H&M Fashion Recommender Notebook

For each customer (customer_id), H&M want a prediction of up to 12 products (article_ids), which is the predicted items a customer will buy in the next 7-day period after the training time period. The file should contain a header and have the following format.

In [2]:
```
import pandas as pd
import numpy as np

#used during data exploration
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

#handling missing values where not dropped
from sklearn.impute import SimpleImputer
from sklearn import preprocessing

#ML model
from sklearn.neighbors import NearestNeighbors

# Cross-Validation (on a rolling basis)
from sklearn.model_selection import TimeSeriesSplit

# performance scoring
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score

from IPython.display import display, clear_output
```

Get the Data

In [3]:
```
#get transaction data
transactions_train_df = pd.read_csv("data/transactions_train.csv") # import the transactions dataset
```

In [4]:
```
#get product meta data
articles_df = pd.read_csv("data/articles.csv")
```

In [5]:
```
#get customer meta data
customers_df = pd.read_csv("data/customers.csv")
```

Prepare the Transaction Dataset

In [6]:
```
#first we will drop the sales channel as it will not be needed
transactions_train_df = transactions_train_df.drop(['sales_channel_id'], axis=1)
```

In [7]:
```
#we then convert our date text into a panda date type.
transactions_train_df["t_dat"] = pd.to_datetime(transactions_train_df["t_dat"])
```

In [8]:
```
#now we split out the date into seperate columns for day, month and year making use of python zip for memory efficiency
days, months, years  = zip(*[(d.day, d.month, d.year) for d in transactions_train_df['t_dat']])
transactions_train_df = transactions_train_df.assign(day=days, month=months, year=years)
```

In [9]:
```
#we drop the t_dat column as it is no longer needed
transactions_train_df = transactions_train_df.drop(['t_dat'], axis=1)
```

In [10]:
```
#we convert articles to string instead of default int.
transactions_train_df['article_id'] = transactions_train_df['article_id'].values.astype(str)
```

In [11]:
```
#we now reorganise the dataset to treat articles as the predictor
transactions_train_df = transactions_train_df[['article_id', 'year', 'month', 'day', 'price', 'customer_id']]
```

In [12]:
```
#we set articles column to index
transactions_train_df.set_index('article_id')
```

Out[12]:

article_id	year	month	day	price	customer_id
663713001	2018	9	20	0.050831	000058a12d5b43e67d225668fa1f8d618c13dc232df0ca...
541518023	2018	9	20	0.030492	000058a12d5b43e67d225668fa1f8d618c13dc232df0ca...
505221004	2018	9	20	0.015237	00007d2de826758b65a93dd24ce629ed66842531df6699...
685687003	2018	9	20	0.016932	00007d2de826758b65a93dd24ce629ed66842531df6699...
685687004	2018	9	20	0.016932	00007d2de826758b65a93dd24ce629ed66842531df6699...
...	...	...	...	...	...
929511001	2020	9	22	0.059305	fff2282977442e327b45d8c89afde25617d00124d0f999...
891322004	2020	9	22	0.042356	fff2282977442e327b45d8c89afde25617d00124d0f999...
918325001	2020	9	22	0.043203	fff380805474b287b05cb2a7507b9a013482f7dd0bce0e...
833459002	2020	9	22	0.006763	fff4d3a8b1f3b60af93e78c30a7cb4cf75edaf2590d3e5...
898573003	2020	9	22	0.033881	fffef3b6b73545df065b521e19f64bf6fe93bfd450ab20...

31788324 rows × 5 columns

In [13]:
```
# we want to see distributions and std dev
transactions_train_df.describe()
```

Out[13]:

	year	month	day	price
count	3.178832e+07	3.178832e+07	3.178832e+07	3.178832e+07
mean	2.019207e+03	6.511067e+00	1.624134e+01	2.782927e-02
std	6.644412e-01	3.273328e+00	8.934254e+00	1.918113e-02
min	2.018000e+03	1.000000e+00	1.000000e+00	1.694915e-05
25%	2.019000e+03	4.000000e+00	8.000000e+00	1.581356e-02
50%	2.019000e+03	6.000000e+00	1.700000e+01	2.540678e-02
75%	2.020000e+03	9.000000e+00	2.400000e+01	3.388136e-02
max	2.020000e+03	1.200000e+01	3.100000e+01	5.915254e-01

Prepare the Product Dataset

In [153…

```
#get product meta data
articles_df = pd.read_csv('data/articles.csv',
                          index_col=['article_id'],
                          usecols=['article_id',
                                   'product_type_no', #e.g Scarf
                                   'graphical_appearance_no', # e.g stripe
                                   'colour_group_code', #e.g white
                                   'index_group_no',#e.g ladies wear
                                   'detail_desc'],
                          dtype={'article_id': 'str'}) # drop text, no clothing size available
```

Split Training and Testing Dataset

We will create a sample from the transactions dataset to inspect relationships and test predictions. We split our sample data into a future test sample and a past training sample

In [14]:

```
aug_df = transactions_train_df.query('year == 2020 & month == 8').sample(n = 50)
```

In [15]:
```
sep_df = transactions_train_df.query('year == 2020 & month == 9').sample(n = 50)
```

In [16]:
```
sample_df = pd.concat([aug_df,sep_df])
```

```
C:\Users\newlo\AppData\Local\Temp\ipykernel_8080\2515608846.py:1: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  sample_df = aug_df.append(sep_df)
```

In [22]:
```
#we set articles column to index
sample_df.set_index('article_id')
```

Out[22]:

article_id	year	month	day	price	customer_id
781833008	2020	8	26	0.006763	80715f6e81ebc23f27308e2ddde918aee046a6e6bce7dc...
706016003	2020	8	2	0.033881	ca9cdf6db1fed23dae0b8b4951d364b1ec8635b36dd901...
505882002	2020	8	14	0.025407	f8ed98a19fbd5ef24a09c221dc8b36023a287aa64beb85...
748355002	2020	8	24	0.022017	ff2ac6b20b6e6cc42dda998278e31c5bea53de2fb64389...
736870001	2020	8	20	0.016932	ef8073b39859baafb5fe3884cc3d8f516afb73f683f6b9...
...	...	...	...	...	...
869379007	2020	9	17	0.033102	6efc93650669dc92ee31b13ba412a00dcfffbe5cf13f2c...
925512001	2020	9	4	0.067780	43787c75a09352cf10839c1fa84a14d49a2fb8da93f7d1...
921266001	2020	9	1	0.016932	7a4379a7d887099f4134e1cce73f05e30285f3cbcc90fb...
922037003	2020	9	20	0.065525	69c9024fd15a772ae51e666c02a85b53d7bc7aead6e5a2...
728703002	2020	9	9	0.042356	eeefd55a289a090d4a1f85083554ed6302b47a51f87f2e...

100 rows × 5 columns

In [132…
```
X = sample_df.values
X = X[:,3:5] # day and price
#print(X)#check before scaling
```

In [40]:
```
y = sample_df.customer_id.values # get customer ids from sample
```

In [41]:
```
tscv = TimeSeriesSplit()
print(tscv)
```

```
TimeSeriesSplit(gap=0, max_train_size=None, n_splits=5, test_size=None)
```

In [42]:
```
TimeSeriesSplit(max_train_size=None, n_splits=3)
```

Out[42]:
```
TimeSeriesSplit(gap=0, max_train_size=None, n_splits=3, test_size=None)
```

In [48]:
```
for train_index, test_index in tscv.split(X):
    #print('TRAIN:', train_index, 'TEST:', test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

In [179…
```
y_test[15]
```

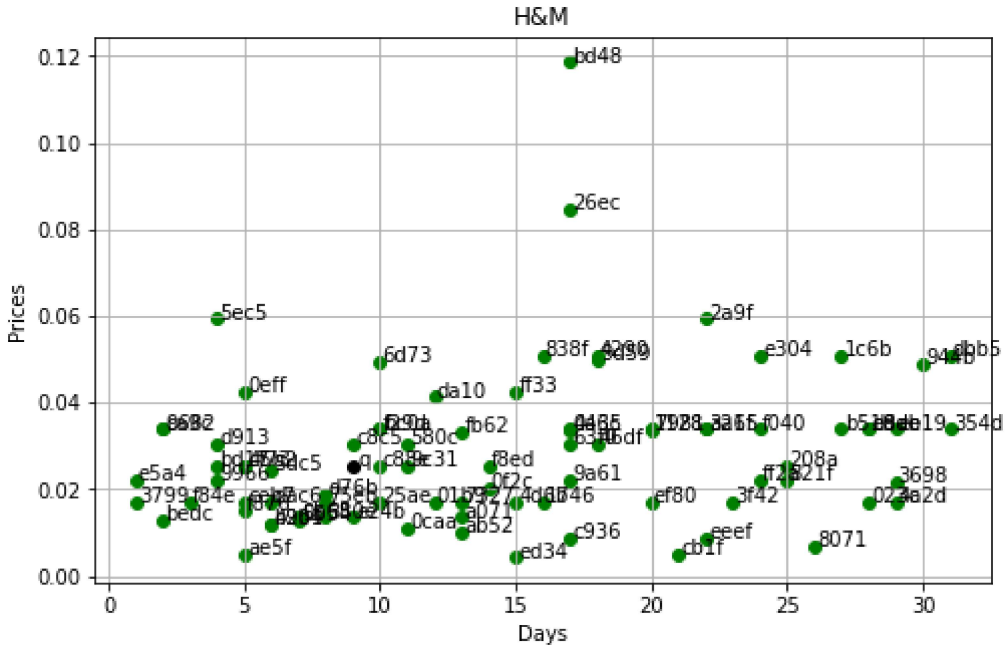
Out[179]:
```
'eeefd55a289a090d4a1f85083554ed6302b47a51f87f2e400e151dde65fc9b71'
```

In [67]:
```
q = X_test[0]# get first item in test set
```

In [68]:
```
names = y_train
```

In [108…
```
days = X_train[:,0] # x
prices = X_train[:,1] # y

q_days = q[0,0]
q_price = q[0,1]

#place day/price points on graph
plt.figure(figsize=(8,5))
plt.scatter(days, prices, color='green')

#place query on graph
plt.scatter(q_days, q_price,color='black')
plt.annotate('q',(q_days+0.2, q_price))

#label graph
plt.title("H&M")
plt.xlabel("Days")
plt.ylabel("Prices")

#display graph
```

```
plt.grid()

#populate product names on the graph
for i, txt in enumerate(names):
    plt.annotate(txt[0:4], (days[i]+0.09, prices[i]))
```



We see that the distance between our q and other products is quite near. We can determine which product is nearest using KNN.

Normalise the Sample Transaction Data

In [109…
```
#we will use N(0,1) rescale with zero mean and unit variance
scaler = preprocessing.StandardScaler().fit(X_train)  #need a handle on the scaler to apply to training and test data
X_train_scaled = scaler.transform(X_train)
```

In [110…
```
q_scaled = scaler.transform(q)
```

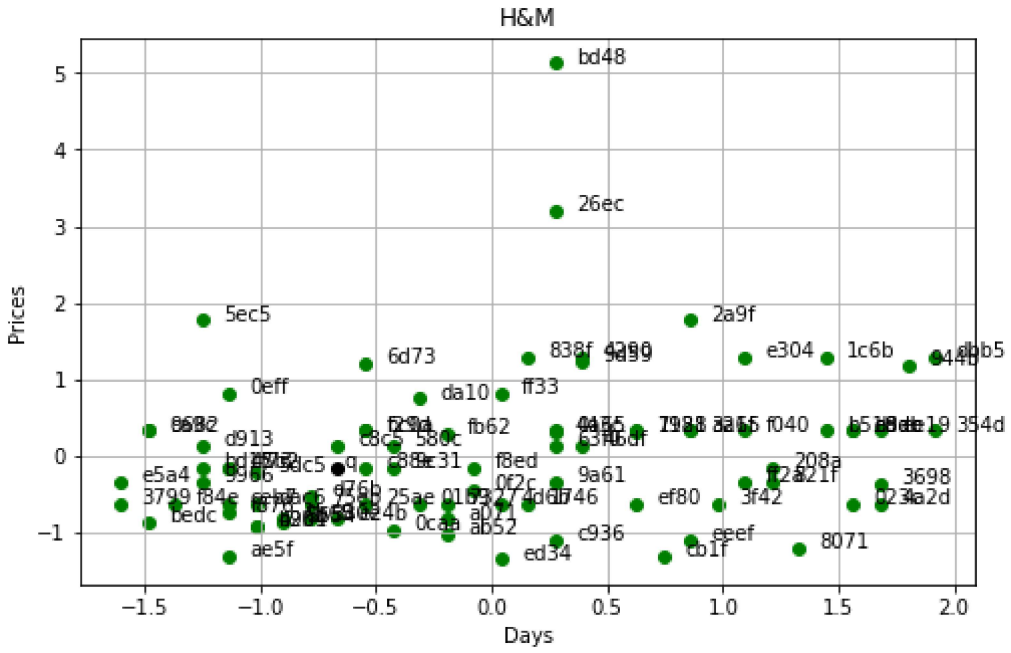
In [111…
```
X_train_scaled[0] #check after scaling
```

Out[111]:
```
array([ 1.33035828, -1.21328469])
```

In [112…
```
q_scaled[0]
```

Out[112]:
```
array([-0.66308519, -0.15573655])
```

In [113…
```
#check new scale
days = X_train_scaled[:,0] # x
prices = X_train_scaled[:,1] # y

q_days = q_scaled[0,0]
q_price = q_scaled[0,1]

plt.figure(figsize=(8,5))
plt.scatter(days, prices, color='green')

plt.scatter(q_days, q_price,color='black')
plt.annotate('q',(q_days+0.03, q_price))

plt.title("H&M")
plt.xlabel("Days")
plt.ylabel("Prices")
plt.grid()
#plt.legend(handles=[red_patch, blue_patch],loc=4)
for i, txt in enumerate(names):
    plt.annotate(txt[0:4], (days[i]+0.09, prices[i]))
```



scaling seems to have worked.

Train a KNN Model on Sample

In [135…
```
knn_model = NearestNeighbors(n_neighbors=12, radius=0.4)
knn_model.fit(X_train_scaled)
```

Out[135]:
```
NearestNeighbors(n_neighbors=12, radius=0.4)
```

Predict with KNN Model

In [136…
```
#get neighbours' names where k=12
result = knn_model.kneighbors(q_scaled, 12)[1][0]#requre 12 nearest neighbors to our query
```

In [144…
```
#result #array([43, 74, 50, 26, 57, 56, 47, 37, 77, 32, 82, 16], dtype=int64)
for n in result:
```

```
    print(n)
```

```
43
74
50
26
57
56
47
37
77
32
82
16
```

In [161…

```
# result contains the 'index' of the nearest neighbours
for n in result:
    cus = y_train[n]
    trans = sample_df.query('customer_id.str.contains("'+ cus + '")')
    p = trans.article_id.values
    p = str(p[0])
    item = articles_df.query('article_id.str.contains("'+ p + '")')
    print("PRODUCT: " + str(p))
    print(item.iloc[0][4])
    print(" ")
```

PRODUCT: 860949002
Lined bandeau bikini top with side support and cups with removable inserts that shape the bust and provide good support. Narrow, t
ie-top shoulder straps, and straps that tie at the back.

PRODUCT: 894668002
5-pocket, ankle-length trousers in twill. Loose fit with a high waist, zip fly and button and gently tapered legs.

PRODUCT: 869331006
Light-support sports bralette suitable for low-intensity workouts with a fast-drying function designed to help keep you dry and co
ol while exercising. Wide shoulder straps, a racer back and lined cups with removable inserts for an adjustable fit and to shape t
he bust. The bralette is designed with the minimum number of seams for a more comfortable fit and increased mobility.

PRODUCT: 831430003
Short top in a fine, textured knit with a deep V-neck, drawstring down the front and long trumpet sleeves.

PRODUCT: 841062003
Long-sleeved top in sturdy jersey with a print motif on the front, round neckline, ribbing at the cuffs and short slits in the sid
es.

PRODUCT: 316085002
Fitted polo-neck top in ribbed jersey with long sleeves.

PRODUCT: 866731003
Ankle-length sports tights in fast-drying functional fabric. High waist with wide, elasticated ribbing. The tights are designed wi
th the minimum number of seams for a more comfortable fit and increased mobility.

PRODUCT: 841699003
Lined, underwired bikini top with padded cups that shape the bust and provide good support. Ties at the back of the neck and an ad
justable metal fastener at the back.

PRODUCT: 818754007
Fitted top in soft, ribbed viscose jersey with a turtle neck and long sleeves.

PRODUCT: 826646009
Fitted T-shirts in soft organic cotton jersey.

PRODUCT: 852584002
High-waisted, ankle-length sports tights in fast-drying functional fabric. Wide panel to hold in and shape the waist. Concealed ke
y pocket in the waistband.

PRODUCT: 706016053
High-waisted jeans in washed superstretch denim with a zip fly and button, fake front pockets, real back pockets and super-skinny
legs.

Evaluate Model on Sample

In [205…
```
names = y_test
y_pred = []

X_test_scaled = scaler.transform(X_test)

knn_model.fit(X_test_scaled)
pred_cus = knn_model.kneighbors(X_test_scaled, len(y_test))[1][0]
#print(y_test[pred_cus])
y_pred = y_test[pred_cus]
```

In [206…
```
print(y_test)
```

```
['c49c29d37de92d310024ef68764c6343b9ad0494be093b304cc80670d26ba604'
 '50f1967ba7d01526145898baf27d87c6224eb011029c3c12a666c2cb0843e8d3'
 '13b13896ffac914b7948c5ee5c56f9c851a645ce372adec605fb1cc46e3a832b'
 '402dc14e27d02b886cd29996d8dbdf7fbeb92688d80359fcbeb731f5d13813ab'
 'd9c7a5fb3d5ae128928b24e7c14a9560cce6033b71594ae153458fa42dd0f1ee'
 '9042c8e748b09a734bf56d1db4f99a6992f1a95f3bd27cc4187a29009f6803be'
 '338d7210e2dbfd18740a71fc7eb097b95ca75b29105b857435babcbb92c14e2c'
 'ceae02ea26c97220e62e1da2c7b4dc41cc219af963e704df529c3f4e0c3126d2'
 '9fdb8e54eb76a45cd633ade7b39bc59b84737da2e01352180aed6d591bd0e6c7'
 '6f6b503111fc78b4ac81fe7e8aed0554139a0c4b47e9bf4c79b439d531d21622'
 '7985102152ffb1ce6fa0425be69a78221efc9cad44f0bc3c8652da7741cd08db'
 '6efc93650669dc92ee31b13ba412a00dcfffbe5cf13f2c8f9f4e4d40274000cb'
 '43787c75a09352cf10839c1fa84a14d49a2fb8da93f7d1c36b7320187cbba967'
 '7a4379a7d887099f4134e1cce73f05e30285f3cbcc90fbdc28b71bddaf92627e'
 '69c9024fd15a772ae51e666c02a85b53d7bc7aead6e5a257e6688a4040b202f1'
 'eeefd55a289a090d4a1f85083554ed6302b47a51f87f2e400e151dde65fc9b71']
```

In [207…
```
print(y_pred)
```

```
['c49c29d37de92d310024ef68764c6343b9ad0494be093b304cc80670d26ba604'
 '9042c8e748b09a734bf56d1db4f99a6992f1a95f3bd27cc4187a29009f6803be'
 '402dc14e27d02b886cd29996d8dbdf7fbeb92688d80359fcbeb731f5d13813ab'
 'eeefd55a289a090d4a1f85083554ed6302b47a51f87f2e400e151dde65fc9b71'
 '6f6b503111fc78b4ac81fe7e8aed0554139a0c4b47e9bf4c79b439d531d21622'
 '6efc93650669dc92ee31b13ba412a00dcfffbe5cf13f2c8f9f4e4d40274000cb'
 '7a4379a7d887099f4134e1cce73f05e30285f3cbcc90fbdc28b71bddaf92627e'
 '13b13896ffac914b7948c5ee5c56f9c851a645ce372adec605fb1cc46e3a832b'
 '9fdb8e54eb76a45cd633ade7b39bc59b84737da2e01352180aed6d591bd0e6c7'
 '338d7210e2dbfd18740a71fc7eb097b95ca75b29105b857435babcbb92c14e2c'
 '50f1967ba7d01526145898baf27d87c6224eb011029c3c12a666c2cb0843e8d3'
 '7985102152ffb1ce6fa0425be69a78221efc9cad44f0bc3c8652da7741cd08db'
 'd9c7a5fb3d5ae128928b24e7c14a9560cce6033b71594ae153458fa42dd0f1ee'
 'ceae02ea26c97220e62e1da2c7b4dc41cc219af963e704df529c3f4e0c3126d2'
 '43787c75a09352cf10839c1fa84a14d49a2fb8da93f7d1c36b7320187cbba967'
 '69c9024fd15a772ae51e666c02a85b53d7bc7aead6e5a257e6688a4040b202f1']
```

In [215…
```
precision_score(y_test, y_pred, average='micro')
```

Out[215]:
```
0.125
```

In [216…
```
accuracy_score(y_test, y_pred)
```

Out[216]:
```
0.125
```

Generate Predictions File

Prepare Customer Dataset

H&M are expecting about 1,371,980 prediction rows, we will only have 1,362,281 because 9,699 customers have not purchased anything yet.

Originally we were going to generate a list of missing customers, determine their demographics and match to those who have made transactions then use their values to fill the missing. This however was computationally expensive.

In [13]:
```
customers_df['customer_id'].size
```

Out[13]:
```
1371980
```

In [14]:
```
cus_pred_df = transactions_train_df.groupby(transactions_train_df.customer_id)[['day','price']].median()
```

In [15]:
```
cus_pred_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1362281 entries, 00000dbacae5abe5e23885899a1fa44253a17956c6d1c3d25f88aa139fdfc657 to ffffd9ac14e89946416d80e791d064701994755c3ab686a1eaf3458c36f52241
Data columns (total 2 columns):
 #   Column  Non-Null Count    Dtype  
---  ------  --------------    -----  
 0   day     1362281 non-null  float64
 1   price   1362281 non-null  float64
dtypes: float64(2)
memory usage: 31.2+ MB
```

In [16]:
```
cus_pred_df['day'].size #we set articles column to index
```

Out[16]:
```
1362281
```

In [17]:
```
cus_pred_df['customer_id'] = cus_pred_df.index
```

In [18]:
```
cus_pred_df = cus_pred_df.reset_index(drop=True)
```

In [19]:
```
cus_pred_df = cus_pred_df[['customer_id', 'day', 'price']]
```

In [20]:
```
cus_pred_df.head()
```

Out[20]:

	customer_id	day	price
0	00000dbacae5abe5e23885899a1fa44253a17956c6d1c3...	25.0	0.030492
1	0000423b00ade91418cceaf3b26c6af3dd342b51fd051e...	21.0	0.025407
2	000058a12d5b43e67d225668fa1f8d618c13dc232df0ca...	18.0	0.033881
3	00005ca1c9ed5f5146b52ac8639a40ca9d57aeff4d1bd2...	9.0	0.030492
4	00006413d8573cd20ed7128e53b7b13819fe5cfc2d801f...	9.0	0.033881

In [21]:
```
non_transaction_customers_df = customers_df[~customers_df['customer_id'].isin(cus_pred_df['customer_id'])]
```

In [22]:
```
non_transaction_customers_df['day'] = 0
```

```
C:\Users\newlo\AppData\Local\Temp\ipykernel_7836\3592874415.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  non_transaction_customers_df['day'] = 0
```

In [23]:
```
non_transaction_customers_df['price'] = 0.0
```

```
C:\Users\newlo\AppData\Local\Temp\ipykernel_7836\2901366215.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  non_transaction_customers_df['price'] = 0.0
```

In [24]:
```
non_transaction_customers_df = non_transaction_customers_df.drop(['FN', 'Active', 'club_member_status', 'fashion_news_frequency', '

#we now reorganise the dataset to treat articles as the predictor
non_transaction_customers_df = non_transaction_customers_df[['customer_id', 'day', 'price']]
```

In [25]:
```
non_transaction_customers_df['customer_id'].size
```

Out[25]:
```
9699
```

In [26]:
```
non_transaction_customers_df = non_transaction_customers_df.reset_index(drop=True)
```

In [27]:
```
non_transaction_customers_df.head()
```

Out[27]:

	customer_id	day	price
0	00058ecf091cea1bba9d800cabac6ed1ae284202cdab68...	0	0.0
1	000df4d2084d142416b8165bdd249bab8fea2393447aed...	0	0.0
2	00193ff7f374dbcfecfa7fead0488e454be4918bec1ebd...	0	0.0
3	001f00e8c1eba437ff0dbad26a9a3d49e47cbf05fff02a...	0	0.0
4	002648d8f3b288531b24860f4a68a31d029ec5a0495c04...	0	0.0

In [28]:
```
cus_pred_df = cus_pred_df.append(non_transaction_customers_df)
```

```
C:\Users\newlo\AppData\Local\Temp\ipykernel_7836\935700407.py:1: FutureWarning: The frame.append method is deprecated and will be
removed from pandas in a future version. Use pandas.concat instead.
  cus_pred_df = cus_pred_df.append(non_transaction_customers_df)
```

In [29]:
```
cus_pred_df.tail()
```

Out[29]:

	customer_id	day	price
9694	ffe5801cb2a5b51d4d068322d7f8082e995f427a6f22a6...	0.0	0.0
9695	ffeb3ca867aba57a312fe9d28d67dd46ef2240fe92a94c...	0.0	0.0
9696	fff456fa60aac9174456c2f36ede5e0f25429a16c88a34...	0.0	0.0
9697	fffa8d3cea26d4f5186472b923629b35fa28051f258030...	0.0	0.0
9698	ffff01710b4f0d558ff62d7dc00f0641065b37e840bb43...	0.0	0.0

In [30]:
```
#we set articles column to index
cus_pred_df.set_index('customer_id')
```

Out[30]:

customer_id	day	price
00000dbacae5abe5e23885899a1fa44253a17956c6d1c3d25f88aa139fdfc657	25.0	0.030492
0000423b00ade91418cceaf3b26c6af3dd342b51fd051eec9c12fb36984420fa	21.0	0.025407
000058a12d5b43e67d225668fa1f8d618c13dc232df0cad8ffe7ad4a1091e318	18.0	0.033881
00005ca1c9ed5f5146b52ac8639a40ca9d57aeff4d1bd2c5feb1ca5dff07c43e	9.0	0.030492
00006413d8573cd20ed7128e53b7b13819fe5cfc2d801fe7fc0f26dd8d65a85a	9.0	0.033881
...	...	...
ffe5801cb2a5b51d4d068322d7f8082e995f427a6f22a6f471e773a29a27a3d0	0.0	0.000000
ffeb3ca867aba57a312fe9d28d67dd46ef2240fe92a94c434a0debd1902c386a	0.0	0.000000
fff456fa60aac9174456c2f36ede5e0f25429a16c88a346dfb99333eb604cd6f	0.0	0.000000
fffa8d3cea26d4f5186472b923629b35fa28051f25803084f9fb0b0b7b3fc0eb	0.0	0.000000
ffff01710b4f0d558ff62d7dc00f0641065b37e840bb43bd852afccb08f2bbd2	0.0	0.000000

1371980 rows × 2 columns

Check Missing Values

In [31]:
```
missing = cus_pred_df.isin([0]).sum(axis=1)
print(cus_pred_df.shape)
print(missing)
```

```
(1371980, 3)
0       0
1       0
2       0
3       0
4       0
       ..
9694    2
9695    2
9696    2
9697    2
9698    2
Length: 1371980, dtype: int64
```

Fix Missing / Zero Values

#replace minus sign in text and check result of dataset after imputing missing values #cus_pred_df.columns = cus_pred_df.columns.str.replace('-', '') cus_pred_df.head()

In [32]:
```
cus_pred_df['day'] = cus_pred_df['day'].replace(0, cus_pred_df['day'].median())
```

In [33]:
```
cus_pred_df['price'] = cus_pred_df['price'].replace(0, cus_pred_df['price'].median())
```

In [34]:
```
missing = cus_pred_df.isin([0]).sum(axis=1)
print(cus_pred_df.shape)
print(missing)
```

```
(1371980, 3)
0       0
1       0
2       0
3       0
4       0
       ..
9694    0
9695    0
9696    0
9697    0
9698    0
Length: 1371980, dtype: int64
```

In [35]:
```
cus_pred_df.tail()
```

Out[35]:

	customer_id	day	price
9694	ffe5801cb2a5b51d4d068322d7f8082e995f427a6f22a6...	17.0	0.025407
9695	ffeb3ca867aba57a312fe9d28d67dd46ef2240fe92a94c...	17.0	0.025407
9696	fff456fa60aac9174456c2f36ede5e0f25429a16c88a34...	17.0	0.025407
9697	fffa8d3cea26d4f5186472b923629b35fa28051f258030...	17.0	0.025407
9698	ffff01710b4f0d558ff62d7dc00f0641065b37e840bb43...	17.0	0.025407

File generation function

In [63]:

```
#H&M Collaborative KNN Model Based Recommendation System
def generate_prediction_file(r_model, cus_df, write_file):
    #get values
    X = cus_df.values

    X = X[:,1:2] # day and price
    y = X[:,0]

    names = y

    #write_file = "ros_predictions.csv"
    with open(write_file, "wt", encoding="utf-8") as output:
        #add headers first
        output.write("customer_id, prediction" + '\n')

        #now we loop through each row and write predictions to csv file
        for index, cus in cus_df.iterrows():
            #select day and price and convert them to np array
            q_cus = np.array([cus['day'],cus['price']], dtype=float) #cus[3:5]

            #normalise data
            q_cus_scaled = scaler.transform([q_cus])

            #get neighbours' names where k=12
            result = r_model.kneighbors(q_cus_scaled, 12)[1][0]

            #create prediction csv file
            r = []
```

```
            r.append(cus.customer_id + ",")
            for n in result:
                cus = names[n]
                trans = transactions_train_df.query('customer_id.str.contains("'+ cus + '")')
                p = trans.article_id.values
                r.append("0" + str(p[0]))
                prediction =  ' '.join(r)
            #write predictions to csv file
            output.write(prediction + '\n')
            clear_output(wait=True)
            display('Processed Row: ' + str(index))
```

Generate the file

In [64]:
```
#we now generate our intial predictions list and save it as a csv file
generate_prediction_file(knn_model, cus_pred_df, "data/ros_predictions5.csv")
```

```
'Processed Row: 9698'
```

Check submission file

In [66]:
```
#inspect our prediction data
predictions_df = pd.read_csv("data/ros_predictions3.csv")
```

In [67]:
```
predictions_df.head()
```

Out[67]:

	customer_id	prediction
0	00000dbacae5abe5e23885899a1fa44253a17956c6d1c3...	0700701005 0715624001 0754238015 0811925001 0...
1	0000423b00ade91418cceaf3b26c6af3dd342b51fd051e...	0811925001 0848988001 0845790006 0700701005 0...
2	000058a12d5b43e67d225668fa1f8d618c13dc232df0ca...	0845790006 0911108003 0694848003 0848988001 0...
3	00005ca1c9ed5f5146b52ac8639a40ca9d57aeff4d1bd2...	0694848003 0873884005 0911108003 0814762002 0...
4	00006413d8573cd20ed7128e53b7b13819fe5cfc2d801f...	0873884005 0694848003 0814762002 0911108003 0...

In [68]:
```
predictions_df['customer_id'].size
```

Out[68]:
```
1371980
```