

۱,۱) برای از بین بردن مقادیر NaN از forward fill استفاده شده است که هر جا که به NaN میرسد، مقدار موجود در ردیف قبلی را جایش میگذارد. در ادامه هم چک شد که هیچ NaNی وجود نداشته باشد (اگر ردیف اول NaN بود این مشکل وجود میداشت که در اون صورت میشد یک بار نیز backward fill انجام داد)

```
##### YOUR CODE STARTS HERE #####
train_df = train_df.fillna(method='ffill')
test_df = test_df.fillna(method='ffill')

#check to see if there are any NaN values left
print("Training set:")
print(train_df.isna().sum())
print("=====")
print("Test set:")
print(test_df.isna().sum())
#No NaN values left so this method is good enough
##### YOUR CODE ENDS HERE #####
```

[4] ✓ 0.4s Python

... Output exceeds the size limit. Open the full output data in a text editor

Training set:	
CustomerID	0
Age	0
Gender	0
City	0
State	0
No_of_orders_placed	0
Sign_up_date	0
Last_order_placed_date	0

۱,۲) ستون‌هایی که عددی هستند را انتخاب کرده (چون StandardScaler روی string کار نمیکند و همچنین اب وجود اینکه میشد برای هر رشته یک مقدار عددی جنریت کرد، انجام این کار روی این داده‌ها معنی نداشت به جز مثلاً تاریخ‌ها) و چون باید train و test هر دو نسبت به train نرمالایز شوند، fit را روی train انجام میدهم.

```
##### YOUR CODE STARTS HERE #####
cols = ["Age", "No_of_orders_placed", "Women's Clothing", \
        "...Men's Clothing", "Kid's Clothing", "Home & Living", "Beauty", "Electronics"]

scaler = StandardScaler().fit(train_df[cols])
train_df[cols] = scaler.transform(train_df[cols])
test_df[cols] = scaler.transform(test_df[cols])
##### YOUR CODE ENDS HERE #####
```

[5] ✓ 0.3s Python

```
#just checking the training set after standardization
train_df
```

[6] ✓ 0.3s Python

	CustomerID	Age	Gender	City	State	No_of_orders_placed	Sign_up_date	Last_order_placed_date	is_premium_member	Women's Clothing
0	CusID_00685	-1.233946	Not_Specified	Bercelona	Singapore	1.614207	2017-01-17	2020-09-19	1	0.124713
1	CusID_06121	0.355093	Male	Sydney	New South Wales	0.500834	2016-01-22	2021-12-09	0	-0.043409
2	CusID_09847	0.355093	Not_Specified	Toronto	Ontario	1.057520	2019-08-07	2021-10-13	1	0.152196
3	CusID_01433	0.090253	Not_Specified	Toronto	Ontario	1.892550	2016-02-27	2020-10-22	1	0.039057
4	CusID_02167	1.282032	Female	?	British Columbia	-1.447568	2019-07-04	2020-03-17	0	0.450816
...
10600	CusID_09355	1.679291	Male	Kuala Lumpur	Singapore	1.335864	2016-01-10	2021-09-18	1	0.139572

۱,۳) در این قسمت هم روی ستون‌ها خواسته شده one hot encoding را پیاده سازی میکنیم.

```
cols_to_be_encoded = ['Gender', 'State']

##### YOUR CODE STARTS HERE #####
train_df = pd.get_dummies(train_df, columns=cols_to_be_encoded)
test_df = pd.get_dummies(test_df, columns=cols_to_be_encoded)
##### YOUR CODE ENDS HERE #####
```

[8] ✓ 0.5s Python

```
train_df
```

[9] ✓ 0.3s Python

	CustomerID	Age	No_of_orders_placed	Sign_up_date	Last_order_placed_date	is_premium_member	Women's Clothing	Men's Clothing	Kid's Clothing	Home
0	CusID_09265	1.944131	1.614207	2019-04-17	2021-05-25	1	0.117018	0.269042	1.073437	
1	CusID_05791	0.355093	0.500834	2017-06-23	2021-07-30	0	-0.039028	1.288364	-1.767676	
2	CusID_07705	2.473810	1.335864	2017-09-29	2021-01-06	1	0.388687	-0.977850	1.562905	
3	CusID_08159	-0.439427	1.057520	2018-11-06	2020-09-02	0	0.313424	-1.484451	0.380756	

۲,۱) برای این قسمت، چون الگوریتم‌های ML همانطور که گفته شد با رشته نمیتوانند کار کنند، ستون‌های رشته اول drop شدند و سپس split انجام شده است.

```
##### YOUR CODE STARTS HERE #####
#dropping string columns since they cause some difficulties in the ML algorithms
traindf = train_df.drop('Sign_up_date', axis=1, inplace=False)
traindf.drop('Last_order_placed_date', axis=1, inplace=True)
traindf = traindf.set_index('CustomerID')

testdf = test_df.drop('Sign_up_date', axis=1, inplace=False)
testdf.drop('Last_order_placed_date', axis=1, inplace=True)
testdf = testdf.set_index('CustomerID')

##### YOUR CODE ENDS HERE #####

# Before that use train_test_split() to split your train data into train/validation
# in order to evaluate your model based on validation data

X_train, X_val, y_train, y_val = train_test_split(traindf.drop('Preferred_Theme', axis=1),
                                                traindf['Preferred_Theme'],
                                                test_size=0.2)
```

برای پیاده سازی، الگوریتم KNN انتخاب شده است که در حقیقت برای اینکه هر نقطه را در کلاس خود قرار دهد (اینجا کلاس‌هایمان دوست داشتن IU قدیم یا جدید است) فاصله آن تا نقاط دیگر را محاسبه میکند (فاصله اقلیدسی یا منتهن یا.. که اینجا اقلیدسی انتخاب شده است) و با توجه به اینکه بین k نقطه نزدیکتر، بیشترشان چه کلاسی دارند، کلاس این نقطه را پیش بینی میکند. برای محاسبه فاصله نقاط نیز فاصله اقلیدسی هر مقدار در هر ستون را با مقدار آن ستون در نقاط موجود در training set محاسبه و جمع میکند.

```
##### YOUR CODE STARTS HERE #####
#implementing KNN from scratch
#note that Scikit-learn uses a KD Tree or Ball Tree to compute nearest neighbors in O[N log(N)] time
#while this algorithm is a direct approach including nested loops that requires O[N^2] time
#therefore processing 900 points in x_val takes about 9 minutes

def euclidean_dist(p1, p2):
    p = p2 - p1
    return np.sum(p*p) #not using np.sqr because it doesn't change the result + it slows down the process

def KNN_predict(x_train, y_train, x_val, k):
    y_vals = []

    for i in range(len(x_val)):
        point_dist = []
        for j in range(len(x_train)): #number of rows in x_train
            distances = euclidean_dist(np.array(x_val.iloc[i]), np.array(x_train.iloc[j]))
            point_dist.append(distances)
        point_dist = np.array(point_dist)

        #sort the array while preserving the index
        dist = np.argsort(point_dist)[:k]

        #val of the K nearest datapoints
        vals = y_train[dist]

        #mode of val
        v = mode(vals)
        v = v.mode[0]
        y_vals.append(v)
    return y_vals

y_pred = KNN_predict(X_train, y_train, X_val, 5)
np.array(y_pred)

##### YOUR CODE ENDS HERE #####
```


