Requirement

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 [92]:
```
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
from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score

from IPython.display import display, clear_output
```

Get the Data

In [2]:
```
#get transaction data
transactions_train_df = pd.read_csv("data/transactions_train.csv") # import the transactions dataset
```

In [3]:
```
#get product meta data
articles_df = pd.read_csv("data/articles.csv")
```

In [4]:
```
#get customer meta data
customers_df = pd.read_csv("data/customers.csv")
```

Prepare the Transaction Dataset

In [5]:
```
#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 [6]:
```
#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 [7]:
```
#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 [8]:
```
#we drop the t_dat column as it is no longer needed
transactions_train_df = transactions_train_df.drop(['t_dat'], axis=1)
```

In [9]:
```
#we convert articles to string instead of default int.
transactions_train_df['article_id'] = transactions_train_df['article_id'].values.astype(str)
```

In [10]:
```
#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 [11]:
```
#we set articles column to index
transactions_train_df.set_index('article_id')
```

Out[11]:

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 [12]:
```
# we want to see distributions and std dev
transactions_train_df.describe()
```

Out[12]:

	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 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

Explore Sample Transaction Data

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 [36]:
```
ts_train_df = transactions_train_df.query('year == 2020 & month == 8').sample(n = 30)
```

In [37]:
```
ts_test_df = transactions_train_df.query('year == 2020 & month == 9').sample(n = 30)
```

In [38]:
```
ts_train_df['article_id'].size
```

Out[38]:
```
30
```

In [39]:
```
ts_train_df.head()
```

Out[39]:

	article_id	year	month	day	price	customer_id
30027601	833499006	2020	8	6	0.013542	c4d15df4098649aef574fbb497993bf2d314fcadc2f3ef...
29923312	814762002	2020	8	4	0.033881	ce5405450c1dacab4e312c341af772f69cb2f43640cb5d...
29870001	832331003	2020	8	3	0.016932	cc2702999be0bcde285174442522195670d3e81d889e87...
29790145	921090001	2020	8	1	0.024441	e4bd1b7f9848a2bc163e47015a92baca3bb79218c30b4a...
30426490	845790006	2020	8	17	0.033881	2acf82660c5dfc20a4e59fe43a31adce428eda07bbbfac...

In [40]:
```
#customer id too long for graph (000058a12d5b43e67d225668fa1f8d618c13dc232df0cad8ffe7ad4a1091e318)
#we will use articles

#we set articles column to index
ts_train_df.set_index('article_id')
```

Out[40]:

article_id	year	month	day	price	customer_id
833499006	2020	8	6	0.013542	c4d15df4098649aef574fbb497993bf2d314fcadc2f3ef...
814762002	2020	8	4	0.033881	ce5405450c1dacab4e312c341af772f69cb2f43640cb5d...
832331003	2020	8	3	0.016932	cc2702999be0bcde285174442522195670d3e81d889e87...
921090001	2020	8	1	0.024441	e4bd1b7f9848a2bc163e47015a92baca3bb79218c30b4a...
845790006	2020	8	17	0.033881	2acf82660c5dfc20a4e59fe43a31adce428eda07bbbfac...
875073003	2020	8	30	0.022017	959c7e95a078c1036975792388c7ec17fbd74542e67060...
920283001	2020	8	25	0.042356	c5b41defd47a2fd9c27dae04c3282296f9ffd8e7de7a87...
811925001	2020	8	22	0.021475	890d56fbe9fd824801beae84441980a680c9080480dbe3...
909921002	2020	8	27	0.013542	a0d640ff7ac7d241b28f79cfe1ba690c21d9a2eeaaffd5...
863937007	2020	8	2	0.011847	1f6c9f64c6f58550748f361e93bca0c202a511f60ed8c0...
767377003	2020	8	2	0.003373	9cb6f81582c1d15f088820968e1fd471f49ee7000eb964...
816423001	2020	8	25	0.013542	f57a3de3dbcfc1f4026f06c36b5e1d9802a85f3503a59b...
782689001	2020	8	19	0.013542	d65fe26fefb1cbce5d559c546b0e5cebeca30e2e1871e1...
694848003	2020	8	13	0.030492	0e91fe351e35b58a23faf8a9e1d486e9c254c044484bce...
903004003	2020	8	30	0.055356	604a3e5bdc04d072e3bf4ad9a02e2673ba18dd843a4f84...
873884005	2020	8	5	0.033881	da1c11cea91fccae9c4e3bcdaac2366fbf53294f6e6d64...
879956002	2020	8	20	0.013542	046769d3e458860f318e6bf464341e1737740c18e1f151...
848988001	2020	8	14	0.025407	a2529c9d41ae638d93bd3209ff9b58f0a13bdc36cbf13c...
733067001	2020	8	27	0.016932	f089442109721194d64fc6efd50b2073c7f02ec0205cdd...
877131002	2020	8	2	0.019051	c8a0c28ca17b1c4648e0505de97a009153af5f83a56b9a...
715624001	2020	8	30	0.025407	a907aa7c2b5f9699f3caf45f2c20db86155f1976c98119...
851606002	2020	8	11	0.016932	b722dcd0e24a517adbfadf763a379dee057ae9f305b19f...
880099003	2020	8	4	0.016932	3e29ef505bef10f1eac503e71103094fb149164b567483...

article_id	year	month	day	price	customer_id
921748002	2020	8	1	0.059305	8c51df9d15c532cf3ca5751ac27edf4147f242982e2be6...
754238015	2020	8	31	0.033881	9baed0b60a048e833a311dd2f6b747899315c1b4ce4469...
700701005	2020	8	30	0.025407	c24d5db1d92f8134e8c9404332c8b2852c064617efb12b...
841383002	2020	8	5	0.008458	96c8d1119a55a29a46f7793ce035c20cf693fac9dda1a5...
911108003	2020	8	14	0.032881	3b7df01c004f9de0d9c3dbcae0f144a46dad9ce76c683e...
875628001	2020	8	3	0.016932	78372aeca912e989c19117ed7e6e6adc4711501c366389...
811899001	2020	8	10	0.042356	e93f20b0b25f507b304c99d2c61c2d5449cb99a539c53b...

In [41]:
```
ts_test_df.set_index('article_id')
```

Out[41]:

article_id	year	month	day	price	customer_id
708138021	2020	9	15	0.065559	b97ea6d41e9aaf2ee485ef163b28681e467dd5bf87d34b...
228257001	2020	9	22	0.004508	0844decc64e0445cdcbc48c2888b38e422c993f7c3250e...
873279004	2020	9	10	0.042356	22602fa3a7357d992f3ff7c6676605162106ff8182b8c5...
803969003	2020	9	20	0.025407	133e52a2466baf3b679fa90a63300d28b2d64713f11fe5...
909370001	2020	9	16	0.033881	a05916b39da1d4e2116a43cfa8e811f4f0a62308eb71f6...
610776083	2020	9	8	0.007610	aeb229826601b7bee6d8a4d2418efa2058c436f2273d04...
624251002	2020	9	4	0.033881	06074be8d839464bbb7d6a89e2470a2aa1f5cee45fa123...
579541089	2020	9	19	0.016932	4023ccabeb706b7558d5d725b134a678dab0e63604b473...
903773005	2020	9	17	0.016932	0453a09d3c90adf0fe0d31e360d3d487acdf80b1f662b3...
851010009	2020	9	7	0.016932	f682c73f9af721204adf25b979c45543431017dda23751...
867205001	2020	9	10	0.054356	5418bb4d05757aa68e93a32c5bd1ecb61fa7144862c8a1...
908081003	2020	9	5	0.042356	91a3184d1971cba2238364c0a7fa97a2765ca54145ebac...
814762002	2020	9	4	0.030000	ee67f3d8ff18dd0fa5697be6f4352b53610e2ae67923bd...
863583001	2020	9	4	0.033881	42d04bf10e843102eb876a45dd75a6b3f1a2765e0983bb...
851374001	2020	9	14	0.022017	f87d909986832d2581d3ff42fdaae60647e6c84e8c8bc8...
862482007	2020	9	3	0.013542	107f8da9f536304de197b105209242ec38aba00459fdc2...
762846027	2020	9	7	0.025407	eb97644bab5e598142e81d51b7784a3504e3bcb9c5ab57...
357751001	2020	9	15	0.010153	b139a0f0df7135f99ec471c5caefc2b2e66a24b9de4e95...
914453002	2020	9	22	0.042356	8409e11b81547c4ff6b132bda13d096282287f69a3596b...
835704008	2020	9	22	0.025407	53a9390aa8fd1f8c69ac7420af01a4636868d4735057cb...
919365008	2020	9	3	0.042356	1efc82802a3728cdde36265831a3dec2cb7b6c9c795ead...
568601006	2020	9	12	0.050831	37aa13e381b05b017cecc6c93f63ceae7f690bc1df2bdb...
816423003	2020	9	14	0.012085	9f190070a997dec0090de28df5f13f85b95187f7c51796...

article_id	year	month	day	price	customer_id
589222005	2020	9	18	0.010153	5e10bb3bb69b77ec405ba7c5dd5a300bf90a9b7267957c...
875072004	2020	9	4	0.025407	339a57b4c8993a9ec621bb4f95211575c5f95895ab5314...
915366001	2020	9	2	0.033881	16cdc0df29a737dd9f235ccf4372f2e3619e6679a4bcbd...
679853001	2020	9	12	0.016932	e1a01c649187dca282555b643fe4fab14220becbe2543a...
801673014	2020	9	13	0.016932	fe4b973b5606e783fccde6e25317a30b1a91def8de5cc2...
824995009	2020	9	4	0.025407	991781df31fb1ee410dce637138110527f2875621a6393...
716947009	2020	9	11	0.008458	a373afe701611ba1bf44ac2dc629cc3a5fd3d11a3f8be5...

In [42]:
```
test_query = ts_test_df.iloc[:1]# get first item in test set
test_query.set_index('article_id')#set index to products
```

Out[42]:

article_id	year	month	day	price	customer_id
708138021	2020	9	15	0.065559	b97ea6d41e9aaf2ee485ef163b28681e467dd5bf87d34b...

In [43]:
```
test_query['customer_id'].values in ts_train_df['customer_id'].values # make sure test is not in training
```

Out[43]:
```
False
```

In [44]:
```
# Store features as numpy array Xy
#names = ts_train_df.index
names = ts_train_df['article_id']

X = ts_train_df.values

X[1] # look at one of our feature values
```

Out[44]:
```
array(['814762002', 2020, 8, 4, 0.0338813559322033,
       'ce5405450c1dacab4e312c341af772f69cb2f43640cb5dfbc02b8e3239590c6e'],
      dtype=object)
```

In [45]:
```
# get day and price of test customer
q = test_query[['day','price', 'article_id']]
q.set_index('article_id')#set product to index
```

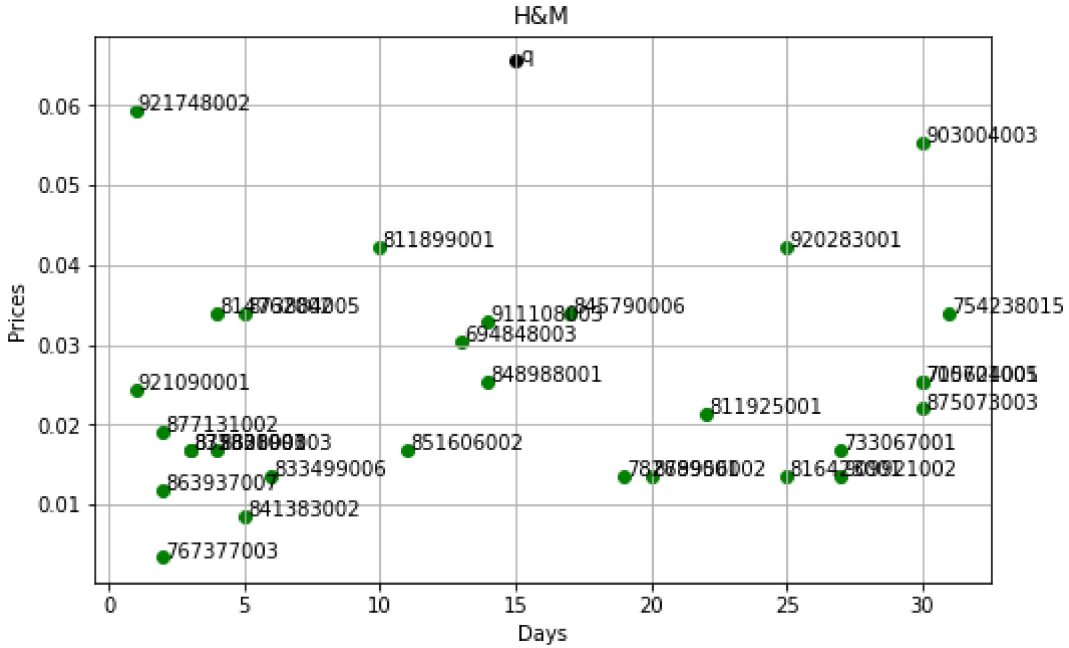
Out[45]:

article_id	day	price
708138021	15	0.065559

In [46]:
```
[q.iloc[0][0], q.iloc[0][1]]
```

Out[46]:
```
[15, 0.0655593220338983]
```

In [47]:
```
days = X[:,3] # x
prices = X[:,4] # y

q_days = q.iloc[0][0]
q_price = q.iloc[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, (days[i]+0.09, prices[i]))
```



We see that the distance between our q and other products is quite far but we can determine which product is nearest.

In [48]:
```
X[:,3:5]#check before scaling
```

Out[48]:
```
array([[6, 0.0135423728813559],
       [4, 0.0338813559322033],
       [3, 0.0169322033898305],
       [1, 0.0244406779661016],
       [17, 0.0338813559322033],
       [30, 0.0220169491525423],
       [25, 0.0423559322033898],
       [22, 0.0214745762711864],
       [27, 0.0135423728813559],
       [2, 0.0118474576271186],
       [2, 0.0033728813559322],
       [25, 0.0135423728813559],
       [19, 0.0135423728813559],
       [13, 0.0304915254237288],
       [30, 0.0553559322033898],
       [5, 0.0338813559322033],
       [20, 0.0135423728813559],
       [14, 0.0254067796610169],
       [27, 0.0169322033898305],
       [2, 0.0190508474576271],
       [30, 0.0254067796610169],
       [11, 0.0169322033898305],
       [4, 0.0169322033898305],
       [1, 0.0593050847457627],
       [31, 0.0338813559322033],
       [30, 0.0254067796610169],
       [5, 0.008457627118644],
       [14, 0.0328813559322033],
       [3, 0.0169322033898305],
       [10, 0.0423559322033898]], dtype=object)
```

Normalise the Sample Transaction Data

In [49]:
```
#we will use N(0,1) rescale with zero mean and unit variance
scaler = preprocessing.StandardScaler().fit(X[:,3:5])  #need a handle on the scaler to apply to training and test data
X_scaled = scaler.transform(X[:,3:5])
```

In [50]:
```
q_scaled = scaler.transform([[q.iloc[0][0], q.iloc[0][1]]])
```

In [51]:
```
X_scaled#check after scaling
```

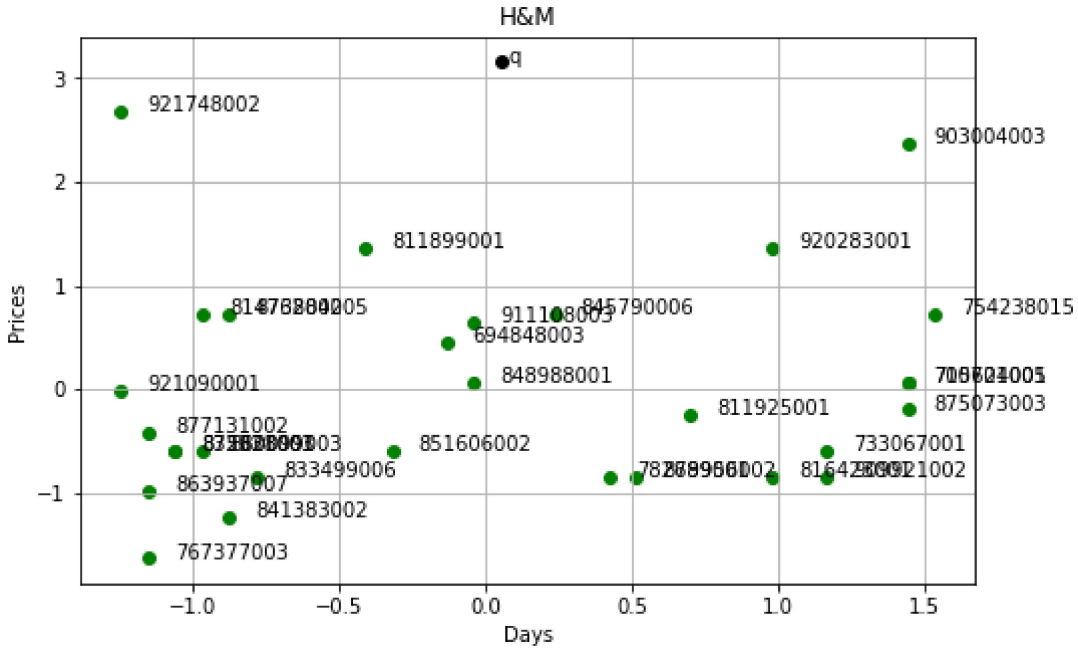
Out[51]:
```
array([[-0.78241228, -0.85045154],
       [-0.9679646 ,  0.71607811],
       [-1.06074076, -0.58936326],
       [-1.24629308, -0.01105274],
       [ 0.23812548,  0.71607811],
       [ 1.44421555, -0.19773085],
       [ 0.98033475,  1.36879879],
       [ 0.70200627, -0.23950498],
       [ 1.16588707, -0.85045154],
       [-1.15351692, -0.98099568],
       [-1.15351692, -1.63371636],
       [ 0.98033475, -0.85045154],
       [ 0.42367779, -0.85045154],
       [-0.13297916,  0.45498983],
       [ 1.44421555,  2.37007232],
       [-0.87518844,  0.71607811],
       [ 0.51645395, -0.85045154],
       [-0.040203  ,  0.06335742],
       [ 1.16588707, -0.58936326],
       [-1.15351692, -0.42618309],
       [ 1.44421555,  0.06335742],
       [-0.31853148, -0.58936326],
       [-0.9679646 , -0.58936326],
       [-1.24629308,  2.67424016],
       [ 1.53699171,  0.71607811],
       [ 1.44421555,  0.06335742],
       [-0.87518844, -1.24208395],
       [-0.040203  ,  0.63905707],
       [-1.06074076, -0.58936326],
       [-0.41130764,  1.36879879]])
```

In [52]:
```
q_scaled
```

Out[52]:
```
array([[0.05257316, 3.15594803]])
```

In [53]:
```
#check new scale
days = X_scaled[:,0] # x
prices = X_scaled[:,1] # y

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

plt.scatter(q_scaled[:,0],q_scaled[:,1],color='black')
```

```
plt.annotate('q',(q_scaled[:, 0]+0.03,q_scaled[:, 1]))

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, (days[i]+0.09, prices[i]))
```



scaling seems to be ok

Train a KNN Model on Sample

In [58]:
```
knn_model = NearestNeighbors(n_neighbors=3, radius=0.4)
knn_model.fit(X_scaled)
```

Out[58]:
```
NearestNeighbors(n_neighbors=3, radius=0.4)
```

Predict with KNN Model

In [59]:
```
#get neighbours' names where k=12
result = knn_model.kneighbors(q_scaled, 12)[1][0]#requre 12 nearest neighbors to our query
```

In [60]:
```
# result contains the 'index' of the nearest neighbours
for n in result:
    p = names.iloc[n]
    item = articles_df.query('article_id == ' + str(p))
    print("PRODUCT: " + str(p))
    print(item.iloc[0][24])
    print(" ")
```

PRODUCT: 921748002
Oversized shirt jacket in cotton corduroy with a collar, buttons down the front and a yoke at the back. Flap chest pockets with a button and diagonal welt front pockets. Dropped shoulders, long sleeves with buttoned cuffs, and short slits in the sides. Slightly longer at the back. Unlined.

PRODUCT: 903004003
Trousers in a recycled polyester weave. High waist with pleats at the front, a zip fly with a hook-and-eye fastening, side pockets, fake back pockets and wide, straight legs with creases.

PRODUCT: 811899001
Fully lined swimsuit with a low-cut back and high-cut legs. Cups with removable inserts that shape the bust and provide good support.

PRODUCT: 920283001
Jumper in a soft, fine, fluffy knit with a square neckline front and back, long puff sleeves and ribbing at the cuffs and hem.

PRODUCT: 845790006
Ankle-length trousers in soft jersey with an elasticated waist, side pockets and gently tapered legs with sewn-in turn-ups at the hems.

PRODUCT: 911108003
Gently fitted shirt in cotton poplin with a pointed collar, buttons down the front and a yoke at the back. Long puff sleeves with buttoned cuffs, and a gently rounded hem.

PRODUCT: 873884005
Short skirt in imitation leather with a high waist and a concealed zip at the back. Lined. The polyurethane content of the skirt is water-based and free from DMF.

PRODUCT: 814762002
V-neck top in airy, crinkled chiffon with lace sections and narrow, adjustable shoulder straps.

PRODUCT: 694848003
Non-wired bras in soft cotton jersey with padded cups that shape the bust and provide good support. Narrow, adjustable shoulder straps and a hook-and-eye fastening at the back.

PRODUCT: 754238015
Sports bra in fast-drying functional fabric. Lined front with removable inserts, a racer back and a wide elasticated hem. Extra firm support.

PRODUCT: 848988001
Short dress in a patterned crêpe weave with a flounce-trimmed V-neck, wrapover front with ties at one side, and long sleeves with buttoned cuffs. Unlined.

PRODUCT: 715624001

```
Long-sleeved top in soft sweatshirt fabric with a jersey-lined, drawstring hood, kangaroo pocket and ribbing at the cuffs and hem.
Soft brushed inside.
```

Evaluate Model on Sample

In [95]:
```
y_true = ts_test_df.customer_id.values # get customer id
y_true.size
```

Out[95]:
```
30
```

In [97]:
```
y_pred = []
for index, cus in ts_test_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 = knn_model.kneighbors(q_cus_scaled, 1)[1][0]
    for c in result:
        pred_cus = y_true[c]
        y_pred.append(pred_cus)
len(y_pred)
```

Out[97]:
```
30
```

In [98]:
```
precision_score(y_true, y_pred, average='micro')
```

Out[98]:
```
0.03333333333333333
```

In [99]:
```
accuracy_score(y_true, y_pred)
```

Out[99]:
```
0.03333333333333333
```

Generate Predictions File

In [63]:
```
#H&M Collaborative KNN Model Based Recommendation System
def hm_rec_sys(r_model, cus_df, write_file):

    #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:
                p = names.iloc[n]
                r.append("0" + str(p))
                prediction =  ' '.join(r)
            #write predictions to csv file
            output.write(prediction + '\n')
            clear_output(wait=True)
            display('Processed Row: ' + str(index))
```

In [64]:
```
#we now generate our intial predictions list and save it as a csv file
hm_rec_sys(knn_model, cus_pred_df, "data/ros_predictions3.csv")
```

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

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
```