

플랫폼 배달운전자의 안전을 위한 사고 예측 및 산재 지원 서비스

- 서울특별시를 중심으로 -

박서윤, 심나은

본 연구는 배달 플랫폼 산업의 성장과 함께 이륜차 배달 라이더들의 안전 문제와 산재 처리의 필요성이 증가하는 상황에서, 이륜차 사고 시 운전자의 상해 정도를 예측하고 산재 처리 과정을 지원하는 시스템을 개발하는 것을 목표로 한다. 이를 위해 2021년부터 2023년까지 TAAS 교통사고 분석 시스템에서 서울특별시 이륜차 교통사고 데이터 14,944건을 수집하였다. 수집된 데이터를 바탕으로 연령, 사고일시 등의 변수를 전처리하고, 머신러닝 모델을 활용하여 피해 운전자의 상해 정도를 예측하였다. 다양한 머신러닝 알고리즘 중 XGBoost 모델이 가장 높은 예측 성능을 보여 최종 모델로 선정되었으며, 이 모델의 Feature Importance를 분석하여 독립변수들의 기여도를 평가하였다. 또한, ChatGPT API를 연동하여 피해운전자의 상해 정도에 따른 산재보험 처리 절차를 안내하는 시스템을 구현하였다.

결론적으로 본 연구는 배달 라이더의 안전을 실시간으로 경고하고, 산재 처리의 효율성을 높일 수 있는 시스템을 구축하여 배달 산업에서의 안전 문화를 선도할 수 있는 중요한 기반을 마련하였다. 향후 연구에서는 모델 성능 고도화 및 시스템의 효용성 강화를 통해 더욱 종합적인 지원 서비스를 제공할 수 있는 방안을 모색할 수 있을 것으로 기대된다.

핵심어: 배달 라이더 안전, 산재 처리 지원, 실시간 경고 시스템

목차

1. 서론

2. 플랫폼 배달운전자의 안전보장 대책에 관한 논의

2.1 플랫폼 배달운전자의 정의

2.2 배달원 수 및 배달 플랫폼 기업 산재 신청 현황

2.3 배달라이더 산재보험 가입 및 신청 현황

2.4 국내 및 해외 배달 플랫폼 기업의 배달운전자 안전보장 대책 비교

3. 연구 방법

3.1 데이터 수집 및 전처리

3.2 탐색적 데이터 분석

3.3 머신모델 모델링

3.4 모델링 결과와 특징

3.5 모델 활용

4. 결론 및 제언

1. 서론

코로나 19로 인해 배달 시장이 성장하며, 배달 플랫폼은 확장되고 배달 운전자 수가 증가하기 시작했다. 그에 따라 전통적으로 산재 신청 건수가 높았던 건설업과 제조업을 제치고, 2022년부터는 국내 대표적인 배달 플랫폼인 ‘배달의 민족’의 물류 서비스를 전담하는 ‘우아한 청년들’이 산재 신청 건수 1위를 차지했다. 또한, 쿠팡이츠 등 다른 배달 플랫폼들도 산재 신청 순위에서 상위를 기록하며, 배달 플랫폼 소속 운전자들의 안전을 보장하기 위한 사회적 논의가 활발해지고 있다. 예를 들어, 올해 4월 30일 정부와 8개 배달 플랫폼은 「배달종사자 교통안전 문화 조성을 위한 협약」을 체결하는 등 배달 안전 확보를 위한 움직임이 나타나고 있다. 그러나 이 협약의 내용은 교통안전 교육, 저렴한 보험 상품 개발 등으로 사고 예방을 위한 실질적인 문제 해결에는 한계가 있으며, 종사자 안전대책 마련에 대해 플랫폼 기업의 자율성에 맡기고 있다.

시장 내 플랫폼 기업들은 치열한 경쟁 관계에 놓여 있어 안전 확보를 위한 개별 기업의 투자나 업계 차원의 움직임은 미진한 편이다. 대부분의 기업들은 안전 교육, 혹서기 대비 안전 물품 지원, 이륜차 무상 안전점검 등을 실시하고 있지만, 이러한 대책들은 사고의 근본적인 원인을 해결하는 데에는 한계가 있다. 올해 배달의 민족은 업계에서 처음으로 노조와 공동으로 위험성 평가를 실시해 사고 유발 요인을 찾아내려는 노력을 기울였다. 이러한 움직임은 최근에야 나타나기 시작했으며, Uber와 Grab과 같은 글로벌 플랫폼 기업에 비하면 운전자들의 안전을 보장하기 위한 대책이 여전히 부족한 상황이다.

사고 예방뿐만 아니라 사고 발생 시 치료와 이륜차 수리, 그리고 배달 수행 불가로 인한 생계 어려움을 대비하기 위해서는 보험이 필요하다. 2022년 5월 전속성 폐지와 관련된 산재보험법 개정으로 2024년 배달 라이더의 산재보험 가입자 수는 30만명으로, 법 개정 이전보다 약 3배 증가했다. 산재보험 혜택을 받을 수 있는 배달 운전자가 늘어났음에도 불구하고, 산재 신청에 어려움을 겪어 실제로 보상받는 이는 많지 않은 것으로 보인다. 서울대학교 사회발전 연구소가 배달라이더 605명을 대상으로 실태조사를 실시한 결과, 사고를 경험한 라이더 10명 중 1명만이 실제 산재보험 처리를 시도해 보상을 받을 것으로 나타났다.

따라서, 본 연구에서는 이륜차 사고의 위험 요인을 바탕으로 사고 발생 시 이륜차 운전자의 상해 정도를 예측하는 모델을 개발하고, 이를 생성형 AI와 연동하여 위험을 경고하고 사고 발생 후 산재 처리 과정을 문자 메시지로 안내함으로써 배달 라이더의 산재보험 처리에 대한 진입장벽을 낮추고, 플랫폼 기업의 사고 해결 시스템 구축에 도움을 주고자 한다.

2. 플랫폼 배달운전자의 안전보장 대책에 관한 논의

2.1 플랫폼 배달운전자의 정의

아직 법적으로 구체적인 플랫폼 노동자 기준이 있진 않다. 하지만 2020년 7월 일자리위원회의 '플랫폼노동과 일자리 TF'에서 제시한 플랫폼 노동자의 조건 네 가지는 다음과 같다.

플랫폼 노동자의 기준

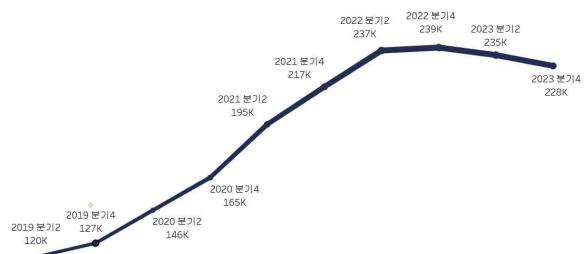
1. 디지털 플랫폼을 통해 거래되는 서비스(용역) 또는 가상재화 생산 노동
2. 디지털 플랫폼을 통해 일거리(short jobs, projects, tasks)를 구할 것
3. 디지털 플랫폼이 보수(payment)를 중개할 것
4. 일거리가 특정인이 아니라 다수에게 열려있을 것

연구 대상은 배달 플랫폼에 소속되어 이륜자동차를 이용해 음식 및 공산품 등을 배달해주는 서비스를 제공하는 배달대행 운전자이다.

2.2 배달원 수 및 배달 플랫폼 기업 산재 신청 현황

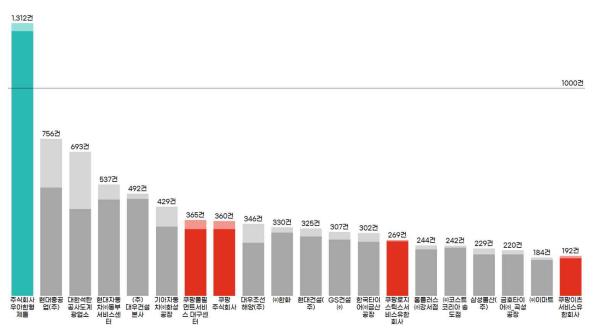
통계청 마이크로데이터에서 2019년부터 2023년까지 지역별고용조사 데이터를 활용해 배달원 수의 변화 추이를 분석한 결과, 2019년 12.7만에서 2023년 22.8만 명으로 약 80% 증가했다.

*표준산업분류상 '소화물전문운송업'이면서 표준직업분류상 '배달원'에 속하는 인원에 '시도가중값'을 적용하여 산출.



〈그림 1〉 2019년~2023년 반기별 배달원 수

2023년 1월부터 8월까지 산재 신청 및 승인 건수 상위 20위 사업장을 살펴본 결과, 5개 사업장이 플랫폼 기업이거나 물류 자회사였다. 그 중 배달 플랫폼은 '(주)우아한 청년들'과 경쟁 기업인 '쿠팡이츠서비스유한회사'로 각각 1위, 20위를 차지했다. '(주)우아한 청년들'은 유일하게 1000건을 넘으며 독보적인 1위를 차지했으며, 7위는 '쿠팡풀필먼트서비스 대구센터', 8위는 '(주)쿠팡', 14위는 '쿠팡로지스틱스 유한회사', 20위는 '쿠팡이츠서비스 유한회사'로 쿠팡과 물류 자회사가 다수를 차지했다.



〈그림 2〉 2023년 1~8월 산재신청 및 승인 건수 상위 20위 사업장

2.3 배달라이더 산재보험 가입 및 산재 신청 현황

2022년 5월 전속성 폐지 산재보험법 개정으로 2023년 7월부터 여러 사업장에 종사하여도 산재보험 혜택을 받을 수 있게 되며, 많은 배달라이더들이 산재보험 울타리 안으로 들어오게 되었다. 고용노동부에 따르면, 배달라이더가 포함되어 있는 퀵서비스 기사의 산재보험 가입자 수는 11만명('23.6월)에서 30만명('24.5월)으로 약 3배 증가하였다.

그러나 산재보험 가입자 수는 늘어났지만, 실제로 산재를 신청해 보상을 받는 라이더는 여전히 적은 것으로 나타났다. 서울대학교 사회발전연구소의 양종민 부연구위원이 2022년 8월 23일부터 9월 23일까지 배달라이더 605명을 대상으로 실시한 실태조사 결과에 따르면, 사고 경험 라이더 10명 중 1명만 실제 산재보험 처리를 시도해 보상을 받을 것으로 나타났다. 특히, 영세 음식점 소속이거나 여러 플랫폼을 동시에 이용하는 배달라이더일수록 산재보험 이용률이 낮았다. 개정된 법이 법 공포('22년 6월) 이후 시행일('23년 7월) 사이에 발생하는 재해에도 유효하다는 점을 고려할 때, 해당 연구는 대부분의 배달 라이더가 산재 신청이 가능해졌음에도 불구하고 여전히 많은 이들이 산재 신청으로 보상을 받지 못하고 있음을 증명했다.

또한, 통계청의 퀵서비스 업종 산재신청 건수 데이터와 한국도로교통공단의 교통사고분석시스템의 이륜차 사고건수를 비교해 사고 발생 시 얼마나 많은 배달 운전자들이 산재보험을 신청하는지 추정치를 산출해보았다. 2022년 퀵서비스 업종 산재신청 건수 6405건이고, 이륜차 사고 건수는 18,295건으로, 사고 발생 시 산재 신청률은 약 35%에 불과했다.

2.4 국내 및 해외 배달 플랫폼 기업의 배달운전자 안전보장 대책 비교

2.4.1 국내 기업

배달의 민족

- 우아한 라이더 살핌기금: 우아한형제들은 사랑의 열매와 업무협약을 체결하여, 음식 배달 중 사고를 당해 어려움을 겪는 라이더를 지원하는 ‘우아한 라이더 살핌기금’ 사회공헌 사업을 진행하고 있다.
- 혹서기 대비: 계절성 물품 지원, 폭염 쉼터 제공, 폭염 대비 가이드 배포, 안전 교육 등을 실시하여 라이더의 안전을 도모하고 있다.
- 시간제 보험 도입: 2019년 12월 1일, 현대해상과 함께 업계 최초로 시간제 보험을 도입하

여 보험료 부담을 낮췄다.

- AI 추천배차 시스템: 2020년, AI 추천배차 시스템을 개발하여 사고 확률을 27.8% 감소시켰다. 이 시스템은 최적의 동선을 통해 배차를 추천하고, 운행 중 다른 라이더와 경쟁을 원천 차단하여 사고 예방에 기여했다.

그러나 2023년 4월 도입된 뒷음 배달 서비스 ‘알뜰배달’에 AI 배차 시스템을 적용되면, 비효율적인 배차 동선과 운행 중 추가 콜을 받아야 하는 방식으로 인해 오히려 안전에 위협을 주고 있다. 또한 사고 예방보다는 배달 과정 최적화를 목적으로 개발되어 안전을 보장하기에는 한계가 있다.

- 위험성 평가 및 상생지원 제도: 2023년, 국내 최초로 배달 플랫폼 노동조합과 협력하여 플랫폼 종사자 라이더의 안전을 위해 공동 위험성 평가를 진행하여 사고 유발 가능성이 높은 요인을 파악했다.

또한, 노조와 협의하여 ‘플랫폼 라이더 상생지원 제도’를 마련하여, 일정 조건을 충족하면(배민 커넥트를 통해 연간 220일 이상, 하루 22~30건 이상 배달을 수행한 라이더) 사고로 인한 입원으로 배달수행이 불가한 경우에 100%, 입원치료가 불필요한 경우에도 일부 인정하여 지원금을 받을 수 있다.

- 라이더 안전 교육: 2018년, 경기도 고양시에서 외부 기관과 협업하여 1세대 배민라이더스쿨을 출범시켰으며, 2025년까지 자체 교육 시스템으로 4세대 이륜차 안전 교육 시설인 ‘하남 배민라이더스쿨’을 건립할 예정이다.

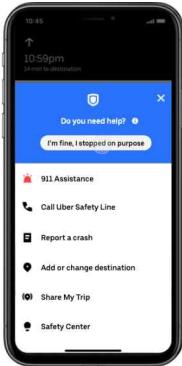
쿠팡이츠

- 안전보건 관리 체계 구축: 태풍, 폭우, 폭설 등 악천후 발생 시 셧다운 기준을 강화하고, 전국 배달파트너 8,000여 명에게 KC 인증 헬멧을 제공하고 있다. 또한 여름철 배달파트너 쉼터에 생수와 이온음료, 포도당 캔디 등을 지원하고 있다.
- 이륜차 안전강화 MOU: 한국오토바이정비협회와 안전업무강화 MOU를 체결하여 이륜차 무상 안전점검 및 안전용품을 지원하고, 이륜차 실습 안전교육을 실시하고 있다.
- 서울시와 한국교통안전공단이 실시한 교통안전 체험 교육 프로그램 후원
- 시간제 보험

2.4.2 해외 기업

Uber

- RideCheck 시스템: RideCheck는 센서와 GPS 추적을 통해 실시간으로 운전자의 상태를 모니터링한다. 사고 발생 시 모바일 앱을 통해 사고를 보고하면 즉시 보험사에 이를 통보하여 운전자가 별도로 개인 보험사에 보고할 필요 없이 신속하게 대응할 수 있는 시스템을 운영하고 있다.
- 24/7 안전 센터: 24시간 연중무휴로 도움을 받을 수 있는 안전 센터를 운영하고 있다. 긴급 상황에서는 앱을 통해 911에 연락하면 실시간 위치, 차량 정보, 차량 번호판을 긴급 구조 대에 자동으로 공유하여 신속한 도움을 받을 수 있다.
- 보험: 사고로 부상을 당해 일할 수 없는 경우 장애 수당, 공제가 없는 의료비, 가족 구성원에 대한 유족 급여를 보장받을 수 있다. 사고 발생 시 운전자를 대신해 최소 100만 달러를 지원하는 상업용 자동차 보험을 유지하고 있다.



〈그림 3〉 RideCheck 작동 화면

Grab

- 주행 모니터링: 주행 중 위험한 상황을 감지하기 위해 지속적으로 주행을 모니터링하고, 사용자에게 지원이 필요한 경우 알림을 보낸다. 또한 운전자가 권장 운전 시간을 초과한 경우 휴식 알림을 전송한다.
- 24/7 안전 센터: 24시간 연중무휴 긴급상황 발생 시 도움을 받을 수 있다.
- 운전 안전 보고: 앱에서 위험한 사건이 발생한 정확한 장소에 대한 세부 정보를 제공한다.
- 보험: Grab Delivery Partner로 가입하면 자동으로 보험에 가입되어 근무 중 사고로 인한 사망, 영구장애, 화상 및 치료비에 대해 높은 보험금을 보장해준다.

국내 플랫폼 기업은 안전교육, 안전 물품 지급, 저렴한 보험상품 개발 등 표면적인 접근에 집중해왔으며, 최근에서야 위험성 평가를 통해 사고의 근본적인 원인을 찾기 시작했다. 해외 플랫폼 기업들은 교육이나 보험상품뿐만 아니라 주행 중 운전자의 상태를 모니터링하고 위험 상황을 감지하기 위한 연구를 통해 사고 예방과 사고 후처리에 대한 선구적인 시스템을 구축했다.

3. 연구방법

3.1 데이터 수집 및 전처리

TAAS 교통사고 분석시스템에서 2021년부터 2023년까지 3년간 서울특별시 교통사고 데이터 중 피해 차종이 ‘이륜차’인 데이터 14,944건을 확보하였다. 수집한 데이터의 목록은 <표 1>과 같다.

데이터 목록	
사고일시	
요일	
시군구	
사고내용	
사망자수/중상자수/경상자수/부상신고자수	
사고유형	
법규위반	
노면상태	
기상상태	
도로형태	
가해운전자 차종/성별/연령/상해정도	
피해운전자 차종/성별/연령/상해정도	

<표 1> 수집 데이터 목록

3.1.1 연령 전처리

```

import re

# 숫자 부분만 추출하는 함수 정의
def remove_non_numeric(age):
    if isinstance(age, str):
        # 정규 표현식을 사용하여 숫자만 추출
        numeric_part = re.findall(r'\d+', age)
        if numeric_part:
            return int(numeric_part[0])
    return age

# '피해운전자 연령' 컬럼에 함수 적용
df['피해운전자 연령'] = df['피해운전자 연령'].apply(remove_non_numeric)
df['가해운전자 연령'] = df['가해운전자 연령'].apply(remove_non_numeric)

# 결과 확인
print(df['피해운전자 연령'].unique())
print(df['가해운전자 연령'].unique())

```



```

# 라벨링 함수
def label_age(age):
    if 1 <= age < 10:
        return '10대 미만'
    elif 10 <= age < 20:
        return '10대'
    elif 20 <= age < 30:
        return '20대'
    elif 30 <= age < 40:
        return '30대'
    elif 40 <= age < 50:
        return '40대'
    elif 50 <= age < 60:
        return '50대'
    elif 60 <= age < 70:
        return '60대'
    elif 70 <= age < 80:
        return '70대'
    else:
        return '80세 이상'

# '가해운전자 연령' 컬럼에 라벨링 적용
df['가해운전자 연령대'] = df['가해운전자 연령'].apply(label_age)
age_groups = ['10대 미만', '10대', '20대', '30대', '40대', '50대', '60대', '70대', '80세 이상']
offend_age_group_counts = df['가해운전자 연령대'].value_counts().reindex(age_groups, fill_value=0)

# '피해운전자 연령' 컬럼에 라벨링 적용
df['피해운전자 연령대'] = df['피해운전자 연령'].apply(label_age)
victim_age_group_counts = df['피해운전자 연령대'].value_counts().reindex(age_groups, fill_value=0)

print(offend_age_group_counts)
print(victim_age_group_counts)

```

<그림 4> ‘연령’칼럼 전처리 코드

‘연령’ 칼럼은 7세부터 91세까지 무분별하게 분포하기 때문에, 숫자만 추출하여 10세 단위인 ‘연령대’로 라벨링 하여 범주화하였다.

3.1.2 사고일시 전처리

```
# 시간 정보를 추출하는 함수
def extract_hour(time_str):
    match = re.search(r'\d{1,2}시$', time_str)
    if match:
        return match.group()
    else:
        return None

df['시간'] = df['사고일시'].apply(extract_hour)
sorted(df['시간'].unique())

# 시간대를 라벨링하는 함수
def label_time_period(time_str):
    if any(hour in time_str for hour in ['01시', '02시', '03시']):
        return '01:00 - 03:00'
    elif any(hour in time_str for hour in ['04시', '05시', '06시']):
        return '04:00 - 06:00'
    elif any(hour in time_str for hour in ['07시', '08시', '09시']):
        return '07:00 - 09:00'
    elif any(hour in time_str for hour in ['10시', '11시', '12시']):
        return '10:00 - 12:00'
    elif any(hour in time_str for hour in ['13시', '14시', '15시']):
        return '13:00 - 15:00'
    elif any(hour in time_str for hour in ['16시', '17시', '18시']):
        return '16:00 - 18:00'
    elif any(hour in time_str for hour in ['19시', '20시', '21시']):
        return '19:00 - 21:00'
    elif any(hour in time_str for hour in ['22시', '23시', '00시']):
        return '22:00 - 24:00'
    else:
        return '0'

# 데이터프레임에 라벨링 적용
df['사고시간대'] = df['시간'].apply(label_time_period)
df['사고시간대'].value_counts()
```

〈그림 5〉 ‘사고일시’ 칼럼 전처리 코드

‘0000년 00월 00일 00시’ 형식으로 구성되어 있는 ‘사고 일시’ 칼럼의 경우, ‘00시’인 시간만을 추출하여 3시간 단위인 ‘사고시간대’로 범주화하여 라벨링 하였다.

3.1.3 데이터 인코딩

```
In [ ]:
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

one_hot_encoder = OneHotEncoder(sparse=False)
label_encoder = LabelEncoder()

# 피해운전자 상해정도 레이블 인코딩을 위한 매핑 딕셔너리
victim_injury_mapping = {
    '상해없음': 0,
    '부상신고': 1,
    '기타불명': 2,
    '경상': 3,
    '중상': 4,
    '사망': 5
}

# 인코딩할 칼럼들
input_variables=['사고시간대','요일','시군구','사고내용','법규위반','노면상태','기상상태','도로형태','가해운전자 차종','가해운전자 연령대','사고유형']
output_variable='피해운전자 상해정도'

# 원-핫 인코더 초기화
one_hot_encoder = OneHotEncoder(sparse=False)

# 레이블 인코딩 적용
df['피해운전자 상해정도_LabelEncoded'] = df['피해운전자 상해정도'].map(victim_injury_mapping)

# 원-핫 인코딩 적용(피해운전자 상해정도 제외)
XM = pd.get_dummies(df[input_variables])

# 피해운전자 상해정도
y = df['피해운전자 상해정도_LabelEncoded']

# 인코딩된 결과 확인
print(XM.head())
print(y)

# 데이터셋의 형태 확인
XM.shape, y.shape
```

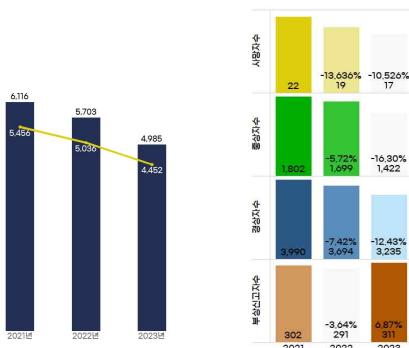
〈그림 6〉 데이터 인코딩 코드

위와 같이 데이터 피처가 지나치게 다양하게 분포되어 있는 칼럼들을 전처리한 이후에는, 독립변수들은 모두 원핫 인코딩으로 변환하였고, 상해 정도의 심각성이 구분이 되는 종속 변수인 '피해운전자 상해정도'는 매핑하여 레이블 인코딩하였다.

변환된 데이터의 종류에 따라 모델에 편파적인 영향이 미칠 수 있기 때문에, 독립변수들은 모두 동일하게 원핫 인코딩으로 전처리하였다.

3.2 탐색적 데이터 분석(EDA: Exploratory Data Analysis)

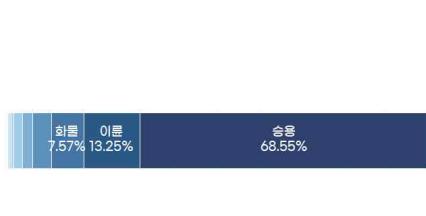
이륜차 사고 현황과 사고가 발생하는 상황적 특징을 파악하고자 탐색적 데이터 분석을 실행하였다. 연도별 사고 건수와 사상자수(사망자수+중상자수+경상자수+부상신고자수)를 살펴보면, 모두 줄어드는 추세이다. 사상자수를 상해 정도별로 나눠서 보면, 부상신고자수를 제외하고 모두 줄어들고 있다. 운전자 특징을 분석한 결과, 피해 운전자는 2040 남성(72%), 가해 운전자는 5060 남성(35%)이 가장 큰 비중을 차지하고 있었다. 사고시간대 중에서는 18시와 21시 사이에 24.63%로 사고가 가장 많이 발생했다. 요일은 금요일이 16.5%로 가장 높았으며, 최소 값은 일요일 12%로, 요소 간 비교적 균등한 비중을 차지했다. 법규위반은 안전운전불이행이 약 48%으로 가장 높았으며, 휴대전화 조작, 흡연, 전방 주시 태만, 운전미숙 등이 포함된다. 기상상태는 도로형태는 교차로에서 52.26%로 사고가 가장 자주 발생하지만, 세부적으로 보면 단일로에서(45%) 사고발생이 가장 많다. 가해차종은 승용차(68%) 가장 많았다. 사고유형은 차대차 사고가 91%로 대부분이었으며, 그 중에서도 충면충돌(34.62%)로 발생하는 사고가 가장 많았다. 시군구를 분석해보면, 구 단위에서는 강남구, 송파구, 동대문구가 각각 10.39%, 6.71%, 6.31%를 차지하며 가장 높았고, 법정동 단위에서는 강남구 역삼동, 관악구 신림동, 관악구 봉천동이 2.968%, 2.903%, 2.423%로 높은 순위를 차지했다. 강남구는 2순위인 관악구와 3.68%로 큰 차이를 보이며 사고 발생이 특히나 잦은 지역으로 나타났다. 상해정도를 살펴보면, 피해운전자는 경상 62.82%, 중상 30.48%였고, 가해운전자는 상해없음이 88.24%를 차지했다. 가해운전자의 상해정도를 차종별로 살펴보면, 승용이 79%로 상당 부분을 차지하고 있으며, 상해없음이 92.74%로 매우 높다. 이를 통해 가해운전자에 비해 피해운전자의 상해정도가 심각하다는 사실을 알 수 있다.



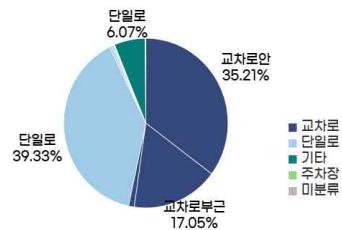
〈그림 7〉 (좌)사고건수 및 사상자수 (우)연도별 상해정도별 사상자수



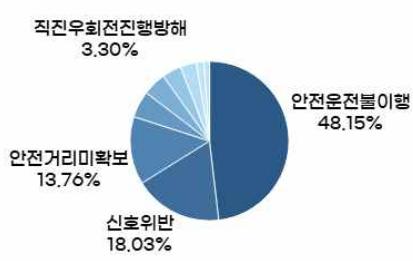
〈그림 8〉 피해운전자 및 가해운전자의 성별 및 연령대



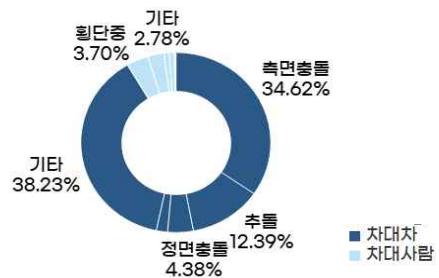
〈그림 9〉 가해차종별 사고 비율



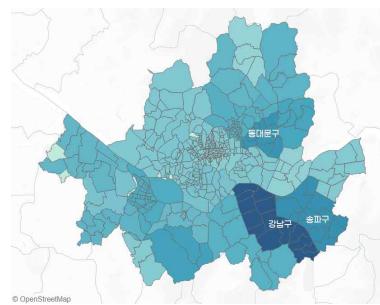
〈그림 10〉 도로형태별 사고 비율



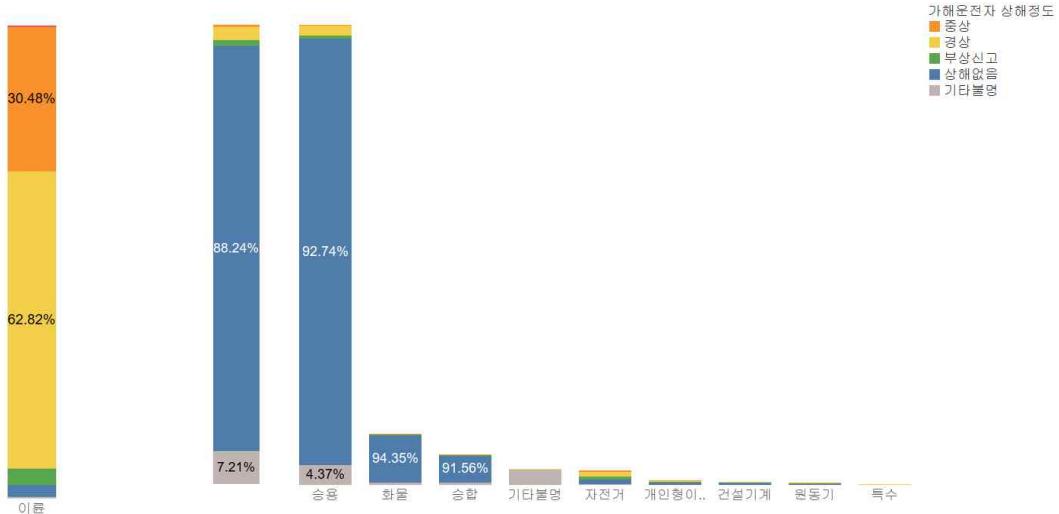
〈그림 11〉 (좌)법규위반별 사고발생 비율



(우)사고유형별 사고발생 비율



〈그림 12〉 시군구별 사고 발생 건수 (좌)구별 (우) 법정동별



〈그림 13〉 피해운전자 상해정도

〈그림 14〉 가해운전자 상해정도 (좌)전체 (우)차종별

3.3 모델링

3.3.1 데이터 스케일링

```
#스케일링
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(XM)
X = scaler.transform(XM)

from sklearn.model_selection import train_test_split

#train, test 8:2로 나누기
X_trn, X_test, y_trn, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
print(X_trn.shape, y_trn.shape, X_test.shape, y_test.shape)

#train set을 8:2로 train, validation 나누기
X_trn, X_val, y_trn, y_val = train_test_split(X_trn, y_trn, test_size=0.2, random_state = 42)
print(X_trn.shape, y_trn.shape, X_val.shape, y_val.shape)
```

〈그림 15〉 데이터 스케일링 및 데이터 분할 코드

데이터 변수들을 일정한 범위로 조정하는 데이터 스케일링의 경우, 중앙값과 4분위수를 사용하여 이상치에 상대적으로 덜 민감한 Robust Scaling과 정규분포로 변환하는 Standard Scaler의 사이의 성능 차이가 매우 미비하여 Standard Scaling을 진행하였다.
또한 과적합을 방지하기 위해 모델에 학습할 데이터를 train set, validation set, test set로 각각 6:2:2로 나누어 진행하였다.

3.3.2 평가지표

```
# 평가지표
from sklearn.metrics import accuracy_score, balanced_accuracy_score, precision_score, recall_score, confusion_matrix, f1_score, roc_auc_score

def get_clf_eval(y_val, pred=None, pred_proba=None):
    confusion = confusion_matrix(y_val, pred)
    accuracy = accuracy_score(y_val, pred)
    balanced_acc = balanced_accuracy_score(y_val, pred)
    precision = precision_score(y_val, pred, average='macro') # 'macro' average for multiclass
    recall = recall_score(y_val, pred, average='macro') # 'macro' average for multiclass
    f1 = f1_score(y_val, pred, average='macro') # 'macro' average for multiclass
    weighted_f1 = f1_score(y_val, pred, average='weighted')
    roc_auc = roc_auc_score(y_val, pred_proba, multi_class='ovr') # for multiclass ROC AUC
    print('오차 행렬:')
    print(confusion)
    print('정확도: {:.4f}, Balanced 정밀도: {:.4f}, 정밀도: {:.4f}, 재현율: {:.4f}, F1: {:.4f}, Weighted F1:{:.4f} AUC:{:.4f}\n'.format(accuracy, balanced_acc, recall, f1, weighted_f1, roc_auc))
```

〈그림 16〉 평가지표를 하나의 함수로 묶은 코드

평가지표로는 정확도, 정밀도, 재현율 F1 Score, ROC AUC Score을 추출하였다. 위 프로젝트에서는 다중 분류를 예측하는 모델 생성을 목표로 하기 때문에 최종 모델 선정에 있어서는 전체 데이터 중 올바르게 예측한 비율인 '정확도'와 정밀도와 재현율의 조화 평균으로, 양성 샘플의 균형 정도를 평가하는 'F1 Score'을 기준하여 평가하였다.

3.3.3 데이터 모델링

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb
import lightgbm as lgb

# LogisticRegression
lg_classifier = LogisticRegression(solver='liblinear', random_state=42)

# RandomForest
rf_classifier = RandomForestClassifier(random_state=42)

# KNN
knn_classifier = KNeighborsClassifier(n_neighbors=10)

# XgBoost
xgb_classifier = xgb.XGBClassifier(random_state=42)

# LightGBM
lgb_classifier = lgb.LGBMClassifier(random_state=42)

# classifiers 학습/예측/평가
classifiers = [lg_classifier, rf_classifier, knn_classifier, xgb_classifier, lgb_classifier]
for classifier in classifiers:
    classifier.fit(X_trn, y_trn)
    pred = classifier.predict(X_val)
    pred_proba = classifier.predict_proba(X_val)
    class_name = classifier.__class__.__name__
    print(f'{class_name}')
    get_clf_eval(y_val, pred, pred_proba)
```

```

LogisticRegression
오차 행렬:
[[ 13  8  0 26  6  0]
 [ 5 75  0  1  0  0]
 [ 0  1  0  7  0  0]
 [ 6  0  0 1510 20  0]
 [ 0  0  0  1 699  1]
 [ 0  0  0  0 10] ]
정확도: 0.9649, Balanced 정확도: 0.6914, 정밀도: 0.7106, 재현율: 0.6914, F1: 0.6923, Weighted F1:0.9597 AUC:0.8942

RandomForestClassifier
오차 행렬:
[[ 7 11  0 29  6  0]
 [ 2 78  0  1  0  0]
 [ 0  1  0  7  0  0]
 [ 1  0  0 1515 20  0]
 [ 0  0  0  0 702  1]
 [ 0  0  0  0 10] ]
정확도: 0.9670, Balanced 정확도: 0.6800, 정밀도: 0.7360, 재현율: 0.6800, F1: 0.6749, Weighted F1:0.9586 AUC:0.8955

KNeighborsClassifier
오차 행렬:
[[ 0  2  0 48  3  0]
 [ 0  2  0 73  6  0]
 [ 0  0  0  8  0  0]
 [ 0  0  0 1485 54  0]
 [ 0  0  0  0 215  1]
 [ 0  0  0  3  6  0] ]
정확도: 0.7131, Balanced 정확도: 0.3159, 정밀도: 0.4722, 재현율: 0.3159, F1: 0.3343, Weighted F1:0.6568 AUC:0.7537

XGBClassifier
오차 행렬:
[[ 18  5  0 24  6  0]
 [ 6 74  0  1  0  0]
 [ 1  0  0  7  0  0]
 [ 10  1  0 1505 20  0]
 [ 3  0  0  0 699  1]
 [ 0  0  0  0 10] ]
정확도: 0.9645, Balanced 정확도: 0.7046, 정밀도: 0.7085, 재현율: 0.7046, F1: 0.7043, Weighted F1:0.9610 AUC:0.9027

LGBMClassifier
오차 행렬:
[[ 12  9  0 26  6  0]
 [ 9 71  0  1  0  0]
 [ 1  0  0  0  0  0]
 [ 9  0  0 1507 20  0]
 [ 0  0  0  1 699  1]
 [ 0  0  0  0 10] ]
정확도: 0.9615, Balanced 정확도: 0.6797, 정밀도: 0.6836, 재현율: 0.6797, F1: 0.6786, Weighted F1:0.9569 AUC:0.9065

```

〈그림 17〉 Logistic Regression, Random Forest, KNeighbor, Light GBM, XGBoost 학습/예측/평가 코드 및 결과

먼저 기본 설정된 머신 러닝 분류 알고리즘 중 Logistic Regression, Random Forest, KNeighbor, Light GBM, XGBoost를 사용하여 학습한 결과는 위와 같았다. 이 중 Light GBM을 선정하여 하이퍼파라미터를 파인튜닝하여 성능을 높이고자 하였다.

```

import numpy as np
from scipy.stats import randint
from imblearn.combine import SMOTEENN
from sklearn.model_selection import RandomizedSearchCV
import lightgbm as lgb

# SMOTEENN 를 통한 오버샘플링과 언더샘플링 조합
smote_enn = SMOTEENN(random_state=42)
X_trn_resampled, y_trn_resampled = smote_enn.fit_resample(X_trn, y_trn)

# LightGBM
lgb_model = lgb.LGBMClassifier()

# hyperparameter 본보기
param_distr = {
    'n_estimators': np.arange(100, 3001, 100), # 결정 트리의 개수
    'num_leaves': randint(20, 50),
    'learning_rate': np.arange(0.005, 0.31, 0.001), # 학습률
    'max_depth': np.arange(5, 30, 1) # 트리의 최대 깊이
    'min_child_samples': randint(10, 50),
    'subsample': np.arange(0.8, 1.0, 0.1), # 학습시 데이터 샘플링 비율
    'colsample_bytree': np.arange(0.8, 1.0, 0.1) # 학습시 피처 샘플링 비율
}

# 하이퍼파라미터 투닝을 위한 RandomSearchCV 이용
random_search = RandomizedSearchCV(estimator=lgb_model,
                                     param_distributions=param_distr,
                                     n_iter=10, # 10번만 투자하니 무독위 하이퍼파라미터 조합 시도
                                     scoring='roc_auc_ovr',
                                     cv=3, # 3 폴드 교차 검증 수행
                                     verbose=2, # 상세한 로그 출력
                                     n_jobs=-1, # 모든 기동한 CPU 코어 사용
                                     random_state=42)

random_search.fit(X_trn_resampled, y_trn_resampled)

# 최적의 하이퍼파라미터 조합 출력
best_lgbm_model = random_search.best_estimator_
best_params = random_search.best_params_
print("Best parameters found: ", best_params)

오차 행렬:
[[ 14  8  0 25  6  0]
 [ 7 73  0  1  0  0]
 [ 1  0  0  7  0  0]
 [ 12  1  0 1504 19  0]
 [ 0  0  0  2 700  1]
 [ 0  0  0  0 10] ]
정확도: 0.9624, Balanced 정확도: 0.6900, 정밀도: 0.6923, 재현율: 0.6900, F1: 0.6881, Weighted F1:0.9581 AUC:0.8946

# 최적의 하이퍼파라미터 기준으로 모델 생성
best_lgbm_model = lgb.LGBMClassifier(
    n_estimators=2200,
    num_leaves=48,
    learning_rate=0.05799999999999996,
    max_depth=14,
    min_child_samples=13,
    subsample=1.0,
    colsample_bytree=0.9

# 학습
best_lgbm_model.fit(X_trn, y_trn)

# 평가
y_pred = best_lgbm_model.predict(X_val)
y_pred_proba = best_lgbm_model.predict_proba(X_val)
get_clf_eval(y_val, y_pred, y_pred_proba)

```

〈그림 18〉 LightGBM 하이퍼파라미터 튜닝 과정과 최적 파라미터 학습/예측/평가 코드 및 결과

'n_estimators', 'num_leaves', 'learning_rate', 'max_depth', 'min_child_samples', 'subsample', 'colsample_bytree'의 파라미터 범위를 설정하여 RandomSearchCV를 사용하여 최적의 하이퍼 파라미터를 설정하여 학습한 결과 정확도는 0.9615에서 0.9624로 F1 Score는 0.9569에서 0.9581로 향상된 것을 확인할 수 있었다.

```

from sklearn.ensemble import VotingClassifier
# Soft Voting 기반 양상률
# 기별 모델로 XgBoost, LightGBM 사용
vo_classifier = VotingClassifier(estimators=[('XGB', xgb_classifier), ('LGB', best_lgbm_model)], voting='soft')
# VotingClassifier 학습/예측/평가
vo_classifier.fit(X_trn, y_trn)
pred = vo_classifier.predict(X_val)
pred_proba = vo_classifier.predict_proba(X_val)
print("Tuned Voting Classifier")
get_clf_eval(y_val, pred, pred_proba)

```

〈그림 19〉 Soft Voting Ensemble 학습/예측/평가 코드 및 결과

또한 성능이 가장 높았던 XGBoost와 최적의 파라미터로 튜닝한 LightGBM을 Soft Voting Ensemble로 구현하여 학습하였다. 학습 결과 정확도와 F1 Score는 튜닝된 LightGBM보다는 높았지만, XGBoost보다는 미세하게 낮은 걸 확인할 수 있었다.

```

from imblearn.combine import SMOTEENN
from catboost import CatBoostClassifier
smote_enn = SMOTEENN(random_state=42)
X_trn_resampled, y_trn_resampled = smote_enn.fit_resample(X_trn, y_trn)
catbst_model = CatBoostClassifier(random_seed=42)
catbst_model.fit(X_trn_resampled, y_trn_resampled, eval_set=(X_val, y_val))
pred = catbst_model.predict(X_val)
pred_proba = catbst_model.predict_proba(X_val)
bestTest = 0.2236134218
bestIteration = 932
Shrink model to first 933 iterations.
오차 행렬:
[[ 27  6  0 15  5  0]
 [ 5 75  0  1  0  0]
 [ 0  1  0  7  0  0]
 [ 39  1  0 1476  20  0]
 [ 10  0  0  0 692  1]
 [ 0  0  0  0  0 10]]
정확도: 0.9536, Balanced 정확도: 0.7301, 정밀도: 0.6826, 재현율: 0.7301, F1: 0.7029, Weighted F1: 0.9553 AUC: 0.9105

```

〈그림 20〉 CatBoost 학습/예측/평가 코드 및 결과

이후에는 범주형 변수들을 효과적으로 처리하는 CatBoost 모델을 사용하여 데이터를 학습시켰지만, 학습 결과 정확도와 F1 Score는 여전히 XgBoost 보다 미세하게 낮은 걸 확인할 수 있었다.



〈그림 21〉 모형별 분류 성능 평가

결과적으로 LogisticRegression, Voting Ensemble, RandomForest, LightGBM, CatBoost의 5가지 분류 알고리즘을 사용해 ‘피해운전자의 상해 정도’를 예측해본 결과, XgBoost 알고리즘이 정확도는 0.9645, F1 Score는 0.961로 가장 높은 값을 나타내어 이를 최종 예측 모델로 채택하였다.

3.4 Feature Importance

```

import numpy as np
import matplotlib.pyplot as plt

# XGBoost feature importances
xgb_importances = xgb_classifier.feature_importances_
lgb_importances = best_lgbm_model.feature_importances_

# 두 모델의 feature importances를 평균하여 결합
avg_importances = np.mean([xgb_importances, lgb_importances], axis=0)

# Feature names 설정
if isinstance(X_trn, np.ndarray):
    features = [f'Feature {i}' for i in range(X_trn.shape[1])]
else:
    features = X_trn.columns

# Feature importances 시각화 (상위 50개 피처만)
indices = np.argsort(avg_importances)[-50:] # 상위 50개의 피처 선택

plt.figure(figsize=(15, 12))
plt.title('Top 50 Feature Importances')
plt.barh(range(len(indices)), avg_importances[indices], align='center')
plt.yticks(range(len(indices)), [X_trn.columns[i] for i in indices])
plt.xlabel('Average Feature Importance')
plt.show()

# y축에 사용할 특정 풀 선택
selected_columns = ['사고시간대', '요일', '사고구', '사고내용', '법규위반', '노면상태', '기상상태', '도로형태', '가해운전자_차종', '가해운전자_연령대', '사고유형']

# 데이터 크기 맞추기
min_length = min(len(xgb_importances), len(selected_columns))
avg_importances = avg_importances[min_length]
features = selected_columns[min_length]

# Feature importances 시각화
indices = np.argsort(avg_importances)

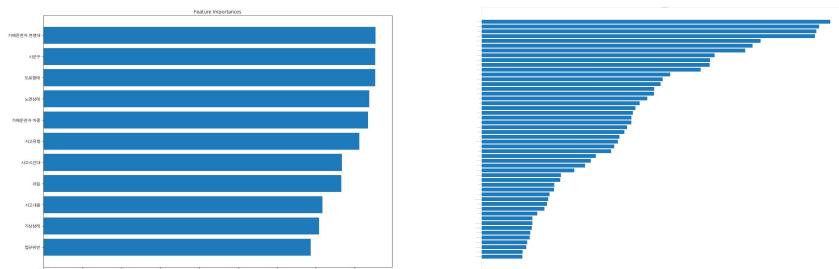
plt.figure(figsize=(15, 12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), avg_importances[indices], align='center', color="#232f52")
plt.yticks(range(len(indices)), [selected_columns[i] for i in indices])
plt.xlabel('Average Feature Importance')
plt.show()

# 순위와 함께 피처 내용 출력
for rank, index in enumerate(indices[-1], start=1): # 높은 순위부터 출력
    print(f'Rank [{rank}]: {features[index]} - Importance: {avg_importances[index]:.4f}')

```

〈그림 22〉 Feature Importance 추출 코드(사진 예시는 Soft Voting Ensemble을 기반으로 함)

모델 선정 이후, 모델 예측 과정에 있어 독립변수 칼럼들의 기여도와 상위 50개의 세분화된 개별 데이터의 중요도를 Feature Importance를 통해 출력해보았다. 칼럼별 Feature Importance는 ‘가해운전자 연령대’, ‘사고구’, ‘도로형태’, ‘노면상태’, ‘가해운전자_차종’, ‘사고유형’, ‘사고시간대’, ‘요일’, ‘사고내용’, ‘기상상태’, ‘법규위반’ 순으로 높았다. 세부적인 피처 차원에서 살펴보면, ‘가해운전자_차종_이륜’, ‘법규위반_안전운전불이행’, ‘도로형태_단일로기타’, ‘사고유형_차대차-기타’, ‘사고시간대_19:00-21:00’ 등이 상위 데이터 특성에 해당되었다.



〈그림 23〉 (좌)칼럼별 Feature Importance (우)상위 50개 Feature Importance

3.5 모델 활용

이후 ChatGPT API를 연결하여 프롬프트를 작성한 후 모델이 예측한 피해운전자 상해정도 값에 대하여 자동적으로 산재보험 처리 과정을 안내하는 메시지를 출력할 수 있도록 하였다. ChatGPT의 프롬프트 엔지니어링 원문은 다음과 같다.

prompt = f"배달 라이더가 배달 운전 중에 피해사고로 {accident_content} 사고가 생겼을 때 산재 보험 처리에 필요한 정보(분비해야 하는 서류 목록이나 병원 정보 등)와 밟아야 하는 행정상의 절차를 문자 메세지 안내형식으로 알려줘. 우리나라 산재 지정 의료기관을 알려면 어떻게 해야하는지도 함께 알려줘. 문자를 받는 사람은 배달라이더 사고 피해자이고, 대한민국 특성상 산재보험에 대한 인식이 미비하기 때문에 이에 대한 안내 문자 메세지를 생성하려고 해"

{accident_content}‘에 ’피해운전자 상해 정도‘ 예측값이 입력되도록 하였다.

```
import openai
import os
import pandas as pd
import time

openai.api_key = "# 여기에 실제 API 키를 입력하세요"

def get_completion(prompt, model="gpt-3.5-turbo"):
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0.5,
    )
    return response.choices[0].message["content"]

# pred 변수의 사고 내용의 예상
accident_mapping = {
    0: "신고",
    1: "부상신고",
    2: "기타불명",
    3: "경보",
    4: "경고",
    5: "사망"
}

# pred 값을 사고 내용으로 변환
pred_value = pred[0]
accident_content = accident_mapping.get(pred_value)

# 프롬프트 작성
prompt = f"메일 리아디가 배달 운전 중에 피해사고로 {accident_content} 사고가 생겼을 때 신고 사고 보험 처리에 필요한 정보(분비해야 하는 서류 목록, 산재 보상 신청 및 절차 단내, 1. 필요한 서류:
- 사고 신고서
- 의료서(진단서, 진료비 앤수증 등)
- 운전사증(운전증)
- 운전사증(운전증)
- 기타 관련 서류

2. 행정상의 절차:
- 사고 발생 후 즉시 1366(근로복지공단콜센터)로 연락하여 사고 신고
- 산재 보상 신청 및 절차 단내

3. 산재 지정 의뢰기관 확인:
- 근로복지공단 홈페이지(www.kcomwel.or.kr)에서 산재 지정 의뢰기관 확인 가능

신고 보험 처리에 관한 자세한 사항은 근로복지공단 산재보상부(☎ 1366)로 문의하시거나 홈페이지를 참고해주시기 바랍니다.

감사합니다. 함께 힘내시길 바랍니다.

# OpenAI API 호출
response = get_completion(prompt)
print(response)
```

〈그림 24〉 (좌)프롬프트 엔지니어링 코드 (우)ChatGPT를 통해 구현된 산재보험 안내 메세지

4. 결론 및 제언

본 연구는 교통사고 분석 시스템 데이터를 기반으로 이륜차 피해 운전자의 상해 정도를 예측하는 머신러닝 모델을 개발하고, 이를 활용하여 배달 라이더의 안전을 위한 실시간 경고 시스템 및 산재 처리 지원 시스템을 구축하였다.

결론적으로 본 연구를 통해 개발된 시스템은 다음과 같은 기대 효과를 가져올 수 있다. 먼저, 배달 라이더 안전을 증진한다. 실시간 사고 위험 예측 및 경고를 통해 라이더의 안전 운전을 유도하고, 사고 발생 시 신속한 대응을 가능하게 하여 라이더의 안전을 확보할 수 있다. 두 번째로 산재 처리 효율성 향상, 산재 보험 처리 과정을 자동화하여 기업의 행정적 부담을 줄이는 동시에 라이더의 편의성과 산재 처리에 대한 진입장벽을 낮추기 때문에 산재 처리 과정을 효율화할 수 있다. 마지막으로 데이터 기반의 의사 결정을 지원할 수 있다. 플랫폼 기업은 수집된 데이터를 분석하여 사고 원인을 파악하고, 예방 대책을 수립하는 데 활용하여 더욱 안전한 배달 환경 처리 시스템을 구축할 수 있다.

향후 연구 제언으로는 다음과 같은 방향을 제시한다. 첫 번째로, 모델 성능 고도화이다. 다양한 데이터를 추가하고, 모델의 하이퍼 파라미터 튜닝을 적용하여 모델의 예측 정확도와 F1 Score을 최적화시킨다. 두 번째로, 경고 시스템 고도화이다. 위험 단계별 맞춤형 메세지 경고에서 나아가, 음성 인식 혹은 진동 기반 상호 작용 등을 구현하여 시스템의 효용성을 높일 수 있도록 한다. 세 번째로 산재 처리 지원 확대이다. 맞춤형 보험 정보 제공, 법률 상담 연계, 심리 상담 지원 등을 통해 라이더에게 종합적인 지원 서비스를 제공해야 한다. 네 번째로 데이터와 예측 모델 활용을 확대할 수 있도록 한다. EDA 시각화 결과를 통한 데이터 특성과 머신러닝 예측 모델을 통하여 보험료 할인, 도로 환경 개선 등의 사회적 가치를 창출할 수 있도록 한다.

본 연구는 이륜차 배달 라이더의 안전 처리 구축을 위한 시스템을 제안하며, 향후 지속적인 연구를 통해 더욱 발전된 시스템을 구축하여 배달 산업의 안전 문화를 선도할 수 있을 것으로 기대된다.

참고문헌

- 고용노동부, “(설명) 경향신문(12.16), 배달라이더 열에 아홉은 “산재보험 보상 경험 없다” 기사 관련“, https://www.moel.go.kr/news/enews/explain/enewsView.do?news_seq=14388
- 국토교통부, ”위험천만한 배달은 이제 그만! 안전한 배달 위해 민·관이 손 잡았다“, https://www.molit.go.kr/USR/NEWS/m_71/dtl.jsp?id=95089731
- 권제인, ”‘배민라이더스쿨’ 하남시대 연다“, 헤럴드경제, 2024년 3월 28일, <https://news.heraldcorp.com/view.php?ud=20240328050384>
- 배민다움, ”라이더 사고율을 47% 줄인 기술“, <https://story.baemin.com/980/>
- 우아한 라이더 살핌기금, ”기금 소개“, <https://woowarider.or.kr/introduce>
- 우아한 형제들, ”우아한청년들, 배달노조와 단체협약 체결... “선도적 배달업 상생보델 확산”“, <https://www.woowahan.com/report/detail/480>
- 조해람, “배달라이더 열에 아홉은 “산재보험 보상 경험 없다”, 경향신문, 2022년 12월 15일, <https://www.khan.co.kr/national/national-general/article/202212152103015>
- 한국교통연구원. 『2022 이륜차 운전자 조사보고서』, 한국교통연구원. 2023, https://www.koti.re.kr/user/bbs/frghtSrvTwoView.do?bbs_no=55466
- Coupang eats services, ”쿠팡이츠 배달파트너 이용 방법“, https://ces.coupageats.com/?page_id=349
- Grab, ”Complimentary Personal Accident Insurance“, <https://www.grab.com/my/transport/personal-accident/>
- Uber, ”Drive with confidence“, https://www.uber.com/us/en/drive/safety/?uclick_id=472ff159-2dde-4863-8d24-d8d938ba78b7
- Uber, ”Our commitment to safety“, https://www.uber.com/us/en/safety/?uclick_id=472ff159-2dde-4863-8d24-d8d938ba78b7