Data Analysis

<programming>

조원준

Q1

문제0

```
data_1 = data_1.drop(columns=data_1.columns[[0,1]])
```

Pd.read_csv로 파일 3개를 불러오고, 1,2열은 key가 index id와 중복된 데이터가 있으므로 drop 시킵니다.

문제1

(1-1) Pd.to_datetime 함수와 dt.month 함수를 이용해서 새로운 month 열 데이터를 만들었습니다. 이후 두개의 데이터셋(data_2,data_3)를 pd.concat 함수를 사용하여 행을 기준으로 합치고, ignore_index=True 파라미터를 입력하여 인덱스 종속을 초기화 시켰습니다.

```
problem_1_2_df = pd.merge(problem_1_1_df,data_1,on='url')
```

(1-2) pd.merge 함수를 사용하여 problem_1_1_df와 data_1을 joint 시킵니다. Pd.merge 함수는 inner merge를 디폴트 값으로 가지고 있기 때문에, 기준('url')만 추가로 입력하여 완성시켰습니다.

문제2

(2-1) isnull 함수를 사용해서 null 여부의 Boolean 값을 데이터로 가지고 있는 데이터베이스를 만들고, 각 열을 기준으로 mean함수를 이용하여 평균을 구합니다

(2-2) 2-1 문제에서 확인한 null 비율에서 'fat_content'와 'calcium_content' 칼럼의 결측치 비율이 60%가 넘는다는 사실을 발견 한 후, drop 함수를 이용하여 그 두 칼럼을 제거합니다.

문제3

(3-1) if 문을 활용하여 함수를 생성합니다. Pd.isnull을 활용하여 문자열 또는 null값을 구분하고, 문자열이라면 split 함수를 활용하여 ','를 기준으로 분리하는 함수를 생성합니다.

(3-2) 각 mixed, country_mixed 열을 milk, country 열 바로 옆에 놓고 싶었고, 그래서 dataframe.insert 함수를 활용하고자 하였습니다. 중복참조 문제를 없애기 위해 .copy()함수를 사용하여 problem_3_fd 데이터를 만들고, apply함수를 활용하여 count_mixed 함수를 적용시킨 뒤에 insert 함수를 활용하여 각 위치에 맞는 데이터를 삽입하였습니다.

문제4 – problem_3_df의 두개의 열의 데이터가 모두 1이어야 하므로 비교연산자(&)를 활용하여 problem_4_df 를 생성하였습니다. 이 경우에서 논리연산자(and)를 쓰게 될 경우 series의 Boolean 값이 series로 산출되는지, Boolean 그 자체로 산출되는지 모호해지기 때문에 비트연산자인 비교연산자를 사용합니다.

문제5 –

```
import re
yel = re.compile('yellow',re.IGNORECASE)

problem_5_df = problem_4_df.copy()
problem_5_df['color'] = problem_5_df['color'].apply(lambda x:'yellow' if pd.notna(x) and yel.search(x) else x)
```

re 모듈을 통해 yellow 문자열을 찾는 모듈을 불러오고, ignorecase를 활용하여 대문자 식별은 하지 않습니다. 이후 중복참조 문제를 없애기 위해 .copy()함수를 사용하여 problem_5_fd 데이터를 선제적으로 만들고, apply 함수를 이용해서 yellow를 찾습니다. Apply 함수를 적용할 때 and가 사용되는데, null 값일 때 yel.search 함수가 적용되지 않는 부분을 방지하기 위해 pd.notna(x)를 먼저 판단합니다.

문제6 – value_counts() 함수는 series를 객체로 받기 때문에, country 열의 데이터를 시리즈로 불러오고, 이에 value_count 함수를 적용하여 country 객체를 만듭니다. 이후 .to_dict 함수를 사용하여 딕셔너리형태의 객체를 만듭니다.

Q2

아이디어: 일단 바닥에 구멍이 없다고 가정하고 문제를 풀고, 이후에 구멍을 고려했습니다. 구멍을 기준으로 input 리스트를 분리하고, 분리된 각각의 리스트에 구멍이 없다고 가정했을 때의 방법론을 똑같이 적용하여 더하면, 저류조에 고인 물의 총량을 구할 수 있습니다.

구현: 구멍이 없다고 가정했을 때, 저류조에 고인 물의 양을 구하는 것을 'trap_water' 함수로 구현하였습니다. Input 값은 저류조 리스트 데이터(list), output 값은 저류조에 고인 물의 양(int)입니다. 저류조 한 칸에 쌓일 수 있는 물의 양은 왼쪽, 오른쪽에서부터의 저류조 높이의 최댓값을 구한 뒤 그 두개의 최댓값중에서 더 작은 값을 의미합니다. 결국엔 파인 부분에 물이 고이게 되는데, 그 파인 부분에 담긴 물의 양을 결정하는 것은 좌측과 우측의 가장자리에 해당되고, 더 낮은 가장자리의 값일 것이기 때문입니다. 쌓일 수 있는 물의 높이를 구한 뒤, 만약 해당 칸의 저류조 자체의 높이가 물의 높이보다 크거나 같다면, 물이 고이지 않게 되고, 저류조 자체의 높이가 쌓일 수 있는 물의 높이보다 낮다면, 그 차이만큼 물이 차게 됩니다. 따라서, left_max, right_max 라는 두개의 리스트를 생성하고 (길이는 input 리스트의 길이) 각 칸에서 식별되는 왼쪽, 오른쪽의 최대 저류조 높이를 저장합니다. 이후 쌓일 수 있는 물의 높이를 water_level 변수에 저장 한 뒤에, 실제 높이와 비교해서 쌓일 수 있다면 trapped_water에 계속해서 더하여 전체 저류조에 고이는 물의 양을 계산합니다.

위의 구현된 함수를 가지고, segments 라는 리스트에, 구멍을 기준으로 분리된 새로운 리스트들을 저장합니다. 이후 segments에 저장된 리스트들 각각에 함수를 apply 시키고, 이를 더합니다.

<EDA>

문제0

이 데이터는 Order_ID를 key로 가지고 있기 때문에 중복데이터는 존재하지 않습니다. 데이터들을 관찰한 결과 결측치를 가지고 있는 데이터, 배송출발지의 위도 값이 infinite인 데이터,배달원의 나이가 20~60세의 범위에 들어오지 않는 데이터, order_rating이 5점을 초과하는 데이터 등이 존재한다는 문제점들을 발견하였고, 이들을 삭제, 대체 하였습니다.

```
# 총 데이터 개수 43739
print(df.shape[0])

# 데이터 삭제
df = df.dropna()
print(df.shape[0])

df = df[df['Order_Time'] != ' NaN ']
print(df.shape[0])

df = df[df['Store_Latitude'] !='infinite']
df = df.astype({'Store_Latitude': float})
df = df[(df['Store_Latitude']>0) & (df['Store_Longitude']>0) & (df['Drop_Latitude']>0) & (df['Drop_Longitude']>0)]
print(df.shape[0])

# 데이터 대체
age_mean = df[(df['Agent_Age'] >= 20) & (df['Agent_Age'] <= 60)]['Agent_Age'].mean()
df['Agent_Age'] = df['Agent_Age'].apply(lambda x: age_mean if (x<20 or x>60) else x)

df['Order_Rating'] = df['Order_Rating'].apply(lambda x: 5.0 if x >5 else x)

# 결측치 제거 후 총 데이터 개수 38781

processed_df = df.copy()
```

결측치들을 제거하고, 위도 결측치 infinite를 제거하였습니다. 위도 결측치를 포함하고 있는 데이터의 개수가 약 4000개로 전체 데이터의 약 10%를 차지하였지만, 다른 대체 방법을 강구함에 어려움을 겪어서 일단 삭제하였습니다. 나이의 경우 정상범위(20~60세)를 넘어갈 때 평균 나이 값으로, 평점의 경우 이상치(5점을 넘는 경우)를 5점으로 대체하였습니다.

문제1

```
plt.figure(figsize= (20, 12))
processed_df.boxplot(column='Delivery_Time', by='Category', grid=False, patch_artist=True,
                     boxprops=dict(facecolor='lightblue', color='blue'),
                     whiskerprops=dict(color='blue'),
                     medianprops=dict(color='red'),
                     flierprops=dict(marker='o', color='orange', alpha=0.5))

plt.title('Delivery Time Distribution by Category', fontsize=16)
plt.suptitle('')
plt.xlabel('Category', fontsize=14)
plt.ylabel('Delivery Time', fontsize=14)

plt.xticks(rotation=45)
plt.show()
```

```
<Figure size 1440x864 with 0 Axes>
```

■ 시각화를 진행했습니다. Boxplot을 그렸고, 특징적인 부분은 Grocery 항목의 Delivery

Time이 다른 카테고리와 비교하여 매우 낮다는 것입니다. 다른 항목들과 비교하여 식료품점의 접근성이 좋을 것이기 때문에 이러한 차이가 관찰되었다고 생각합니다.

문제2

```
processed_df[['Delivery_Time','Order_Rating']].corr()
```

0.0s

	Delivery_Time	Order_Rating
Delivery_Time	1.000000	-0.303724
Order_Rating	-0.303724	1.000000

- Delivery time과 order rating을 상관관계 분석을 살펴보면 약 -0.3으로 약한 음의 상관관계가 있는 것을 확인했습니다. 조금 더 시각적으로 파악하기 위해 넓은 범위에 흩뿌려져 있는 delivery time을 사분위수로 분류 한 뒤에 추가적인 시각화를 진행하겠습니다.

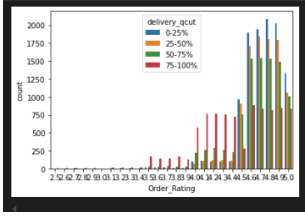
```
processed_df['Delivery_Time'].describe()
```

0.0s

```
count    38781.000000
mean       125.073747
std         51.955841
min         10.000000
25%         90.000000
50%        125.000000
75%        160.000000
max        270.000000
Name: Delivery_Time, dtype: float64
```

```
processed_df['delivery_qcut'] = pd.qcut(processed_df['Delivery_Time'], 4, labels=['0-25%', '25-50%', '50-75%', '75-100%'])
ax = sns.countplot(x='Order_Rating',hue='delivery_qcut',data=processed_df)
```

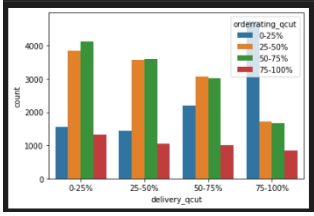
0.5s



- Delivery time을 사분위수로 분류한 뒤, order rating에 따른 delivery time의 분포를 시각화 했습니다. 4.5점 이전까지 배송시간 160분 이상에 해당되는 (빨간색) 4사분위수의 수가 많은 한편, 4.5 이상의 평점에 대해서는 오래 걸리지 않은 배송 건의 빈도가 급격하게 많아지는 것을 관찰할 수 있었습니다.

```
processed_df['orderrating_qcut'] = pd.qcut(processed_df['Order_Rating'], 4, labels=['0-25%', '25-50%', '50-75%', '75-100%'])
ax = sns.countplot(hue='orderrating_qcut',x='delivery_qcut',data=processed_df)
```

0.2s



- 배송이 오래 걸린 건일수록 낮은 평점을 부여하는 것으로 나타났습니다. 이를 통해 delivery time과 order rating은 음의 상관관계를 가지고 있으며, 특히 4.5점 이하의 점수를 부여한 배송건에게서 더 극단적으로 나타났습니다.

<modeling>

```
processed_df['distance'] = ((processed_df['Store_Latitude'] - processed_df['Drop_Latitude'])**2 + (processed_df['Store_Longitude'] - processed_df['Drop_Longitude'])**2)**(1/2)
processed_df['Delayed'] = processed_df['Delivery_Time'].apply(lambda x: 1 if x > 150 else 0)
processed_df.drop(columns=['Order_ID','Store_Latitude','Store_Longitude','Drop_Latitude','Drop_Longitude','delivery_qcut','orderrating_qcut','Order_Date','Order_Time','Pickup_Time','Delivery_Time'],axis=1,inplace=True)
processed_df.info()
```
✓ 0.0s

문제 1

- Eda 문항에서 사용했던 processed_df 데이터를 가지고 옵니다. 너무 많은 열 데이터를 가지고 있는 것을 방지하기 위해, 좌표 관련 데이터들은 거리 값을 가지는 열로 대체 한 후 drop 시켰습니다. 각 좌표 사이의 거리를 계산하고 'distance'라는 새로운 열을 만들어서 데이터를 하였습니다. Delivery_Time의 중간값은 150, 평균은 150.7이기 때문에 Delivery time이 150 이하인 행들은 0으로, 150분을 초과하는 행들은 1로 저장하는 Delayed 열을 만들어서 데이터를 생성합니다. 이후, 데이터 자체와는 크게 상관이 없는 order_id, distance 값으로 대체한 좌표 데이터, eda를 위해 생성했던 추가적인 데이터들(delivery_qcut,orderrating_qcut), 기존의 데이터들에 종속되어 있을 것이라 보여지는 시계열 데이터들(order_time, pickup_time, delivery_time)은 drop 시켰습니다. 시계열 데이터의 경우 예측 모델의 복잡성을 어느정도 최소화 시키기 위해 일단 배제하였습니다.

문제 2-1

```
x = pd.get_dummies(modeling_df.drop(columns=['Delayed']))
y = modeling_df['Delayed']
```
✓ 0.0s

- 정제된 데이터를 가지고 pd.get_dummies 함수를 이용하여 범주형 데이터들을 수치형 데이터로 인코딩 시킵니다. 독립변수를 저장할 x 에는 target인 'delayed' 열을 제거하여 인코딩을 하고, y에는 타겟 열의 데이터만 저장합니다.
- 정제된 데이터를 가지고 train-set과 test-set을 train_test_split 함수를 통해서 생성합니다. Train과 test 의 비율은 7:3 정도로 세팅합니다. 데이터들이 잘 split 되었는지 확인하기 위해 각 데이터 셋의 shape를 확인합니다.

```
model = DecisionTreeClassifier()
model.fit(x_train,y_train)
```
✓ 0.0s

- DecisionTreeClassifier 함수를 이용해서 모델을 만들고, train 셋을 이용해서 훈련합니다.

문제 2-2

```
print(classification_report(y_test,y_test_pred))
```
✓ 0.0s

	precision	recall	f1-score	support
0	0.91	0.90	0.91	8438
1	0.75	0.77	0.76	3197
accuracy			0.87	11635

- 전체 데이터 중 30%의 test 데이터들로 모델을 평가했고, 평가지표들의 값들은 다음과 같았습니다. 1(배달지연)을 예측할 때, Precision_score과 recall_score이 낮은 것으로 보아

모델의 예측 성능이 전반적으로 좋지 않음을 보여주고 있습니다. Precision의 낮음은, 모델이 1로 예측한 것들 중에서 실제로 1인 비율이 낮다는 의미이고, recall의 낮음은 모델이 실제 1인 데이터를 잘 찾아내지 못한다는 의미입니다. 따라서, 이 둘이 동시에 낮은 경우는 모델이 배달 시간이 150분을 초과하는 경우에 대한 예측에 전반적으로 부족함을 나타냅니다.

- 배달 시간이 상대적으로 오래 걸리는 배달을 파악하는 것을 중요하게 생각한다고 함은 precision, recall 평가지표를 개선해야 함으로, 이 둘의 속성을 모두 포함하는 f1-score을 모델 성능 개선의 기준으로 삼고자 합니다. 1(배달지연)의 f1-score는 0.76입니다.

문제 2-3

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=200, random_state=45)
clf.fit(x_train, y_train)
y_pred_clf = clf.predict(x_test)
print(classification_report(y_test,y_pred_clf))
```

✓ 7.2s

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93      8438
           1       0.81      0.80      0.81      3197

    accuracy                           0.89     11635
```

- 첫번째로, 앙상블 학습 방법의 일종인 random forest 예측 모델을 사용해보았습니다. 결과는 f1-score 0.81이 나왔습니다.

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93      8438
           1       0.81      0.80      0.81      3197

    accuracy                           0.89     11635
```

- 모델의 성능 차이가 크지 않다는 결과를 관찰하여 decisiontreeclassifier 모델로 돌아와서 gridsearch 모듈을 활용해서 최적의 하이퍼 파라미터를 탐색하고, 이를 적용시켜보았습니다. 0.81의 f1-score 이 나온 것을 확인했고, 적당한 하이퍼파라미터를 넣으면 거의 비슷한 f1-scrode이 나온다는 것을 알게 되었습니다.

```
#연속형 데이터 스케일링

from sklearn.preprocessing import MinMaxScaler

numeric_df = modeling_df[['Agent_Age','Order_Rating','distance']]
scaler = MinMaxScaler()
df_normalized = scaler.fit_transform(numeric_df)
df_normalized = pd.DataFrame(df_normalized, columns=numeric_df.columns,index=numeric_df.index)
modeling_df[['Agent_Age', 'Order_Rating', 'distance']] = df_normalized
```

✓ 0.0s

```
x = pd.get_dummies(modeling_df.drop(columns=['Delayed']))
y = modeling_df['Delayed']
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=1)
model =DecisionTreeClassifier(max_depth=12,min_samples_leaf=15)
model.fit(x_train,y_train)
y_test_pred = model.predict(x_test)
print(classification_report(y_test,y_test_pred))
```

✓ 0.2s

```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      8438
           1       0.84      0.83      0.84      3197

    accuracy                           0.91     11635
```

- 모델 중에 numeric 한 변수들을 스케일링 한 뒤에 모델을 만들어 다시 적용해보았습니다. 가장 높은 0.84의 f1-score 이 나왔습니다.

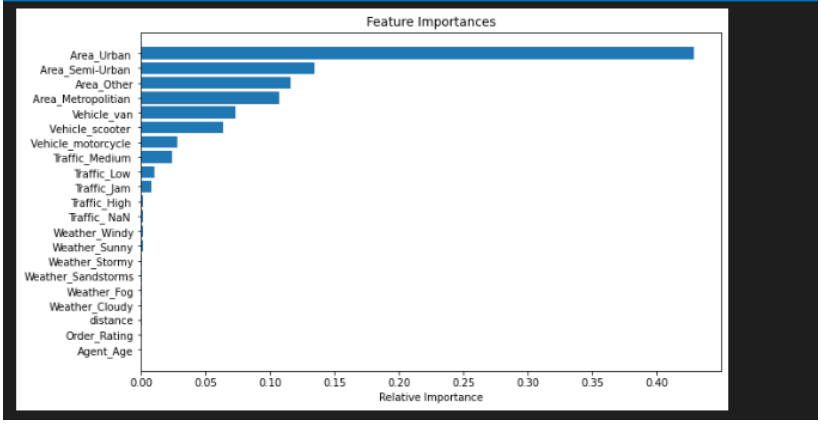
문제 2-4

```
# 스케일링 된 데이터 시각화

importances = model.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(10, 6))
plt.title('Feature Importances')
plt.barh(range(x.shape[1]), importances[indices], align='center')
plt.yticks(range(x.shape[1]), x.columns)
plt.xlabel('Relative Importance')
plt.show()
```

✓ 0.5s



- Feature_importances_를 이용하여 머신러닝 모델이 학습과정에서 사용된 각 특성의 중요도를 나타냈습니다. 가장 중요한 특성 Area_Urban 나타났으며, 이외에도 전반적으로 Area, Vehicle, Traffic 순으로 배달 지연에 영향을 주는 feature들이 관찰되었습니다.
- 이 분석결과가 도심지역에서의 배달이 지연의 원인이라고 단정지을 수는 없지만, 도시 지역에서의 배달 여부가 배달 지연 여부를 예측하는 데 가장 큰 영향을 미친다는 것을 의미합니다. 도시 지역에서의 배달이 다른 지역에 비해 지연될 가능성이 크다는 것은 교통체증, 높은 주문량등이 원인이 될 수 있습니다.
- 이 결과를 통해 배달 지연을 줄이기 위해서는 도시 지역의 배달에 관심을 둘 필요가 있다는 것을 알 수 있습니다. 이 문제를 해결하기 위해서 도심지역에 리소스(배달 인력 등)을 확충함으로써 배달 지연에 대한 배달 서비스 공급을 늘릴 수 있습니다. 또한 실시간 교통 정보 기반 경로 최적화를 지원하고, 배차 시스템의 개선, 그리고 배달 픽업 포인트를 설정하여 배달 자체의 효율성을 늘리는 방안으로 도심 지역의 배달지연 문제를 개선시킬 수 있습니다.

<전략도출>

HD현대에너지솔루션은 HD현대의 계열사로서 세계적 수준의 태양광 셀과 모듈을 생산하고 있는 기업입니다. 높은 기술력을 보유하고 있음에도 불구하고, 미국 IRA 법에 따른 세제 혜택을 받지 못하고, 중국의 저가 공세가 계속되는 등의 이유로, 글로벌 시장에서의 경쟁력 강화에 어려움을 겪고 있습니다. 이러한 문제를 해결하기 위해서, 기존의 한국 내수 시장에 국한된 전략을 탈피하고, 새로운 시장으로의 확장이 반드시 필요합니다. 중동 시장은 HD현대 계열사 기존 네트워크 강점을 활용할 수 있는 지역으로, 중국의 저가 공세가 아직 도달하지 않은 상태에서 차별화된 기술력과 신뢰도를 바탕으로 빠르게 진출할 수 있는 잠재력이 큽니다. 게다가 국내 태양광 설치량이 전년대비 15% 감소한 반면, 중동의 태양광 설치 규모는 2030년까지 800% 증가할 것으로 예상되는 만큼 중장기적 성장요인이 뛰어납니다. 이를 통해 HD현대에너지솔루션은 글로벌 시장에서의 입지를 강화하고, 장기적인 성장 동력을 확보할 수 있을 것입니다.