우버데이터_Multiple_Linear_Regression

Multiple Linear Regression with Uber Dataset

> **다중 선형 회귀란?**
>
> - 여러 개의 독립 변수를 사용하여 종속 변수를 예측하는 통계적 방법
> - 수식:
>
> $$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon$$
>
> - β_0 : 절편, β_1 ~ β_n : 회귀 계수, ϵ : 오차항

Import packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

np.set_printoptions(precision=6, suppress=True)
```

데이터셋 로딩

> **Modified Uber Dataset**
>
> - **종속 변수**: fare_amount (우버 요금 지불 금액)
> - **독립 변수**:
>   - pickup_x: 승객 탑승 x 좌표
>   - pickup_y: 승객 탑승 y 좌표
>   - dropoff_x: 승객 하차 x 좌표
>   - dropoff_y: 승객 하차 y 좌표
>   - passenger_count: 탑승 승객 수

```
# Download dataset file
# !wget "https://dongaackr-
my.sharepoint.com/:x:/g/personal/sjkim_donga_ac_kr/EYMXYk25h2VBtPXtu3QaBqoBJK4cK-TI9mamHoFGJRYn5Q?
e=8x2MTn&download=1" -q -O modified_uber.csv

# Load dataset file
data = pd.read_csv('modified_uber.csv')
data = data.drop(['Unnamed: 0.1', 'Unnamed: 0', 'key', 'pickup_datetime'], axis=1)
data = data.dropna()
```

L1, L2 이동거리 계산 및 열 추가

L1 Distance (맨해튼 거리):

L1_distance = |pickup_x - dropoff_x| + |pickup_y - dropoff_y|

L2 Distance (유클리드 거리):

L2_distance = √((pickup_x - dropoff_x)² + (pickup_y - dropoff_y)²)

- 절댓값 계산 함수: np.abs()
- 제곱근 (square root) 계산 함수: np.sqrt()

```
data['L1_distance'] = np.abs(data['pickup_x'] - data['dropoff_x']) + np.abs(data['pickup_y'] -
data['dropoff_y'])
data['L2_distance'] = np.sqrt((data['pickup_x'] - data['dropoff_x'])**2 + (data['pickup_y'] -
data['dropoff_y'])**2)

data # L1/L2_distance 열이 정상적으로 계산/추가되었는지 확인
```

데이터셋 전처리

가우시안 정규화 (Gaussian Normalization)

- 데이터를 평균이 0, 표준편차가 1인 정규분포 형태로 변환
- 수식:

$$x' = \frac{x - \mu}{\sigma}$$

- 목적:
  1. 서로 다른 스케일의 특성들을 동일한 스케일로 변환
  2. 모델의 학습 안정성과 성능 향상
  3. 이상치의 영향 감소

```
data = data.dropna()

# 데이터셋 가우시안 정규화
data_normalized = (data - data.mean()) / data.std()
data_normalized
```

택시요금 예측에 사용할 데이터 지정

```
# 예측에 사용할 데이터들에 대한 2차원 행렬 변환
X = np.array(data_normalized[['L1_distance', 'L2_distance', 'passenger_count']]) # 입력 데이터 설정
Y = np.array(data_normalized[['fare_amount']])

# Train dataset / Test dataset 분할
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1234)

# Train dataset 형상 확인
print(X_train.shape)
print(Y_train.shape)
```

Least Square Method 기반 선형 회귀 모델 구현

최소 제곱법 (Least Square Method)

- 예측값과 실제값의 차이(오차)의 제곱합을 최소화하는 방법

수식

```
XT = X.T
```

$$X^\top$$

```
XTX = np.dot(XT, X)
```

$$X^\top X$$

```
XTX_inverse = np.linalg.inv(XTX)
```

$$(X^\top X)^{-1}$$

```
XTY = np.dot(XT, Y)
```

$$X^\top Y$$

```
self.theta = np.dot(XTX_inverse, XTY)
```

$$\theta = (X^\top X)^{-1} X^\top Y$$

```
class LinearRegression_LSM():
    def __init__(self):
        self.theta = None

    def fit(self, X, Y):
        N = X.shape[0] # N = 입력 데이터 개수

        # 입력 X에 대해 bias 차원 추가
        bias = np.ones((N, 1))    # N x 1
        X = np.hstack([X, bias])  # N x 2

        # theta (W, b) 저장을 위한 배열 초기화
        self.theta = np.zeros(X.shape[1])

        # Least Square Method 수행
        XT = X.T
        XTX = np.dot(XT, X)
        XTX_inverse = np.linalg.inv(XTX)
        XTY = np.dot(XT, Y)

        self.theta = np.dot(XTX_inverse, XTY)
        return self.theta

    def predict(self, X):
        N = X.shape[0] # N = 입력 데이터 개수

        # 입력 X에 대해 bias 차원 추가
        bias = np.ones((N, 1)) # N x 1
        X = np.hstack([X, bias]) # N x 2

        Y_hat = np.dot(X, self.theta)
        return Y_hat
```

모델 학습 및 평가

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

```
# X_train, Y_train 데이터를 이용한 linear regression 수행 (학습)
model_LSM = LinearRegression_LSM()
```

```
theta = model_LSM.fit(X_train, Y_train)

## X_test, Y_test 데이터를 이용한 linear regression 성능 검증 (테스트)
# MSE 계산 함수 정의
def MSE(Y, Y_hat):
    error = Y - Y_hat
    mse = np.mean(error ** 2)
    return mse

# 테스트 데이터에 대한 예측 및 평가
Y_hat = model_LSM.predict(X_test)
mse = MSE(Y_test, Y_hat)
```

임의 데이터 X 입력 시 Y_hat 예측

가우시안 정규화:

$$x' = \frac{x - \mu}{\sigma}$$

가우시안 역정규화:

$$x = x' \cdot \sigma + \mu$$

새로운 데이터에 대한 예측

```
# 가우시안 정규화/역정규화를 위한 평균, 표준편차 저장
mean_array = data.mean()
std_array = data.std()

mean_L1 = mean_array['L1_distance']
mean_L2 = mean_array['L2_distance']
mean_passenger = mean_array['passenger_count']
mean_fare = mean_array['fare_amount']

std_L1 = std_array['L1_distance']
std_L2 = std_array['L2_distance']
std_passenger = std_array['passenger_count']
std_fare = std_array['fare_amount']

# 임의 데이터 X 생성
L1_distance = 250
L2_distance = 197
passenger_count = 2

# 각 입력 변수 X에 대한 정규화 수행
L1_distance_norm = (L1_distance - mean_L1) / std_L1
L2_distance_norm = (L2_distance - mean_L2) / std_L2
passenger_count_norm = (passenger_count - mean_passenger) / std_passenger

X_new = np.array([[L1_distance_norm, L2_distance_norm, passenger_count_norm]])

# 학습한 모델 theta를 이용해 Y_hat 예측
Y_hat = model_LSM.predict(X_new)

# 출력 변수 Y에 대한 역정규화 수행
Y_hat = (Y_hat * std_fare) + mean_fare

print(f"Y_hat = {Y_hat}")
```

주의사항

1. 새로운 데이터 예측 시 반드시 정규화 필요
2. 예측 결과는 역정규화하여 해석
3. L1, L2 거리는 실제 거리와 비례하도록 계산
4. 승객 수는 정수값으로 입력