Question 1: Data understanding

- **Read data:**

  Import modules and read the three csv files with pandas. The three data frames user_features.csv, product_features.csv, and click_history.csv are named as userFeatures, proFeatures and click, respectively

- **Check the structure of data**

  userFeatures: has 12000 rows and 4 columns, with user_id as integer, number_of_clicks_before as object, ordered_before as boolen, personal_interest as object, and it has many values in a cell. It needs to be separated in the further steps.

```
# read user features data and check the data structure
userFeatures=pd.read_csv("user_features.csv")
userFeatures.info() # check the structure of the data
userFeatures.head() # check what the data looks like
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12000 entries, 0 to 11999
Data columns (total 4 columns):
user_id                  12000 non-null int64
number_of_clicks_before  11500 non-null object
ordered_before           12000 non-null bool
personal_interests       12000 non-null object
dtypes: bool(1), int64(1), object(2)
memory usage: 293.1+ KB
```

	user_id	number_of_clicks_before	ordered_before	personal_interests
0	104939	2	True	['body', 'makeup', 'nail', 'hand', 'foot', 'me...
1	101562	2	True	['men_skincare', 'men_fragrance', 'tools', 'sk...
2	102343	2	True	['tools', 'makeup', 'foot', 'nail']
3	106728	5	True	['hand', 'men_skincare']
4	107179	0	True	['makeup', 'body', 'skincare', 'foot', 'men_sk...

  proFeatures: has 1000 rows and 5 columns, with product_id as integer, category as object, on_sale as boolen, number_of_reviews as integer, avg_review_score as float.

```
# read product features data and check the data structure
proFeatures=pd.read_csv("product_features.csv")
proFeatures.info() # check the data structure
proFeatures.head() # check the head of the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
product_id           1000 non-null int64
category             1000 non-null object
on_sale              1000 non-null bool
number_of_reviews    1000 non-null int64
avg_review_score     1000 non-null float64
dtypes: bool(1), float64(1), int64(2), object(1)
memory usage: 32.4+ KB
```

	product_id	category	on_sale	number_of_reviews	avg_review_score
0	1134	tools	False	101	3.349452
1	1846	skincare	False	111	5.000000
2	1762	fragrance	False	220	4.882706
3	1254	hair	True	446	5.000000

  click: has 35990 rows and 3 columns, with user_id as integer, product_id as integer, clicked as boolen.

```
# read product features data and check the data structure
click=pd.read_csv("click_history.csv")
click.info() # check the structure of the dataframe
click.head() # take a look at the head of the data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35990 entries, 0 to 35989
Data columns (total 3 columns):
user_id       35990 non-null int64
product_id    35990 non-null int64
clicked       35990 non-null bool
dtypes: bool(1), int64(2)
memory usage: 597.6 KB
```

	user_id	product_id	clicked
0	104863	1350	False
1	108656	1321	True
2	100120	1110	False
3	104838	1443	True
4	107304	1397	True

- **Check the descriptive statistics of data**

userFeatures: the count of user_id=12000, informing there are 12000 users, other descriptive statistics is not informative since this variable is only the user's identification.

```
#get the summary of the c
userFeatures.describe() #
```

	user_id
count	12000.00000
mean	105999.50000
std	3464.24595
min	100000.00000
25%	102999.75000
50%	105999.50000
75%	108999.25000
max	111999.00000

proFeatures: For product_id: count=1000, indicating 1000 products. For number_of_reviews, minimum=66 and maximun=2307390, and it can be seen there is a big difference. For avg_review_score, it has a minimum of -1 and maximun of 5.

```
#Check the summary of the product features
proFeatures.describe() # Check the descriptive sta
```

	product_id	number_of_reviews	avg_review_score
count	1000.000000	1.000000e+03	1000.000000
mean	1499.500000	1.157725e+05	2.660656
std	288.819436	5.028997e+05	1.741875
min	1000.000000	6.600000e+01	-1.000000
25%	1249.750000	2.570000e+02	1.428969
50%	1499.500000	4.710000e+02	2.769397
75%	1749.250000	7.042500e+02	4.180860
max	1999.000000	2.307390e+06	5.000000

Click: The count of user_id and product_id is 35990. The other descriptive statistics are not informative since they are only id numbers.

```
#check the summary of the click history
click.describe()  # Check the descripti
```

	user_id	product_id
count	35990.000000	35990.000000
mean	106017.080161	1500.232898
std	3483.480090	288.101984
min	100001.000000	1000.000000
25%	102976.500000	1250.000000
50%	106060.000000	1503.000000
75%	109049.000000	1749.000000
max	111999.000000	1999.000000

Question 2: Data cleaning and preprocessing

- **Check duplicated rows:**

  The output shows there is no duplicated rows in each of the three data frames.

```
#check duplicated rows
print(userFeatures.duplicated().sum()) #
print(proFeatures.duplicated().sum()) #
print(click.duplicated().sum()) # output
```

```
0
0
0
```

- **Check missing value:**

  uerFeatures: 500 missing values in the column number_of_clicks_before

  proFeatures: no missing values

  click: no missing values

```
# check missing value

print("Missing value for userFeatures: \n", userFeatures.isnull().sum())
# the variable number_of_clicks_before has 500 missing values

print("\n Missing value for proFeatures: \n", proFeatures.isnull().sum().values.sum())

print("\n Missing value for clicks: \n", click.isnull().sum().values.sum()) # no missin
```

```
Missing value for userFeatures: 
 user_id                    0
number_of_clicks_before    500
ordered_before             0
personal_interests         0
dtype: int64

 Missing value for proFeatures: 
 0

 Missing value for clicks: 
 0
```

- **Handling missing values**

  userFeatures: I checked the rows with missing values, it looks the missing values are randomly distributed and it is hard to tell what reason caused the missing values. And then I checked the distribution of number_of_clicks_before, and found the number_of_clicks_before=2 is the most frequent number. Therefore, I decided to fill the missing value with 2.

```
#check the distribution of the number_of_clicks_before

print(userFeatures.groupby(["number_of_clicks_before"]).count()) # the 
#so I decide to replace the missing value with 2

userFeatures.fillna(2,inplace=True) # fill the missing value with 2
```

```
                         user_id  ordered_before  personal_interests
number_of_clicks_before
0                           2196            2196                2196
1                           2399            2399                2399
2                           2710            2710                2710
3                             59              59                  59
4                            711             711                 711
5                           1057            1057                1057
6+                          2368            2368                2368
```

```
#check if there is still missing value
userFeatures.isnull().sum() # ouput is 0, no missing value
```

```
user_id                    0
number_of_clicks_before    0
ordered_before             0
personal_interests         0
dtype: int64
```

- **Separate cells where there are multiple values, get_dummies to convert them to numeric variables and check missing value:**

  userFatures: I found the column personal_interest has many values in a cell and this information seems important since it includes the interest of the users and it might be helpful in predicting if some products will be clicked or not. Therefore, I decided to separate the values and get dummies to convert them into numeric variables.

  Drop the original personal_interest column, since the dummies has replaced its information.

  Again, check the missing values, and I found there are 1096 missing values in those interest variables. I think these missing values are from those cells where the user does not have that interest, so I fill these missing values with 0.

```python
# separate personal_interests variable
import ast

userFeatures.personal_interests = userFeatures.personal_interests.apply(lambda x: ast.literal_eval(x))
```

```python
# convert it to numerical variable using get_dummies
dumPerInt = pd.get_dummies(userFeatures.personal_interests.apply(pd.Series).stack()).sum(level=0)

dumPerInt.head(3)
```

	body	foot	fragrance	hair	hand	makeup	men_fragrance	men_skincare	nail	skincare	tools
0	1	1	1	1	1	1	1	0	1	0	1
1	1	0	0	0	0	1	1	1	1	1	1
2	0	1	0	0	0	1	0	0	1	0	1

```python
# Concatenate the original dataframe with the dummy dataframe
userFeatures1 = pd.concat([userFeatures, dumPerInt], axis=1)

userFeatures1.head(3)
```

	user_id	number_of_clicks_before	ordered_before	personal_interests	body	foot	fragrance	hair	hand	makeup	men_fragrance	men_skincare	nail	skinc
0	104939	2	True	[body, makeup, nail, hand, foot, men_fragrance...	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	
1	101562	2	True	[men_skincare, men_fragrance,	1.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	

```python
#drop the personal_interests column since the dummy vairables has
userFeatures1.drop(["personal_interests"], axis=1, inplace=True)

# check if there is new missing values appeared
userFeatures1.isnull().sum() # all the dummies has 1096 missing
```

```
user_id                       0
number_of_clicks_before       0
ordered_before                0
body                       1096
foot                       1096
fragrance                  1096
hair                       1096
hand                       1096
makeup                     1096
men_fragrance              1096
men_skincare               1096
nail                       1096
skincare                   1096
tools                      1096
dtype: int64
```

```python
# fill the NA with 0 since the NAs ar

userFeatures1.fillna(0,inplace=True)

# again check the missing values
userFeatures1.isnull().sum()
```

```
user_id                    0
number_of_clicks_before    0
ordered_before             0
body                       0
foot                       0
fragrance                  0
hair                       0
hand                       0
makeup                     0
men_fragrance              0
men_skincare               0
nail                       0
skincare                   0
tools                      0
dtype: int64
```

- **Merge data frames:**

  Merge **userFeatures** with **click** on "user_id" and named as df1, and then merge the new dataframe df1 with **proFeatures** on "product_id" named as df2. Check the missing values and shape. Now df2 has no missing values and has 35990 rows and 20 columns.

```python
# merge the data frame
df1=pd.merge(userFeatures1,click,on="user_id")
df1.head()
df2=pd.merge(df1,proFeatures,on="product_id")
df2.isnull().sum() # there is no missing value
df2.shape
```

```
(35990, 20)
```

- **Encoding the category variable into numeric variables:**

  Encoding columns "ordered_before, category, clicked, on_sale, number_of_clicks_before" into numeric variables.

```python
# encoding the category variables into numbers

df2.category.unique() # check the category variable

df2.category=df2.category.astype("category").cat.codes #encoding the category variable

# Encoding the other categorical variables
df2.ordered_before=df2.ordered_before.astype("category").cat.codes #encoding ordered_before

df2.clicked=df2.clicked.astype("category").cat.codes #encoding_clicked

df2.on_sale=df2.on_sale.astype("category").cat.codes # encoding on_sale

df2.number_of_clicks_before=df2.number_of_clicks_before.astype("category").cat.codes #encoding number_of_clicks before

df2.head()
```

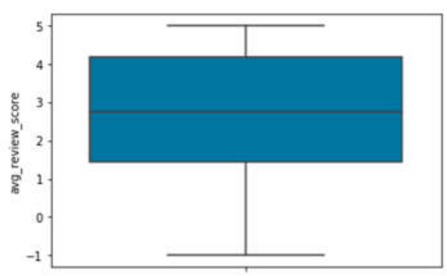
	user_id	number_of_clicks_before	ordered_before	body	foot	fragrance	hair	hand	makeup	men_fragrance	men_skincare	nail	skincare	tools	pı
0	104939	3	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	
1	101992	2	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

- **Check outliers:**

  Based on the previous understanding of the dataframes. The variables **avg_review_score** and **number_of reviews** are informative numerical variables.

```python
# make plot to check the outliers for avg_review_score
sns.boxplot(y=df2.avg_review_score) # from the boxplot, i
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x153bf1b4b70>
```



Therefore, I only checked the outlier of those two variables. For **avg_review_score**, from the boxplot, there is no outliers if the outliers are defined as more than 1.5IQR and lesson then -1.5IQR.

Also, I write a function to check the outliers based on ±1.5IQR, and found there is indeed no outliers in avg_review_score.

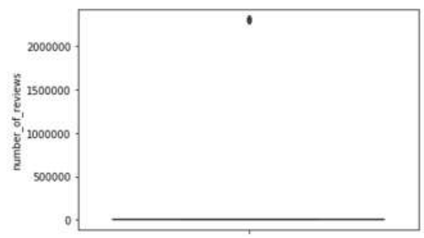
```python
# define a function to check outliers

def outliers_IQR(Var):
    Q1, Q3 = np.percentile(Var, [25, 75]) # get Q1 and Q3 of a
    IQR = Q3-Q1    # caculate the IQR
    lower_bound=Q1-(IQR*1.5)  # define the lower threshold of c
    upper_bound=Q3 + (IQR*1.5) # define the upper threshold of
    return np.where((Var > upper_bound) | (Var < lower_bound))

print(outliers_IQR(df2.avg_review_score)) # call the function a

# output is empty array, indicating no outliers in this variabl
```

```
(array([], dtype=int64),)
```

For **number_of_reviews**, from the boxplot and bar plot, there is a group of extremely big values. Also, I used the function to check the outliers and found there are 1770 outliers.

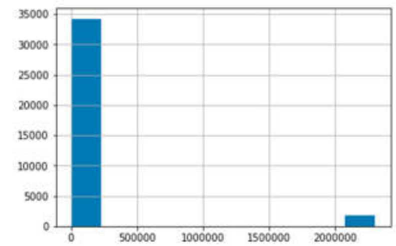
```
# visulaize the distribution of number_of_reviews
sns.boxplot(y=df2.number_of_reviews) #it seems there is
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x153bf14e160>
```




```
# check the histogram
df2.number_of_reviews.hist() # the number_of_reviews dist
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x153bf0d2160>
```




```
# check the outliers using the function
review_Out=outliers_IQR(df2.number_of_reviews)
print(review_Out) # display the index of outliers
print(len(review_Out[0])) # display number of outliers 1770 outliers
```

```
(array([   413,    414,    415, ..., 35859, 35860, 35861], dtype=int64),)
1770
```

- **Handing outliers**

I checked the distribution of outliers grouped by the "clicked" variable and found most (1126) of the outliers belong to the group clicked=0. According to a discussion with my previous applied statistics professor, if the outliers belong to a certain group, their existing is reasonable and it will not influence the result. Here most of the outliers belong to a group, I think it will not influence the result much, but I am still curious how much it will influence the result.

2.6.1 keep the outliers---df2

```
df3=df2.loc[review_Out[0],:] # extracting the rows with outliers from the dataframe

df3.groupby("clicked").count() # it seems the most of outliers are from product have been clicked, so it mostl
#belongs to one group, I think maybe it will not influence the result much, I decide to keep the outliers
```

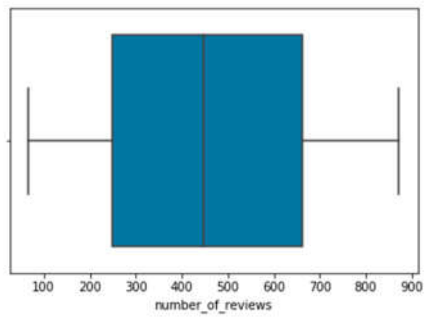
clicked	user_id	number_of_clicks_before	ordered_before	body	foot	fragrance	hair	hand	makeup	men_fragrance	men_skincare
0	1126	1126	1126	1126	1126	1126	1126	1126	1126	1126	1126
1	644	644	644	644	644	644	644	644	644	644	644

Therefore, I decide to **keep the outliers in df2**, and **make a copy of df2, which is df2a**. In **df2a, I removed the outliers** in order to compare the result from the two data frames in the further steps.

2.6.2 Remove the outliers---df2a

```
# removing outliers and save it to another dataframe to com
df2a=df2.copy() # copy the merged dataset
df2a.drop(df2a.index[review_Out[0]],axis=0,inplace=True) #
sns.boxplot(df2a.number_of_reviews) #no more outliers anymc
print(df2a.shape)
```

```
(34220, 20)
```



Question 3: Model building and evaluating

- **Import modules and split the data**

  Drop the variables "user_id, product_id, click" from df2 and df2a respectively, and take remaining data as features. Select the column "clicked" as target.

  Split the features and target into training and testing data at the ratio of 0.7 to 0.3 from df2 and df2a respectively.

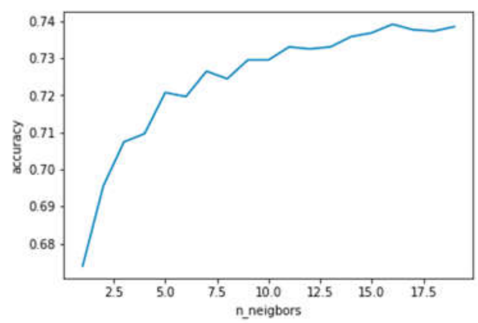
```
from sklearn.model_selection import train_test_split # import split module

#split the dataframe into features and target

X=df2.drop(columns=["user_id","product_id","clicked"],axis=1) # drop the three columns and set
Y=df2.loc[:,"clicked"]   # set clicked column as target

#do the samething as above using dataset without outliers
X1=df2a.drop(columns=["user_id","product_id","clicked"],axis=1)
Y1=df2a.loc[:,"clicked"]

# split the data with 0.7/0.3 for training and testing
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,random_state=0,train_size=0.7)

X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1,random_state=0,train_size=0.7)
```

- **KNN (KNeighborsClassifier)**

  Fit the KNeighborsClassifier model with training data and use the model to predict the test data.

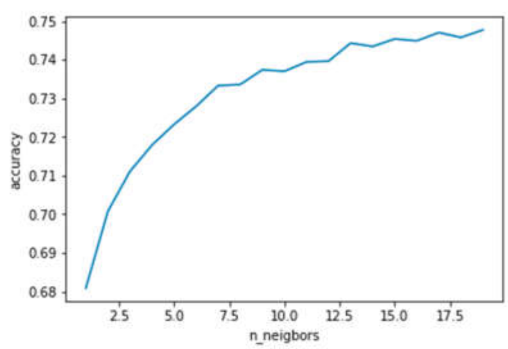
  Check the accuracy with different number of n_neighbors, k and choose the model with best accuracy to check the result.

For df2 (keep the outliers), it has the highest accuracy of 0.7391 when k=16. The precision and recall for the class 0 are 0.77 and 0.85 respectively. Precision and recall are also important for making a business decision since accuracy is sometimes miss-leading and decision should be made based on the consequence of the predicted result.




```
accuracy:
 0.7391868111512457
confusion_matrix:
 [[5853 1061]
 [1755 2128]]
classification_report:
              precision    recall  f1-score   support

           0       0.77      0.85      0.81      6914
           1       0.67      0.55      0.60      3883

   micro avg       0.74      0.74      0.74     10797
   macro avg       0.72      0.70      0.70     10797
weighted avg       0.73      0.74      0.73     10797
```

For df2a (removing the outliers), it has the highest accuracy of 0.7477 when k=19. The precision of class 0 is 0.79 and 0.83 respectively, the precision is a little bit better than data from df2, but recall is lower.



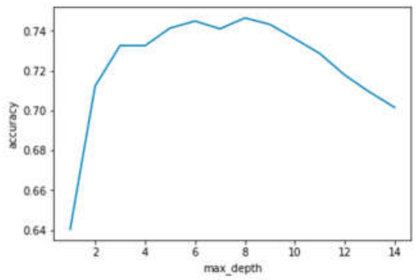

```
accuracy_score:
 0.7477108903175531
confusion_matrix:
 [[5513 1123]
 [1467 2163]]
classification_repor:
              precision    recall  f1-score   support

           0       0.79      0.83      0.81      6636
           1       0.66      0.60      0.63      3630

   micro avg       0.75      0.75      0.75     10266
   macro avg       0.72      0.71      0.72     10266
weighted avg       0.74      0.75      0.74     10266
```

- **Decision Tree**

Fit the decision tree model with training data and use the model to predict the test data.

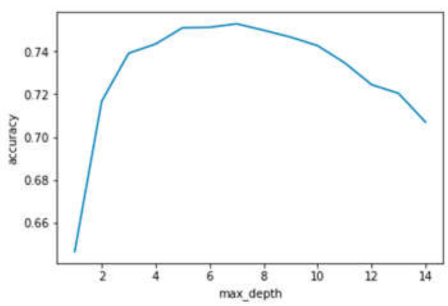
Check the accuracy with different number of max_depth, k2 and choose the model with best accuracy to check the result.

For df2 (keep the outliers), it has the highest accuracy of 0.7464 when k2=8. The accuracy is better than KNeighborsClassifier model. But the precision is similar and recall is lower than KNeighborsClassifier model for df2.




```
accuracy_score:
 0.7464110401037325
confusion_matrix:
 [[5769 1145]
 [1593 2290]]
classification_report:
              precision    recall  f1-score   support

           0       0.78      0.83      0.81      6914
           1       0.67      0.59      0.63      3883

   micro avg       0.75      0.75      0.75     10797
   macro avg       0.73      0.71      0.72     10797
weighted avg       0.74      0.75      0.74     10797
```

For df2a (removing the outliers), it has the highest accuracy of 0.7530 when k2=7.The accuracy is better than data from df2. Also, the accuracy is higher than df2a fitted with KNeighborsClassifier model. Precision is the highest compared to the previous models. and recall is close to other models.




```
accuracy_score:
 0.7529709721410481
confusion_matrix:
 [[5517 1119]
 [1417 2213]]
classification_report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.81      6636
           1       0.66      0.61      0.64      3630

   micro avg       0.75      0.75      0.75     10266
   macro avg       0.73      0.72      0.72     10266
weighted avg       0.75      0.75      0.75     10266
```

- **Random Forest**

  Fit the random forest model (n_estimator=30) with training data and use the model to predict the test data.

  For df2 (keep the outliers), it has the highest accuracy of 0.7316. This model has lower accuracy then model fitted by KNeighborsClassifier and decision tree. The precision is also lower, but it has the high recall, which is also important for making business decisions.

```
accuracy_score:
 0.7315921089191442
confusion_matrix:
 [[5842 1072]
 [1826 2057]]
classification_report:
              precision    recall  f1-score   support

           0       0.76      0.84      0.80      6914
           1       0.66      0.53      0.59      3883

   micro avg       0.73      0.73      0.73     10797
   macro avg       0.71      0.69      0.69     10797
weighted avg       0.72      0.73      0.72     10797
```

For df2a (removing the outliers), it has the highest accuracy of 0.7358. The performance is better than df2 both in accuracy and precision and recall. It has lower accuracy than the other two models but high recall.

```
accuracy_score:
 0.7358270017533606
confusion_matrix:
 [[5553 1083]
 [1629 2001]]
classification_report:
              precision    recall  f1-score   support

           0       0.77      0.84      0.80      6636
           1       0.65      0.55      0.60      3630

   micro avg       0.74      0.74      0.74     10266
   macro avg       0.71      0.69      0.70     10266
weighted avg       0.73      0.74      0.73     10266
```

Question 4: Which model has the best performance? What have you learned from the models you built?

1. **Which model has the best performance?**

- The accuracy of each model is in the table below:

	df2 (keep outliers)	df2a (removing outliers)
K Neighbors Classifier	0.7391	0.7477
Decision tree	**0.7464**	**0.7530**
Random Forest	0.7316	0.7358

- The decision tree model has the best performance for both datasets.
- Removing the outliers improves the model and get better accuracy than data with outliers.

2.What we have learned from the model:

- Data cleaning and processing is time consuming, but it is very important and worth doing, since it is obvious that outliers influence the result of the prediction.
- Keep curious about models built from data with different preprocessing help gaining insight about how to improve model and diagnose problems in the future.
- Random forest model is based on decision tree and it should have got better performance, but more time is needed to improve the model the get a better performance and this work will be done in my future study.
- Although decision tree has the highest accuracy and precision, the recall is not the highest in the model. When we choose a model in real life, we should also consider the recall since it is relevant to the consequence of the problem.