문에 필요하다. |
| send\_mail( … ) | Django에서 제공하는 이메일 발송 함수이다. |
| except Exception as e:  print(f”Failed to send email: {str(e)}”) | 이메일 발송 중 예외가 발생하면, 이를 처리하고 콘솔에 오류 메시지를 출력한다.  (예외가 발생할 수 있는 상황에는 이메일 서버 문제, 사용자 이메일 주소 오류 등이 있다.) |
| if ‘download’ in request.GET: | 사용자의 GET 요청에 ‘download’라는 매개변수가 포함되어 있는지 확인한다. |
| response = HttpResponse(image\_png, content\_type = ‘image/png’) | ‘image\_png’는 앞서 생성한 PNG 형식의 이미지 데이터이다.  ‘content\_type = ‘image/png’’는 응답의 콘텐츠 타입을 PNG 이미지로 설정한다. |
| response[‘Content-Disposition’] = ‘attachment; filename = “fraud\_prediction\_map.png”’ | 브라우저가 파일을 다운로드하도록 강제하고, 다운로드되는 파일의 이름을 ‘fraud\_predictions.png’로 지정한다. |
| return response | 최종적으로 이 응답을 반환하여 사용자가 이미지를 다운로드할 수 있도록 한다. |
| elif ‘download\_csv’ in request.GET: | GET 요청에 ‘download\_csv’ 매개변수가 포함되어 있는지 확인한다. |
| response = HttpResponse(content\_type = ‘text/csv’) | 응답의 콘텐츠 타입을 CSV 파일로 설정한다. |
| response[‘Content-Disposition’] = ‘attachment; filename = “fraud\_predictions.csv”’ | 파일을 다운로드하도록 설정하고, 다운로드되는 파일의 이름을 ‘fraud\_predictions.csv’로 지정한다. |
| fraud\_transactions.to\_csv(path\_or\_buf = response, index - False) | ‘fraud\_transactions’ 데이터프레임을 CSV 형식으로 변환하여 ‘response’ 객체에 기록한다.  ‘index = False’는 CSV 파일에 pandas 인덱스가 포함되지 않도록 설정한다. |
| return response | 최종적으로 응답을 반환하여 CSV 파일을 다운로드할 수 있도록 한다. |
| 입력 | |
| request | HTTP 요청 객체. 세션에 저장된 데이터를 가져오고, 사용자가 요청한 파일 다운로드를 처리한다. |
| 출력 | |
| 이상 거래 예측 결과를 시각화하여 HTML 페이지에 그래픽으로 표시한다. 이상 거래를 탐지한 경우, 이메일을 발송하고 다운로드 요청이 있을 경우 이미지나 CSV 파일을 반환한다. | |
| 매개변수 | |
| request | HTTP 요청 객체로, 사용자 세션에서 데이터를 가져오고 렌더링한다. |
| 예외 처리 | |
| 세션에 데이터가 없는 경우, 오류 메시지가 출력된다. | |
| 이메일 발송 및 Elasticsearch 인덱싱 실패 시 예외 처리 로직이 포함되어 있어 오류 메시지를 출력한다. | |
| 알고리즘 | |
| 세션에서 데이터를 가져와 모델 예측을 수행하고, 그 결과를 시각화한다. 사용자가 요청한 경우 다운로드 기능을 제공하고, 이상 거래를 감지하면 이메일로 알림을 발송한다. | |
| 추가 설명 | |
| 이 함수는 사용자가 업로드한 데이터를 기반으로 실시간 이상 거래 예측을 수행하며, 시각화 및 알림 기능을 제공한다. 또한, 데이터를 직관적으로 확인할 수 있도록 다양한 출력을 제공한다. | |