dacon_pdf

customers = pd.read_csv('./dacon_data/open/customers.csv')
locations = pd.read_csv('./dacon_data/open/locations.csv')
orders = pd.read_csv('./dacon_data/open/orders.csv')
order_items = pd.read_csv('./dacon_data/open/order_items.csv')
payments = pd.read_csv('./dacon_data/open/payments.csv')
products = pd.read_csv('./dacon_data/open/products.csv')
reviews = pd.read_csv('./dacon_data/open/reviews.csv')
sellers = pd.read_csv('./dacon_data/open/sellers.csv')

dacon_data_check 파일

8개의 df merge하고 필요없는 colums 삭제

temp = pd.merge(reviews,order_items, how='inner',on='Order_id')
temp = pd.merge(temp,orders, how='inner',on='Order_id')
temp = pd.merge(temp,payments, how='inner',on='Order_id')
temp = pd.merge(temp,products, how='inner',on='Product_id')
temp = pd.merge(temp,customers, how='inner',on='Customer_id')
temp = pd.merge(temp,sellers, how='inner',on='Seller_id')
temp =
temp.drop(["Review_creation_date","Review_answer_timestamp","Order_status","Payment_sequential","Payment_type","Payment_installments"],axis=1)
temp =
temp.drop(["Product_weight_g","Product_length_cm","Product_height_cm","Customer_zipcode_prefix","Seller_zipcode_prefix"],axis=1)
temp = temp.drop(["Product_width_cm"],axis=1)

원본데이터 해치지않기 위해서 copy

test= temp.copy() #test용

날짜형식으로 변환하고, 월별로 꺼내기

순이익 계산 + 실배송날짜와 기대배송날짜 차이 계산

test['Order_purchase_timestamp']=pd.to_datetime(test['Order_purchase_timestamp'])
test['YearMonth'] = test['Order_purchase_timestamp'].dt.strftime('%Y%m') #월별로 분류

test["earn"] = (test['Price']-test["Freight_value"]) * test['Order_item_id'] #화물가치빼기
test'Order_delivered_carrier_date','Order_delivered_customer_date' =
test'Order_delivered_carrier_date','Order_delivered_customer_date'.apply(pd.to_datetime) #
format='%Y-%m-%d %H:%M:%S.%f'
test['time_gap'] = test['Order_delivered_customer_date'] - test['Order_delivered_carrier_date']

고객 주를 기준으로 전체 주문의 몇퍼를 차지하고있

test_city = test.groupby(["Customer_state"])["Customer_city"].count().reset_index()
test_city["ratio"]=test_city["Customer_city"].apply(lambda x : x/test.shape[0])

고객주를 기준으로 평균 배송시간 구하기

test_city_time_diff=test.groupby(["Customer_state"])["time_gap"].sum().reset_index()

고객주를 기준으로 리뷰 시간구하기

test_city_review=test.groupby(["Customer_state"])["Review_score"].sum().reset_index()

고객주를 기준으로 순이익 구하기

test_city_max = test.groupby(["Customer_state"])["earn"].sum().reset_index()

데이터 머지하기

test_city_max = pd.merge(test_city_payment_value,test_city_max,
how='inner',on='Customer_state')
test_city_max = pd.merge(test_city_time_diff,test_city_max, how='inner',on='Customer_state')
test_city_max = pd.merge(test_city_review,test_city_max, how='inner',on='Customer_state')

데이터 머지하기

test_city_ratio_earn= pd.merge(test_city, test_city_max, on='Customer_state', how='inner')

평균값들 계산하기

test_city_ratio_earn["time_mean_gap"] = test_city_ratio_earn["time_gap"] /
test_city_ratio_earn["Customer_city"]
test_city_ratio_earn["review_mean_gap"] = test_city_ratio_earn["Review_score"] /
test_city_ratio_earn["Customer_city"]

전체 번 순이익

all_earn_sum = test_city_ratio_earn["earn"].sum()

주별로 번 이익률 계산

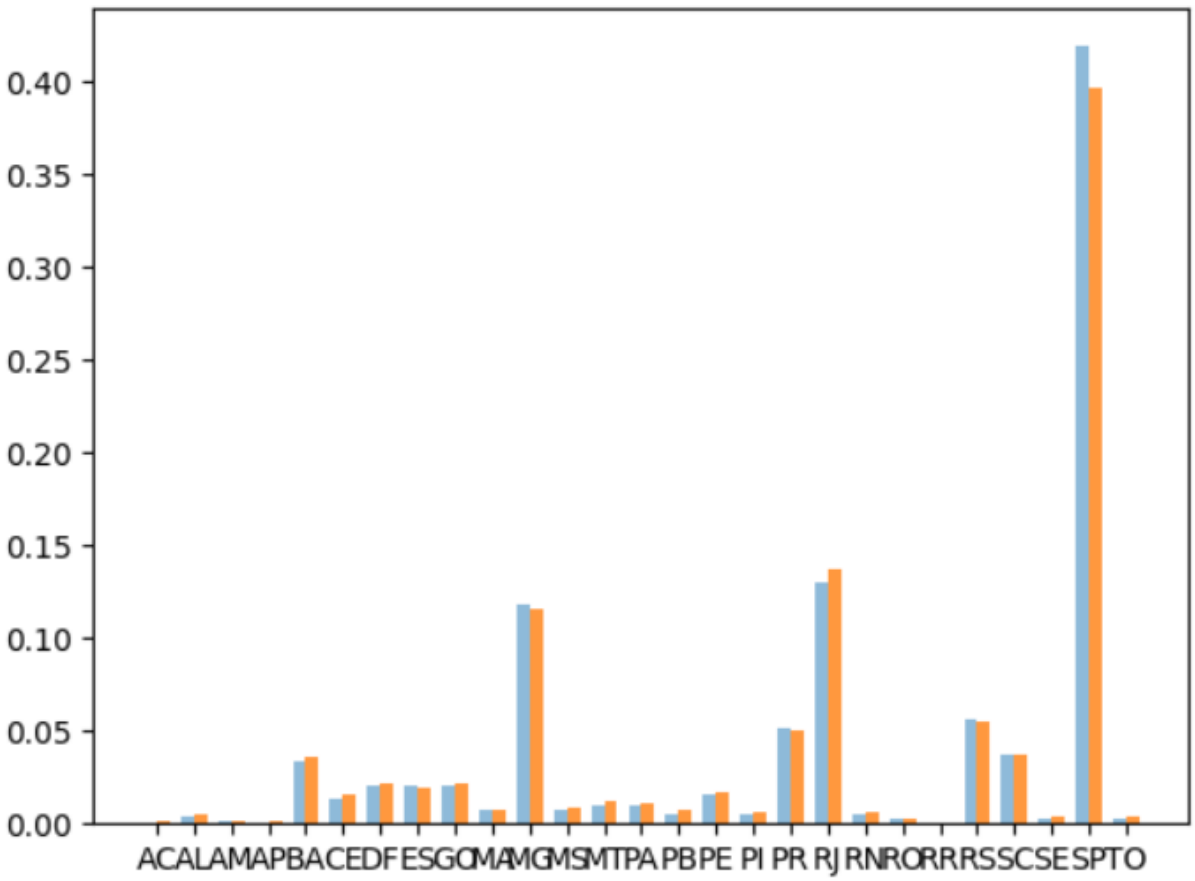
test_city_ratio_earn["earn_ratio"] = test_city_ratio_earn["earn"].apply(lambda x : x/all_earn_sum)

총 주의 개수

test_city_ratio_earn.Customer_state.size

주의 주문차지율과 주의 이익률 그래프

import matplotlib.pyplot as plt
x=np.arange(27)
width = 0.35
fig, axes = plt.subplots()
axes.bar(x - width/2, test_city_ratio_earn["ratio"], width, alpha = 0.5) #물품비율
axes.bar(x + width/2, test_city_ratio_earn["earn_ratio"], width, alpha = 0.8) #매출비율
plt.xticks(x)
axes.set_xticklabels(test_city_ratio_earn["Customer_state"])



주의 주문차지율에서 주의 이익률 차이 계산

test_city_ratio_earn["ratio_gap"] = test_city_ratio_earn["earn_ratio"] - test_city_ratio_earn["ratio"]

1퍼 이상이 가치가 있다고 생각하여 상위 14

test_city_ratio_earn_top14 = test_city_ratio_earn.sort_values(["earn_ratio"],ascending=False)[:14]

필요없는 컬럼값들 drop

test_city_ratio_earn = test_city_ratio_earn.drop(["Payment_value","Review_score","time_gap","earn"],axis=1)

배송평균을 ~일걸리는지만 출력

test_city_ratio_earn["just_days"] = test_city_ratio_earn["time_mean_gap"].apply(lambda x : int(str(x).split("days")[0]))

상관계수

test_city_ratio_earn.corr(numeric_only=True, method='pearson')

	Customer_city	ratio	review_mean_gap	earn_ratio	ratio_gap	just_days
Customer_city	1.000000	1.000000	0.254157	0.999465	-0.835409	-0.653167
ratio	1.000000	1.000000	0.254157	0.999465	-0.835409	-0.653167
review_mean_gap	0.254157	0.254157	1.000000	0.243676	-0.386096	-0.235865
earn_ratio	0.999465	0.999465	0.243676	1.000000	-0.816978	-0.666595
ratio_gap	-0.835409	-0.835409	-0.386096	-0.816978	1.000000	0.314225
just_days	-0.653167	-0.653167	-0.235865	-0.666595	0.314225	1.000000

just_day의 상관계수를 보면 음의 상관계수를 띄므로 just_day의 값을 낮추면 earn_ratio가 증가한다

값 확인

test_city_ratio_earn.sort_values(["ratio_gap"]) #양수일수록좋은거
test_city_ratio_earn_minus = test_city_ratio_earn[test_city_ratio_earn.ratio_gap <0]

주에따라 실질적으로 버는 비율과 그 비율에 대해서 차이가 얼마나 나는지

test_city_ratio_earn_minus["temp"] = -(test_city_ratio_earn_minus["earn_ratio"] / test_city_ratio_earn_minus["ratio_gap"])

	Customer_state	Customer_city	ratio	time_mean_gap	review_mean_gap	earn_ratio	ratio_gap	just_days	temp
7	ES	2108	0.020085	12 days 07:48:20.936907020	3.987192	0.018909	-0.001175	12	16.091592
10	MG	12380	0.117954	-9 days +16:49:01.299458033	4.110016	0.115097	-0.002857	-9	40.284269
17	PR	5396	0.051412	8 days 21:43:52.494069681	4.126019	0.050762	-0.000650	8	78.112173
21	RR	42	0.000400	24 days 15:52:45.142857142	3.785714	0.000398	-0.000002	24	197.735710
22	RS	5883	0.056052	12 days 06:03:25.777834438	4.061533	0.054940	-0.001112	12	49.408114
25	SP	43918	0.418442	0 days 19:52:20.163971274	4.175964	0.396472	-0.021970	0	18.046004

그러므로 ES->SP->MG->RS를 올려야한다
그 지역이 벌어들이는 매출비율 / RATIO GAP(전체지역에서 차지하는 비율 - 벌어들이는 매출비율)

손해보는 개수는

es = test[test.Customer_state=="ES"]
sp = test[test.Customer_state=="SP"]
mg = test[test.Customer_state=="MG"]
rs = test[test.Customer_state=="RS"]
es[es.earn<0].shape 해보면
111개 794개 466개 272개
es_minus = es[es.earn<0] #손해를 보는게 111개
sp_minus = sp[sp.earn<0].shape #794개
mg_minus = mg[mg.earn<0].shape #466개
rs_minus = rs[rs.earn<0].shape #272

구매자 판매자 위도경도 수정

locations_group_lat=locations.groupby(["Geolocation_zipcode_prefix"])["Geolocation_lat"].mean().reset_index()
locations_group_lng=locations.groupby(["Geolocation_zipcode_prefix"])["Geolocation_lng"].mean().reset_index()
locations_group = pd.merge(locations_group_lat, locations_group_lng, how = 'inner', on="Geolocation_zipcode_prefix")
es_minus = pd.merge(es_minus, locations_group, left_on = 'Customer_zipcode_prefix', right_on = 'Geolocation_zipcode_prefix', how = 'inner')
es_minus = es_minus.rename(columns={'Geolocation_lat':'Customer_lat',"Geolocation_lng" :

"Customer_lng" })
es_minus = pd.merge(es_minus, locations_group, left_on = 'Seller_zipcode_prefix', right_on = 'Geolocation_zipcode_prefix', how = 'inner')
es_minus = es_minus.rename(columns={'Geolocation_lat':'Seller_lat',"Geolocation_lng" : "Seller_lng" })
es_minus.drop(["Geolocation_zipcode_prefix_x","Geolocation_zipcode_prefix_y","Customer_zipcode_prefix", "Seller_zipcode_prefix"], axis=1,inplace=True)

#earn 을 구하는 공식이 test["earn"] = (test['Price']-test["Freight_value"]) * test['Order_item_id']
#화물가치빼기

price의 고유값 확인후에 다양한 값이 있다면 리뷰평점에 따라서 높은 값을 주는 물건으로 추천을 해주자

es_minus_group = es_minus.groupby(["Product_id"])["Price"].unique().reset_index()
es_minus_group["price_choose"] = es_minus_group["Price"].apply(lambda x : len(x))
es_minus_group_length_list = es_minus_group["price_choose"].unique() #1과 2뿐이므로
es_minus_product_id_list = es_minus_group[es_minus_group.price_choose==2]["Product_id"]
#4가지의 이름에서만 조절

넓게 바라봐서 음수인 df에서는 손실을 최소화할수는 있으나

sp로 다시 한번 테스트

여기서 바꿀수있는값이 freight_Value를 줄이는법분

데이터 모으는 데이터프레임!!!

all_data_df=pd.DataFrame()
저기에 이제 ES->SP->MG->RS 이 나라들에 속하는 애들중 product_id가 똑같지만 가격이 2개 이상인것을 고른다, 가격이 높은걸 팔수록 회사에게 이득이므로

글고 원본데이터에서 저 product_id에 포함된 애들중에 max 순이익을 계산해서 최대이득을 계산해봄

only_one_df = pd.DataFrame()
for i in range(len(product_id_list)):
temp_max = test[test.Product_id==product_id_list[i]]["earn"].max()
temp_temp = test[(test.Product_idproduct_id_list[i]) & (test.earntemp_max)].reset_index()

```
# print(a.shape[0])
if temp_temp.shape[0]1:
# print(temp_temp.shape[0])
only_one_df = pd.concat([only_one_df , temp_temp] , ignore_index=True) #성공
pass
else:
# print(temp_temp.loc[0])
temp_temp_1 = temp_temp.loc[0].to_frame().T
# display(temp_temp_1)
only_one_df = pd.concat([only_one_df , temp_temp_1] , ignore_index=True)
# display(temp_temp.iloc[0])
# pass
#test [test.Product_id"PRODUCT_04479"]["earn"].max()
all_data_df.earn_y.sum() - all_data_df.earn_x.sum() #이득은 2519원이다
```

dacon_data_check_1 파일

파일 읽고 머지한후 필요없는 컬럼 드랍

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
customers = pd.read_csv('./dacon_data/open/customers.csv')
locations = pd.read_csv('./dacon_data/open/locations.csv')
orders = pd.read_csv('./dacon_data/open/orders.csv')
order_items = pd.read_csv('./dacon_data/open/order_items.csv')
payments = pd.read_csv('./dacon_data/open/payments.csv')
products = pd.read_csv('./dacon_data/open/products.csv')
reviews = pd.read_csv('./dacon_data/open/reviews.csv')
sellers = pd.read_csv('./dacon_data/open/sellers.csv')
temp = pd.merge(reviews,order_items, how='inner',on='Order_id')
temp = pd.merge(temp,orders, how='inner',on='Order_id')
temp = pd.merge(temp,payments, how='inner',on='Order_id')
temp = pd.merge(temp,products, how='inner',on='Product_id')
temp = pd.merge(temp,customers, how='inner',on='Customer_id')
temp = pd.merge(temp,sellers, how='inner',on='Seller_id')
temp =
```

temp.drop(["Review_creation_date","Review_answer_timestamp","Order_status","Payment_sequential","Payment_type","Payment_installments"],axis=1)

우편번호로 위도경도 평균으로 통합하기

locations_group_lat=locations.groupby(["Geolocation_zipcode_prefix"])["Geolocation_lat"].mean().reset_index()
locations_group_lng=locations.groupby(["Geolocation_zipcode_prefix"])["Geolocation_lng"].mean().reset_index()

그리고 머지하기

locations_group = pd.merge(locations_group_lat, locations_group_lng, how = 'inner', on="Geolocation_zipcode_prefix")

고객 우편번호로 다른 컬럼명이므로 통합하기

temp = pd.merge(temp, locations_group, left_on = 'Customer_zipcode_prefix', right_on = 'Geolocation_zipcode_prefix', how = 'inner')

컬럼명 변경

temp = temp.rename(columns={'Geolocation_lat':'Customer_lat',"Geolocation_lng" : "Customer_lng" })

셀러에게도 반복

temp = pd.merge(temp, locations_group, left_on = 'Seller_zipcode_prefix', right_on = 'Geolocation_zipcode_prefix', how = 'inner')
temp = temp.rename(columns={'Geolocation_lat':'Seller_lat',"Geolocation_lng" : "Seller_lng" })

컬럼 드랍

temp.drop(["Geolocation_zipcode_prefix_x","Geolocation_zipcode_prefix_y","Customer_zipcode_prefix", "Seller_zipcode_prefix"], axis=1,inplace=True)

순수 고객과 판매자 거리계산

temp["Distance"] = ((temp["Customer_lat"] - temp["Seller_lat"])**2 + (temp["Customer_lng"] - temp["Seller_lng"])**2) **(1/2)

temp[temp.Freight_value==0] #384 0.3퍼라서 무시가능
temp[temp.Distance==0]["Freight_value"] #이동거리가 없는데 왜 요금을 받을까? 어떻게 처리할까=>무시하자
temp["Distance"].describe()

거리와 운임요금이 0인값들 무시

temp_drop = temp[temp.Distance!=0]
temp_drop = temp_drop[temp_drop.Freight_value!=0]

거리에 따른 요금 계산

temp_drop["Pay_distance"] = temp_drop["Freight_value"] / temp_drop["Distance"]
data_pay_distance = np.array(temp_drop.Pay_distance)

이상치 계산

Q1 = np.percentile(data_pay_distance, 25)
Q3 = np.percentile(data_pay_distance, 75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 *IQR*
upper_bound = *Q3* + *1.5* IQR
outliers = data_pay_distance[(data_pay_distance < lower_bound) | (data_pay_distance > upper_bound)]
이상치의 개수가 15112, 이상치에 포함이 안되는것은 104031개다
7퍼를 무시할수있을까?

temp_drop["Outliers"] = (temp_drop['Pay_distance'] < lower_bound) | (temp_drop['Pay_distance'] > upper_bound)

운임요금과 거리가 0인 값드을 무시할수있나 확인하기 위해 가치매기기

temp_drop["earn"] = (temp_drop['Price']-temp_drop["Freight_value"]) * temp_drop['Order_item_id'] #화물가치빼기

all_earn_sum = temp_drop["earn"].sum()
all_earn_sum # 11665320.41 운임요금과 거리가 0인걸 제거해도 큰차이가 나지않는다 #11762681.18
drop을 하나 안하나 sum값을 비교해도 큰 차이가 나지 않는다.

즉 무시해도 된다

이상치 의미 확인하기

temp_drop_Outliers = temp_drop.groupby(["Outliers"])["earn"].sum()

```
temp_drop_Outliers #True => 1928918.77
```

```
Outliers
False    9736401.64
True     1928918.77
Name: earn, dtype: float64
```

temp_drop_Outliers[1] / all_earn_sum #16퍼를 차지하기에 무시못한다 이상치를 고려해줘야 한다.

무게에 운임요금이 연관성이 있을까 str을 float으로 변

temp_drop['Product_weight_g'] = temp_drop['Product_weight_g'].replace('Unknown', np.nan).fillna(0).astype(float)

상관계수 해보기

```
temp_drop[['Freight_value', 'Distance',"Product_weight_g"]].corr()  #0.392 정도 운임요금과 거리 0.616무게와 더 연관성이 있다
#거의 40퍼와 60퍼
# 거리는 줄일수있지만 무게는 줄일수없다.
# 무게는 같은 품목일경우 근처로 정해주자
```

	Freight_value	Distance	Product_weight_g
Freight_value	1.000000	0.391769	0.616224
Distance	0.391769	1.000000	-0.008208
Product_weight_g	0.616224	-0.008208	1.000000

product_id에 따른 개수세기

product_id_counts = temp_drop['Product_id'].value_counts()

```
product_id_counts.describe()
```

```
count    29190.000000
mean         3.563926
std         10.550529
min          1.000000
25%          1.000000
50%          1.000000
75%          3.000000
max        497.000000
Name: count, dtype: float64
```

이러므로 10개 이상에서 효력이 있다고 생각
product_id_counts_over_10 = product_id_counts[product_id_counts >= 10]
product_id_df = product_id_counts.reset_index()
product_id_df_over_10 = product_id_df[product_id_df["count"] >=10]
product_id_list = product_id_df_over_10.Product_id #10개가 넘는 품목의 리스트를 저장
temp_drop_10 = temp_drop[temp_drop["Product_id"].isin(product_id_list)] #10개가 넘는 품목의 리스트를 저장하는 데이터프레임

나중에 교정후에 값 비교해보기위해서 값 저장

temp_drop_10.earn.sum() #총 합 4530190.869999999 나중에 비교해보기위해서
#같은 물건을 파는 사람들을 더 거리가 가까운 지점으로 옮기는거야

같은 물건을 파는 사람들의 최소값

min_values = temp_drop_10.groupby('Product_id')['Freight_value'].min().reset_index() #각 product_id에 따른 운임요금의 최소값
temp_drop_10_1 = temp_drop_10.copy()
temp_drop_10_merged = pd.merge(temp_drop_10_1,min_values, how='inner',on='Product_id')

기존의 earn 값을 덮기

temp_drop_10_merged.head(1) #기존의 earn을 덮자
temp_drop_10_merged["earn"] = (temp_drop['Price']-temp_drop["Freight_value"]) * temp_drop['Order_item_id']
temp_drop_10_merged.earn.sum() #수정후 바뀐값

차이

```
temp_drop_10_merged.earn.sum() - temp_drop_10.earn.sum()
775126.5199999996
```