Big Basket

January 26, 2022

1 Big basket Analysis

1.1 Creating a class for all the basic functions on the dataset

[5]:
```
import matplotlib.pyplot as plt
import pandas as pd
```

[8]:
```
class big_basket:
    def __init__(self):
        ''' '''
        import pandas as pd
        self.data=pd.read_excel('bigbasketcasedataset1.xlsx','Sheet1') #␣
↪Reading the sheet 1

    def get_unique(self):
        ''' To get unqiue details about the dataset '''
        print('[+] The dataset has -',self.data.shape[0],' data points')
        print('                    -',self.data['Member'].nunique(),' customer␣
↪details')
        print('                    -',self.data['Description'].nunique(),'␣
↪unique products')
        print('                    - data for ',(self.data['Date'].max()-self.
↪data['Date'].min()).days,' days' )
```

[14]:
```
bb=big_basket()
```

[15]:
```
bb.get_unique()
```

```
[+] The dataset has - 61208  data points
                    - 106  customer details
                    - 215  unique products
                    - data for  1367  days
```

2 Exploratory Data Analysis

2.1 Popularity based recommender system

[17]:
```
bb.data['Description'].value_counts()[0:15]
```

[17]:
```
Other Vegetables     4537
Beans                4503
Root Vegetables      4247
Other Dals           3212
Organic F&V          3089
Gourd & Cucumber     2939
Whole Spices         2933
Brinjals             2539
Namkeen              2206
Banana               2157
Exotic Vegetables    1385
Moong Dal            1353
Sugar                1324
Toor Dal             1285
Sooji & Rava         1259
Name: Description, dtype: int64
```

The above are the product with maximum sales. Hence these can be recommended for new users, when we do not have any specific user details

Adding some new columns

[18]:
```
bb.data['Month']=bb.data['Date'].apply(lambda x: x.month )
bb.data['Day']=bb.data['Date'].apply(lambda x: x.day )
bb.data['Year']=bb.data['Date'].apply(lambda x: x.year )
```

[19]:
```
bb.data.head()
```

[19]:

	Unnamed: 0	Member	Order	SKU	Date	Description	lastdate	Month	Day	Year
0	0	M09736	6468572	34993740	2014-09-22	Other Sauces	NaN	9	22	2014
1	1	M09736	6468572	15669800	2014-09-22	Cashews	NaN	9	22	2014
2	2	M09736	6468572	34989501	2014-09-22	Other Dals	NaN	9	22	2014
3	3	M09736	6468572	7572303	2014-09-22	Namkeen	NaN	9	22	2014
4	4	M09736	6468572	15669856	2014-09-22	Sugar	NaN	9	22	2014

3 Checking for missing values or discrepancies

[20]:
```
bb.data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61208 entries, 0 to 61207
Data columns (total 10 columns):
Unnamed: 0     61208 non-null int64
Member         61208 non-null object
Order          61208 non-null int64
SKU            61208 non-null int64
Date           61208 non-null datetime64[ns]
Description    61208 non-null object
lastdate       0 non-null float64
Month          61208 non-null int64
Day            61208 non-null int64
Year           61208 non-null int64
dtypes: datetime64[ns](1), float64(1), int64(6), object(2)
memory usage: 4.7+ MB
```

Seems like there are no NAN values

[21]:
```
bb.data['Member'].unique()
```

[21]:
```
array(['M09736', 'M39021', 'M47229', 'M76390', 'M77779', 'M78365',
       'M78720', 'M82651', 'M84827', 'M86304', 'M86572', 'M90375',
       'M91098', 'M96365', 'M99030', 'M99206', 'M04158', 'M08075',
       'M09303', 'M12050', 'M12127', 'M14746', 'M16218', 'M16611',
       'M18732', 'M22037', 'M25900', 'M27458', 'M27871', 'M31101',
       'M31908', 'M31966', 'M32039', 'M32409', 'M32449', 'M32480',
       'M32655', 'M33064', 'M33422', 'M33491', 'M33558', 'M33745',
       'M33767', 'M34566', 'M35070', 'M35464', 'M35538', 'M35649',
       'M36366', 'M36432', 'M36702', 'M36876', 'M37253', 'M37600',
       'M38622', 'M40184', 'M41700', 'M41747', 'M41781', 'M42182',
       'M42513', 'M42827', 'M43189', 'M43831', 'M43977', 'M44156',
       'M45375', 'M45470', 'M46325', 'M46328', 'M46575', 'M46687',
       'M48101', 'M48154', 'M48938', 'M50038', 'M50094', 'M50420',
       'M50767', 'M51043', 'M51278', 'M52629', 'M54100', 'M54345',
       'M54382', 'M54619', 'M54796', 'M55932', 'M56255', 'M56309',
       'M56368', 'M56489', 'M56516', 'M56897', 'M57093', 'M57327',
       'M57354', 'M58761', 'M58939', 'M59012', 'M59232', 'M62656',
       'M62833', 'M63404', 'M64055', 'M64379'], dtype=object)
```

[22]:
```
bb.data['Order'].nunique()
```

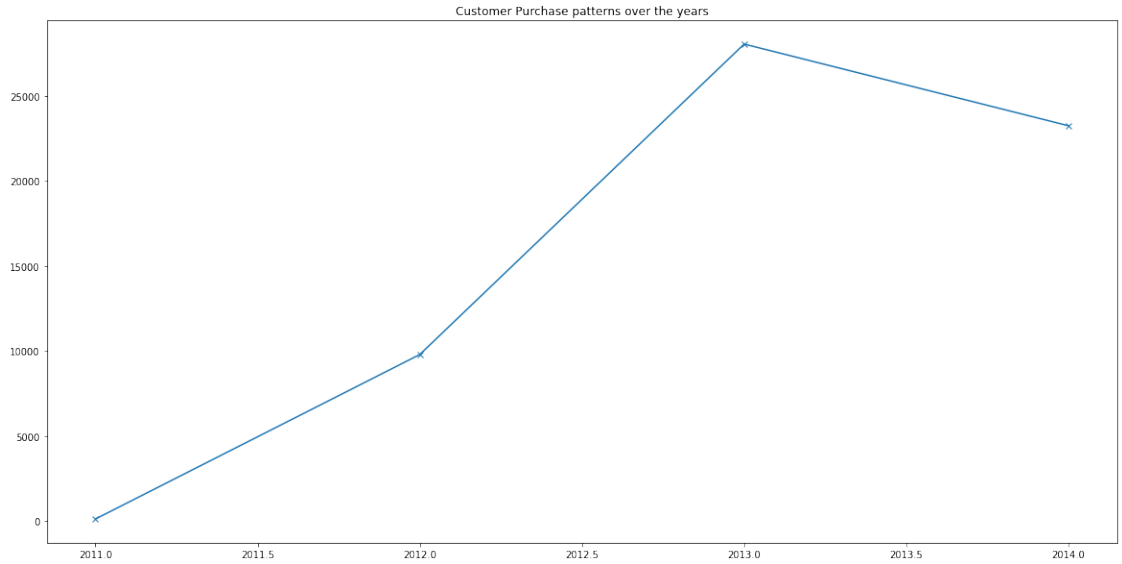
[22]:
```
8275
```

The dataset does not seem to have any discrepancies

4 Understanding the purchase patterns over time

[23]:
```
plt.figure(figsize=(20,10))
plt.plot(bb.data.groupby(by='Year').count()['Order'].index,bb.data.
 ↪groupby(by='Year').count()['Order'].values,marker='x')
plt.title('Customer Purchase patterns over the years')
```

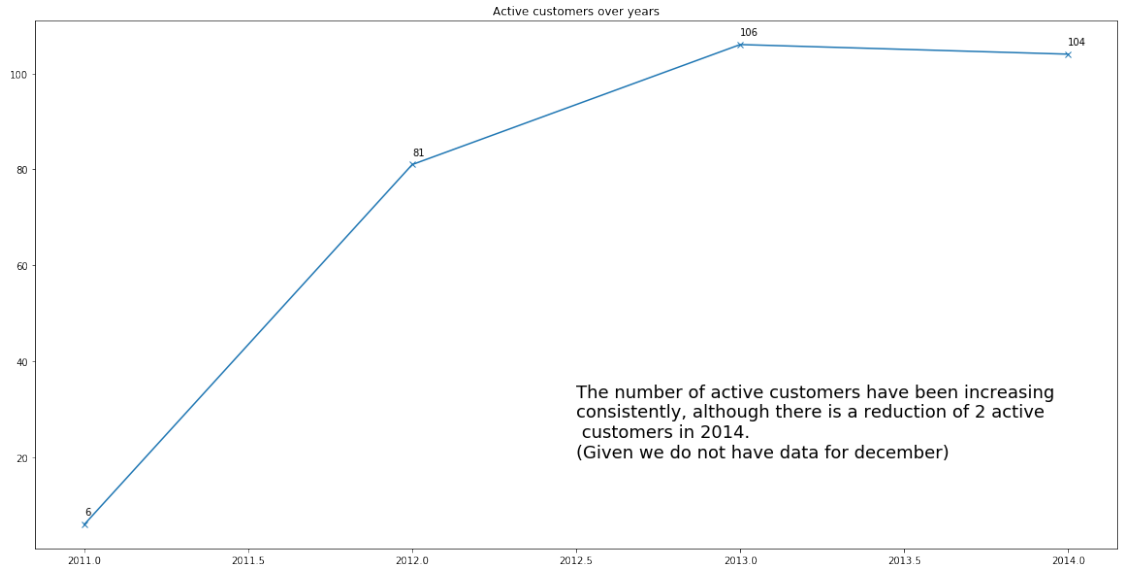
[23]:
```
Text(0.5, 1.0, 'Customer Purchase patterns over the years')
```



[26]:
```
# Trying to understand the reason for drop in 2014
bb.data['Date'].max()

# We have data till 8th December, which is almost the whole year.
```

[26]:
```
Timestamp('2014-12-08 00:00:00')
```

[27]:
```
# Checking the number of customers over years
plt.figure(figsize=(20,10))
customers_over_years=[bb.data[bb.data['Year']==year]['Member'].nunique()   for␣
 ↪year in bb.data['Year'].unique()]
plt.plot(bb.data['Year'].unique(),customers_over_years,marker='x')
plt.title('Active customers over years')
for year,value in zip(bb.data['Year'].unique(),customers_over_years):
    plt.text(year,value+2,str(value))
plt.text(2012.5, 20, 'The number of active customers have been increasing␣
 ↪\nconsistently, although there is a reduction of 2 active\n customers in␣
 ↪2014. \n(Given we do not have data for december)', fontsize = 18)
plt.show()
```

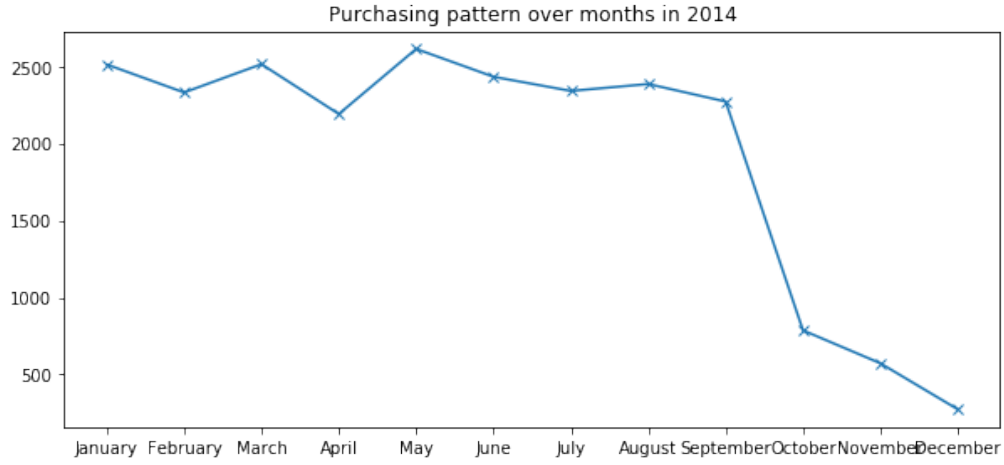


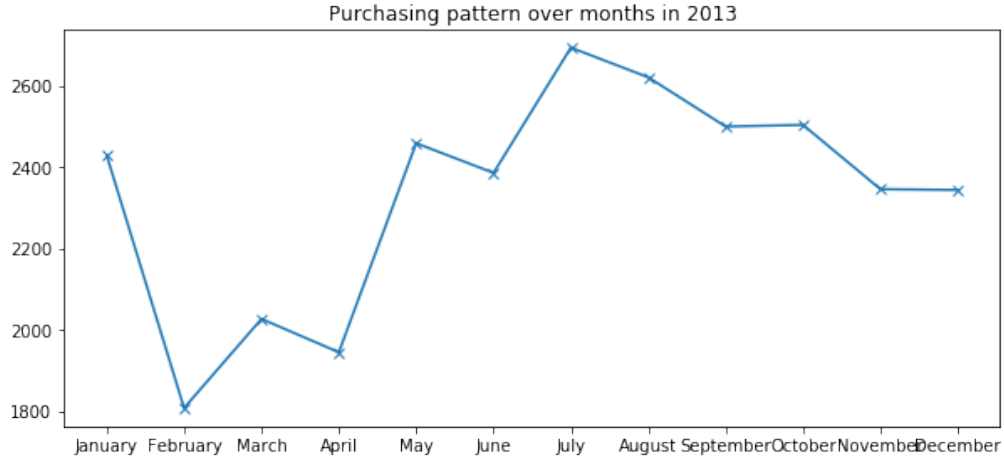
[28]:
```
bb.data[bb.data['Year']==2013].groupby(by='Month').count()['Member'].values
```

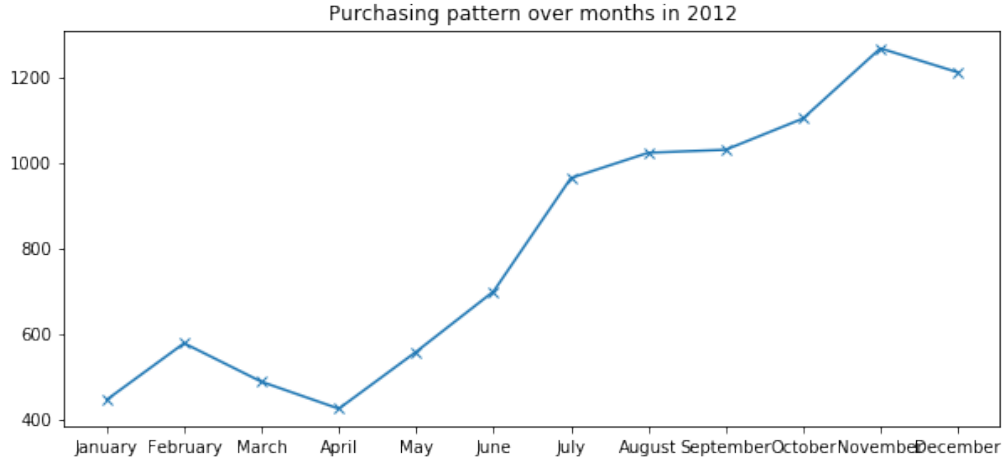
[28]:
```
array([2428, 1807, 2026, 1944, 2458, 2385, 2693, 2620, 2499, 2503, 2345,
       2343])
```

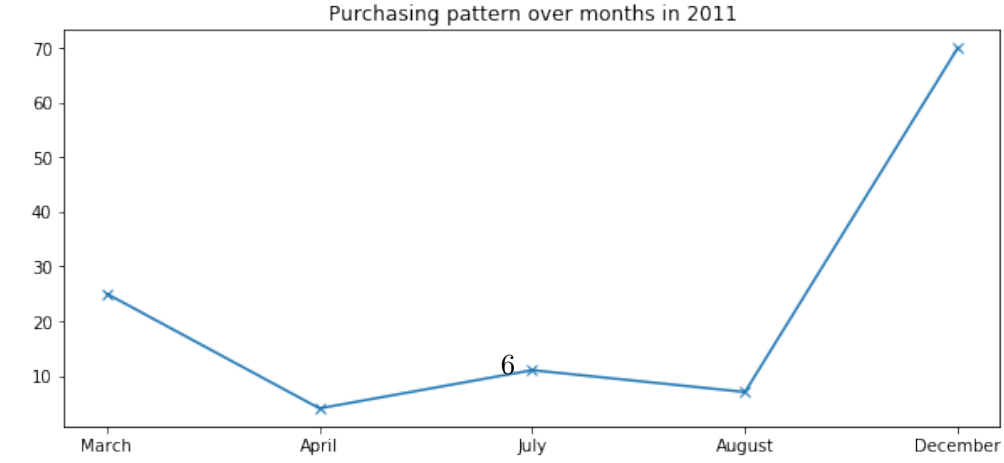
[29]:
```
months={1:'January',2:'February',3:'March',4:'April',5:'May',6:'June',7:
 ↪'July',8:'August',9:'September',10:'October',11:'November',12:'December'}
```

[30]:
```
# Understanding the change in purchase habbits over the months
plt.figure(figsize=(10,20))
i=1
for year in bb.data['Year'].unique():
    plt.subplot(4,1,i)
    data=bb.data[bb.data['Year']==year].groupby(by='Month').count()['Member']
    plt.plot([months[i] for i in data.index],data.values,marker='x')
    plt.title('Purchasing pattern over months in '+str(year))
    i+=1
```


Purchasing pattern over months in 2014
2500
2000
1500
1000
500
January
February
March
April
May
June
July
August
September
October
November
December


Purchasing pattern over months in 2013
2600
2400
2200
2000
1800
January
February
March
April
May
June
July
August
September
October
November
December


Purchasing pattern over months in 2012
1200
1000
800
600
400
January
February
March
April
May
June
July
August
September
October
November
December

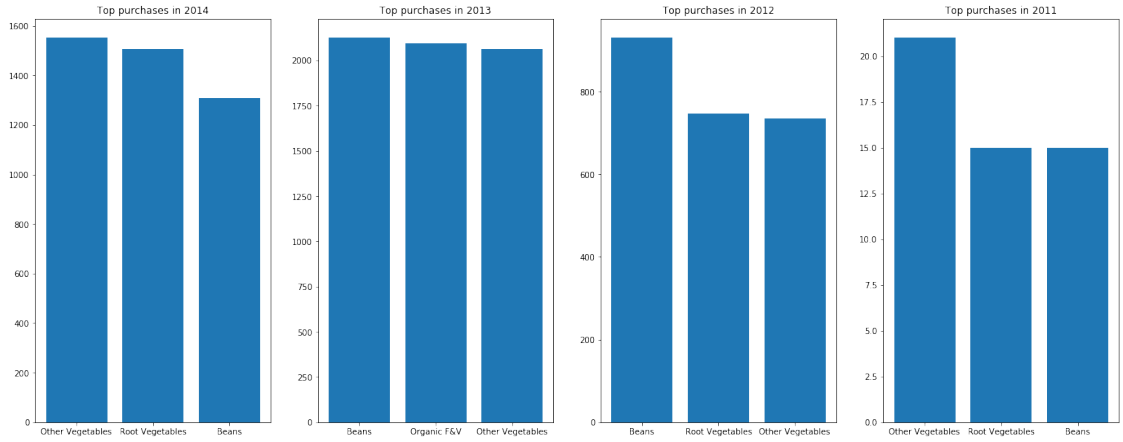

Purchasing pattern over months in 2011
70
60
50
40
30
20
10
March
April
July
August
December

It can be seen that there is considerable increase in customer purchases till 2014 May, and the purchases has been dropping since then

5 Customer Purchasing patterns over time

5.1 Variation of Customer Purcharse habbits

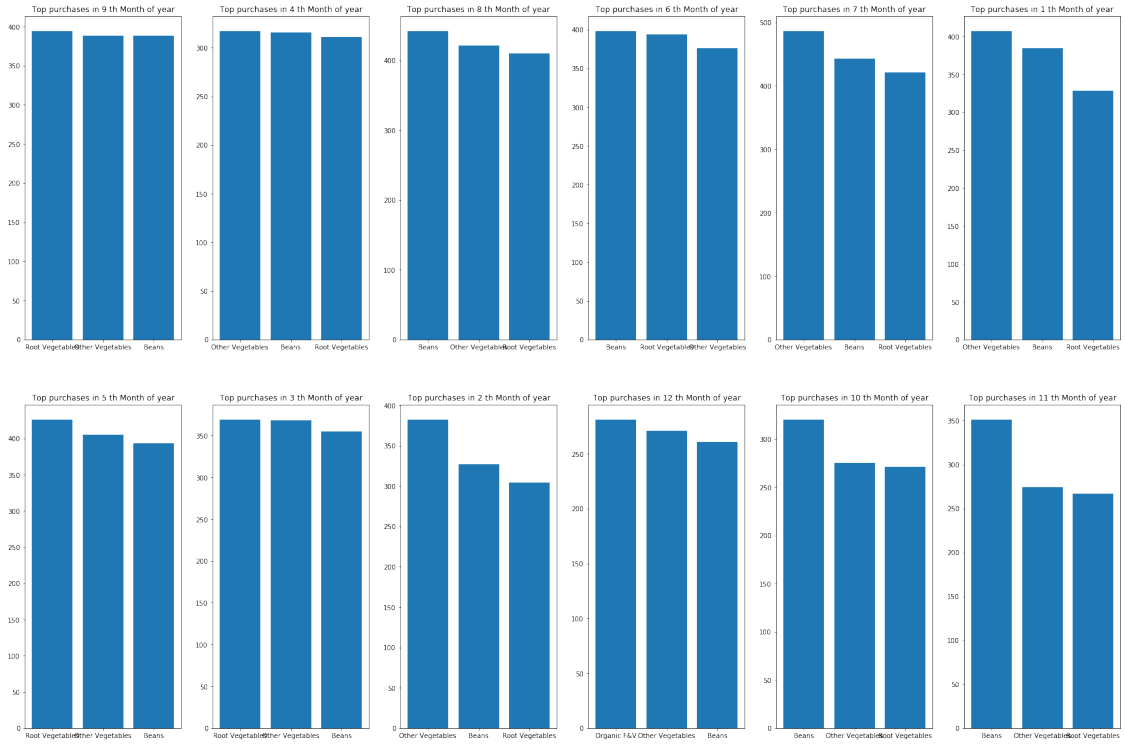
[8]:
```
plt.figure(figsize=(30,20))
i=1
for year in bb.data['Year'].unique():
    x_axis=[bb.data[bb.data['Year']==year]['Description'].value_counts()[0:3].
 ↪index]
    y_axis=[bb.data[bb.data['Year']==year]['Description'].value_counts()[0:3].
 ↪values]
    plt.subplot(2,5,i)
    plt.bar(x_axis[0],y_axis[0])
    plt.title('Top purchases in '+str(year))
    i+=1
```



5.1.1 It seems that Beans, Root Vegatables, and Organic Vegatables are the most purchsed product over the years

[9]:
```
plt.figure(figsize=(30,20))
i=1
for month in bb.data['Month'].unique():
    x_axis=[bb.data[bb.data['Month']==month]['Description'].value_counts()[0:3].
 ↪index]
```

```
    y_axis=[bb.data[bb.data['Month']==month]['Description'].value_counts()[0:3].
↪values]
    plt.subplot(2,6,i)
    plt.bar(x_axis[0],y_axis[0])
    plt.title('Top purchases in '+str(month)+' th Month of year')
    i+=1
```



5.1.2 Based on the most purchased products and the changing trends over the years and months, it seems that Beans, Other vegatables and Root vegetables are something which can be recommended to new users

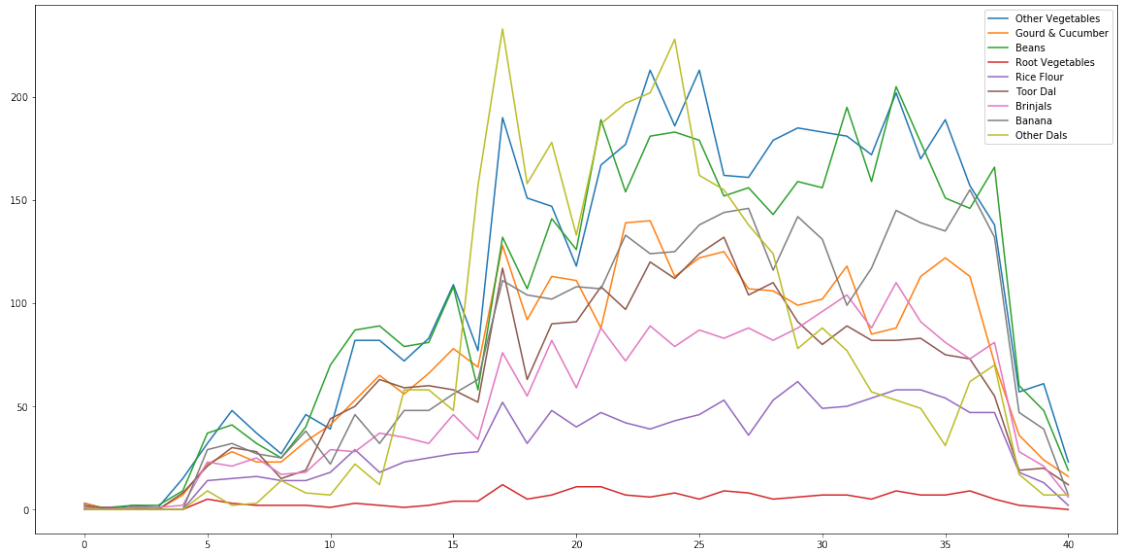
5.1.3 Understanding the purchase differences over time for the top 5 most purchased products

[152]:
```
# Finding the 3 items which are top 3 in each month

top_few=[]
top_few_values=[[] for i in range(11)]
for year in np.sort(bb.data['Year'].unique()):
    for month in np.sort(bb.data[bb.data['Year']==year]['Month'].unique()):
        top=bb.data[(bb.data['Year']==year)&(bb.
↪data['Month']==month)]['Description'].value_counts().index[0:3]
```

```
        top_values=bb.data[(bb.data['Year']==year)&(bb.
↪data['Month']==month)]['Description'].value_counts().values[0:3]
        for item in top:
            if item not in top_few:
                top_few.append(item)


top_few_values=[[] for i in range(11)]
for year in np.sort(bb.data['Year'].unique()):
    for month in np.sort(bb.data[bb.data['Year']==year]['Month'].unique()):
        top=bb.data[(bb.data['Year']==year)&(bb.
↪data['Month']==month)]['Description'].value_counts().index
        top_values=bb.data[(bb.data['Year']==year)&(bb.
↪data['Month']==month)]['Description'].value_counts().values
        idx_list=[]
        for item in top_few:
            idx=top_few.index(item)
            try:
                top_few_values[idx].append(top_values[list(top).index(item)])
                idx_list.append(idx)
            except:
                pass
        for i in range(11):
            if i not in idx_list:
                top_few_values[i].append(0)
```

[157]:
```
# Creating top_few plots
plt.figure(figsize=(20,10))
plt.plot(np.arange(len(top_few_values[0])),top_few_values[0])
plt.plot(np.arange(len(top_few_values[0])),top_few_values[1])
plt.plot(np.arange(len(top_few_values[0])),top_few_values[3])
plt.plot(np.arange(len(top_few_values[0])),top_few_values[4])
plt.plot(np.arange(len(top_few_values[0])),top_few_values[5])
plt.plot(np.arange(len(top_few_values[0])),top_few_values[6])
plt.plot(np.arange(len(top_few_values[0])),top_few_values[7])
plt.plot(np.arange(len(top_few_values[0])),top_few_values[8])
plt.plot(np.arange(len(top_few_values[0])),top_few_values[9])
plt.legend(top_few)
```

[157]:
```
<matplotlib.legend.Legend at 0x7f51eb53c630>
```



6 **It can be seen that the orders have reduced considerably from May 2014. It was also observed that the total number of customers placing order has not decreased. But the quantity of orders placed seems to be decreasin.**

7 **It could be beacause of reduced inentory or some other reason unknown. Need to validate hypothesis**

8 **Creating clusters for different user groups**

[158]:
```
top1=[]
top2=[]
top3=[]
top4=[]
members=[]
for member in bb.data['Member'].unique():
    members.append(member)
    top1.append(bb.data[bb.data['Member']==member]['Description'].
 ↪value_counts()[0:4].index[0])
    top2.append(bb.data[bb.data['Member']==member]['Description'].
 ↪value_counts()[0:4].index[1])
    top3.append(bb.data[bb.data['Member']==member]['Description'].
 ↪value_counts()[0:4].index[2])
    top4.append(bb.data[bb.data['Member']==member]['Description'].
 ↪value_counts()[0:4].index[3])
```

[159]:
```
t1=[]
t2=[]
t3=[]
t4=[]
for i,row in bb.data.iterrows():
    index = members.index(row['Member'])
    t1.append(top1[index])
    t2.append(top2[index])
    t3.append(top3[index])
    t4.append(top4[index])
```

[160]:
```
bb.data['Top1']=t1
bb.data['Top2']=t2
bb.data['Top3']=t3
bb.data['Top4']=t4
```

Label Encoding the data

[161]:
```
products=list(bb.data['Top1'])+list(bb.data['Top2'])+list(bb.
 ↪data['Top3'])+list(bb.data['Top4'])
products=list(set(products))
product_dictionary={}
product_dictionary={ products[i]: i for i in range(len(products))}
```

[162]:
```
cluster_data=bb.data[['Top1', 'Top2', 'Top3', 'Top4']]
for column in cluster_data.columns:
    cluster_data[column]=cluster_data[column].apply(lambda x:␣
 ↪product_dictionary[x])
```

```
/home/blink/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3:
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: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
```

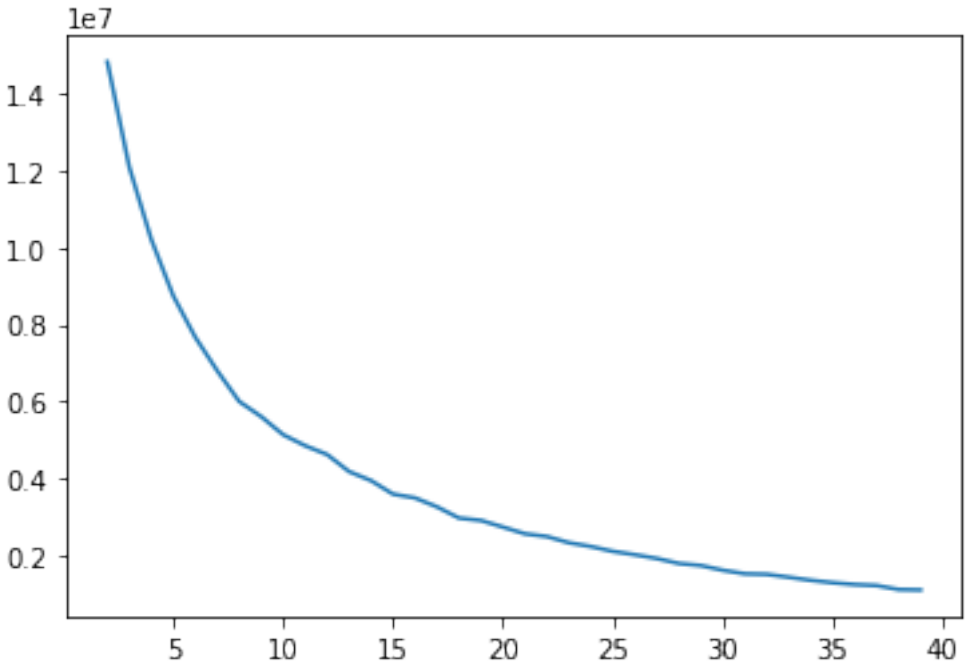
8.0.1 Creating clusters with Kmeans

[163]:
```
import numpy as np
from sklearn.cluster  import KMeans
from tqdm import tqdm
scores=[]
for i in tqdm(range(2,40)):
    km=KMeans(n_clusters=i)
    km.fit(cluster_data)
```

```
    scores.append(km.inertia_)

plt.plot(np.arange(2,40),scores)
```

```
/home/blink/anaconda3/lib/python3.7/site-
packages/sklearn/utils/validation.py:37: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  LARGE_SPARSE_SUPPORTED = LooseVersion(scipy_version) >= '0.14.0'
100%|      | 38/38 [00:40<00:00,  1.61s/it]
```

[163]:
```
[<matplotlib.lines.Line2D at 0x7f51e29b57b8>]
```



[164]:
```
km=KMeans(n_clusters=5)
km.fit(cluster_data)
```

[164]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

[165]:
```
classes=km.predict(cluster_data)
```

[166]:
```
bb.data['Cluster']=classes
```

[167]:
```
bb.data[bb.data['Cluster']==1].head()
```

[167]:
```
      Unnamed: 0  Member    Order       SKU       Date       Description  \
1037        1063  M47229  6728685  15668455 2014-11-07          Brinjals
1038        1064  M47229  6728685  15668381 2014-11-07  Other Vegetables
1039        1065  M47229  6728685  15668453 2014-11-07          Brinjals
1040        1066  M47229  6728685   7729965 2014-11-07    Sunflower Oils
1041        1067  M47229  6728685   7621580 2014-11-07   Diapers & Wipes

      lastdate  Month  Day  Year   Top1            Top2              Top3  \
1037       NaN     11    7  2014  Beans  Root Vegetables  Other Vegetables
1038       NaN     11    7  2014  Beans  Root Vegetables  Other Vegetables
1039       NaN     11    7  2014  Beans  Root Vegetables  Other Vegetables
1040       NaN     11    7  2014  Beans  Root Vegetables  Other Vegetables
1041       NaN     11    7  2014  Beans  Root Vegetables  Other Vegetables

                 Top4  Cluster
1037  Diapers & Wipes        1
1038  Diapers & Wipes        1
1039  Diapers & Wipes        1
1040  Diapers & Wipes        1
1041  Diapers & Wipes        1
```

[168]:
```
bb.data[bb.data['Cluster']==0].head()
```

[168]:
```
   Unnamed: 0  Member    Order       SKU       Date   Description  lastdate  \
0           0  M09736  6468572  34993740 2014-09-22  Other Sauces       NaN
1           1  M09736  6468572  15669800 2014-09-22       Cashews       NaN
2           2  M09736  6468572  34989501 2014-09-22    Other Dals       NaN
3           3  M09736  6468572   7572303 2014-09-22       Namkeen       NaN
4           4  M09736  6468572  15669856 2014-09-22         Sugar       NaN

   Month  Day  Year    Top1   Top2        Top3             Top4  Cluster
0      9   22  2014  Banana  Beans  Other Dals  Root Vegetables        0
1      9   22  2014  Banana  Beans  Other Dals  Root Vegetables        0
2      9   22  2014  Banana  Beans  Other Dals  Root Vegetables        0
3      9   22  2014  Banana  Beans  Other Dals  Root Vegetables        0
4      9   22  2014  Banana  Beans  Other Dals  Root Vegetables        0
```

[169]:
```
bb.data[bb.data['Cluster']==2].head()
```

[169]:
```
     Unnamed: 0  Member    Order       SKU       Date       Description  \
610         626  M39021  6422636   7580802 2014-09-28    Sunflower Oils
611         627  M39021  6422636  15668453 2014-09-28          Brinjals
612         628  M39021  6422636  15668375 2014-09-28   Root Vegetables
613         629  M39021  6422636  15668379 2014-09-28  Other Vegetables
614         630  M39021  6422636  15669760 2014-09-28      Whole Spices

     lastdate  Month  Day  Year   Top1            Top2              Top3  \
```

```
610       NaN      9   28  2014  Beans  Root Vegetables  Sunflower Oils
611       NaN      9   28  2014  Beans  Root Vegetables  Sunflower Oils
612       NaN      9   28  2014  Beans  Root Vegetables  Sunflower Oils
613       NaN      9   28  2014  Beans  Root Vegetables  Sunflower Oils
614       NaN      9   28  2014  Beans  Root Vegetables  Sunflower Oils

                Top4  Cluster
610  Other Vegetables        2
611  Other Vegetables        2
612  Other Vegetables        2
613  Other Vegetables        2
614  Other Vegetables        2
```

[170]:
```
bb.data[bb.data['Cluster']==3].head()
```

[170]:
```
      Unnamed: 0  Member    Order       SKU       Date      Description  \
1423        1465  M76390  6430330  15668375 2014-09-30  Root Vegetables
1424        1466  M76390  6430330  34987569 2014-09-30  Powdered Spices
1425        1467  M76390  6447929   7580802 2014-06-10   Sunflower Oils
1426        1468  M76390  6447929  15669787 2014-06-10         Raw Rice
1427        1469  M76390  6447929  15669860 2014-06-10        Moong Dal

      lastdate  Month  Day  Year     Top1         Top2   Top3  \
1423       NaN      9   30  2014  Namkeen  Organic F&V  Beans
1424       NaN      9   30  2014  Namkeen  Organic F&V  Beans
1425       NaN      6   10  2014  Namkeen  Organic F&V  Beans
1426       NaN      6   10  2014  Namkeen  Organic F&V  Beans
1427       NaN      6   10  2014  Namkeen  Organic F&V  Beans

                Top4  Cluster
1423  Other Vegetables        3
1424  Other Vegetables        3
1425  Other Vegetables        3
1426  Other Vegetables        3
1427  Other Vegetables        3
```

[171]:
```
bb.data[bb.data['Cluster']==4].head()
```

[171]:
```
       Unnamed: 0  Member    Order       SKU       Date          Description  \
16222       16483  M32039  7350873  93073679 2013-10-18              Namkeen
16223       16484  M32039  7350873   7569802 2013-10-18              Namkeen
16224       16485  M32039  7350873  15668378 2013-10-18     Other Vegetables
16225       16486  M32039  7350873  21408947 2013-10-18  Other Rice Products
16226       16487  M32039  7350873  15668460 2013-10-18     Gourd & Cucumber

       lastdate  Month  Day  Year              Top1        Top2  \
16222       NaN     10   18  2013  Other Vegetables  Other Dals
```

```
16223       NaN     10   18  2013  Other Vegetables  Other Dals
16224       NaN     10   18  2013  Other Vegetables  Other Dals
16225       NaN     10   18  2013  Other Vegetables  Other Dals
16226       NaN     10   18  2013  Other Vegetables  Other Dals

                    Top3   Top4  Cluster
16222  Gourd & Cucumber  Beans        4
16223  Gourd & Cucumber  Beans        4
16224  Gourd & Cucumber  Beans        4
16225  Gourd & Cucumber  Beans        4
16226  Gourd & Cucumber  Beans        4
```

8.0.2 Now we can segment the customers into these clusters and give them recommendation based on the shopping patterns.

8.0.3 This will also help in classifying a new customer quickly based on their shopping patterns

8.0.4 The problem with this approach is, if a new product is added, the model will have to be trained again.

[ ]:
```
bb.data.to_csv('pre_preocessed_data.csv',index=False)
```

9 Loading the saved dataset

[558]:
```
import pandas as pd
bb=pd.read_csv('pre_preocessed_data.csv')
bb.drop(bb.columns[0:2],axis=1,inplace=True)
from datetime import datetime
bb['Date']=bb['Date'].apply(lambda x : datetime.strptime(x,'%Y-%m-%d'))
bb.head()
```

[558]:
```
    Member    Order       SKU       Date   Description  lastdate  Month  Day  \
0  M09736  6468572  34993740 2014-09-22  Other Sauces       NaN      9   22
1  M09736  6468572  15669800 2014-09-22       Cashews       NaN      9   22
2  M09736  6468572  34989501 2014-09-22    Other Dals       NaN      9   22
3  M09736  6468572   7572303 2014-09-22       Namkeen       NaN      9   22
4  M09736  6468572  15669856 2014-09-22         Sugar       NaN      9   22

   Year    Top1   Top2        Top3             Top4  Cluster
0  2014  Banana  Beans  Other Dals  Root Vegetables        0
1  2014  Banana  Beans  Other Dals  Root Vegetables        0
2  2014  Banana  Beans  Other Dals  Root Vegetables        0
3  2014  Banana  Beans  Other Dals  Root Vegetables        0
4  2014  Banana  Beans  Other Dals  Root Vegetables        0
```

Coverting date into a number

[559]:
```
def datestdtojd (stddate):
    fmt='%Y-%m-%d'
    sdtdate = datetime.strptime(stddate, fmt)
    year=str(sdtdate.year)
    sdtdate = sdtdate.timetuple()
    jdate = sdtdate.tm_yday
    return(int(year+str(jdate)))
```

[560]:
```
bb['current_julian_date']=bb['Date'].apply(lambda x: x.to_julian_date())
```

[561]:
```
bb.head()
```

[561]:

	Member	Order	SKU	Date	Description	lastdate	Month	Day	\
0	M09736	6468572	34993740	2014-09-22	Other Sauces	NaN	9	22	
1	M09736	6468572	15669800	2014-09-22	Cashews	NaN	9	22	
2	M09736	6468572	34989501	2014-09-22	Other Dals	NaN	9	22	
3	M09736	6468572	7572303	2014-09-22	Namkeen	NaN	9	22	
4	M09736	6468572	15669856	2014-09-22	Sugar	NaN	9	22	

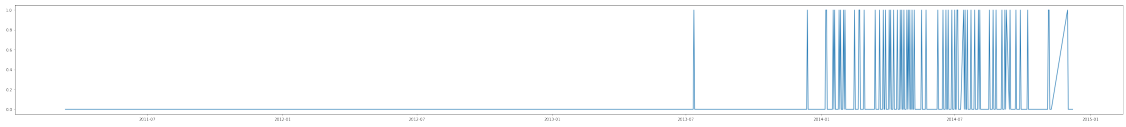
	Year	Top1	Top2	Top3	Top4	Cluster	\
0	2014	Banana	Beans	Other Dals	Root Vegetables	0	
1	2014	Banana	Beans	Other Dals	Root Vegetables	0	
2	2014	Banana	Beans	Other Dals	Root Vegetables	0	
3	2014	Banana	Beans	Other Dals	Root Vegetables	0	
4	2014	Banana	Beans	Other Dals	Root Vegetables	0	

	current_julian_date
0	2456922.5
1	2456922.5
2	2456922.5
3	2456922.5
4	2456922.5

10 Understanding some customers purchase patterns

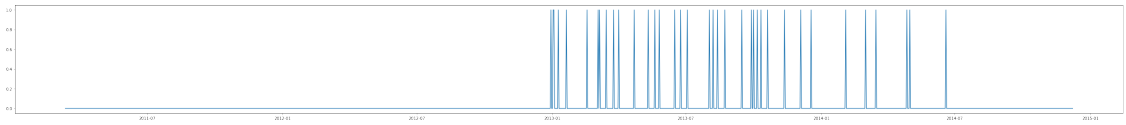
[562]:
```
purchase_dates=bb[bb['Member']==bb['Member'].unique()[0]]['Date'].unique()
status=[]
for date in np.sort(bb['Date'].unique()):
    if date in purchase_dates:
        status.append(1)
    else:
        status.append(0)
plt.figure(figsize=(50,5))
plt.xlim()
plt.plot(np.sort(bb['Date'].unique()),status)
```

[562]:
```
[<matplotlib.lines.Line2D at 0x7f51c99e15f8>]
```



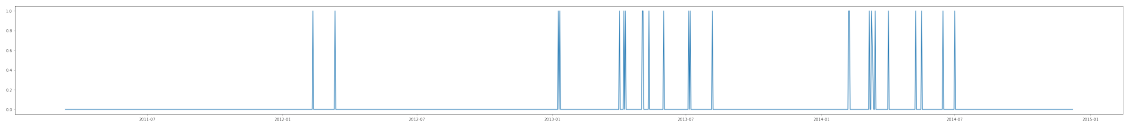
[563]:
```
purchase_dates=bb[bb['Member']==bb['Member'].unique()[20]]['Date'].unique()
status=[]
for date in np.sort(bb['Date'].unique()):
    if date in purchase_dates:
        status.append(1)
    else:
        status.append(0)
plt.figure(figsize=(50,5))
plt.xlim()
plt.plot(np.sort(bb['Date'].unique()),status)
```

[563]:
```
[<matplotlib.lines.Line2D at 0x7f51c9af17f0>]
```



[564]:
```
purchase_dates=bb[bb['Member']==bb['Member'].unique()[100]]['Date'].unique()
status=[]
for date in np.sort(bb['Date'].unique()):
    if date in purchase_dates:
        status.append(1)
    else:
        status.append(0)
plt.figure(figsize=(50,5))
plt.xlim()
plt.plot(np.sort(bb['Date'].unique()),status)
```

[564]:
```
[<matplotlib.lines.Line2D at 0x7f51c9a7f940>]
```



[565]:
```
pd.DataFrame(bb.groupby(by='Order').count().
 ↪sort_values(by='Member',ascending=False)['Member']).head()
```

[565]:

Order	Member
6738016	42
6438096	41
7696664	38
6506666	37
7597091	36

[566]:
```
pd.DataFrame(bb.groupby(by='Order').count().
 ↪sort_values(by='Member',ascending=False)['Member']).tail()
```

[566]:

Order	Member
7931542	1
8272504	1
7707525	1
8131347	1
7927146	1

Based on analysis it seems that there are different types of customers Customers who are consistent

Customers who are rarely use the app less frequently

Customers who have stopped using the service

Customers who have used the service only once or twice

We shall try to predict the next customer purchase date based on the previous 4 purchase dates. For that purpose, we will have to drop the data of all the customers who have data less than 4 . That is

Observation window : last 4 purchases (It can extend over a few days or months)

[567]:
```
df=pd.DataFrame(bb.groupby(by='Order').count().
 ↪sort_values(by='Member',ascending=False)['Member'])
selected_customers=df[df['Member']>4].index
```

[568]:
```
dataset=bb[bb['Order'].isin(selected_customers)]
dataset=dataset[['Member','Order','Description','Cluster','current_julian_date']]
dataset.head()
```

[568]:

	Member	Order	Description	Cluster	current_julian_date
0	M09736	6468572	Other Sauces	0	2456922.5
1	M09736	6468572	Cashews	0	2456922.5
2	M09736	6468572	Other Dals	0	2456922.5
3	M09736	6468572	Namkeen	0	2456922.5

```
4  M09736  6468572          Sugar         0             2456922.5
```

[569]:
```
dataset.reset_index(inplace=True)
dataset.drop(dataset.columns[0],axis=1,inplace=True)
```

Combining the entries which belong to the same order

[570]:
```
# Getting indexes of same orders
order_indexes={}
current_date=[]
member=[]
for order in tqdm(dataset['Order'].unique()):
    order_indexes[order]=dataset[dataset['Order']==order].index.values
    current_date.append(dataset[dataset['Order']==order]['current_julian_date'].
 ↪unique()[0])
    member.append(dataset[dataset['Order']==order]['Member'].unique()[0])
orders=order_indexes.keys()
dataset.drop(['Order'],axis=1,inplace=True)
```

```
100%|      | 5739/5739 [00:16<00:00, 339.08it/s]
```

[571]:
```
common_orders=list(order_indexes.values())
products=[]
for indexes in tqdm(common_orders):
    try:
        products.append(list(dataset.iloc[indexes]['Description'].unique()))
    except:
        print('Error')
```

```
100%|      | 5739/5739 [00:04<00:00, 1304.94it/s]
```

[572]:
```
training_data=pd.DataFrame()
training_data['Order']=orders
training_data['Member']=member
training_data['products']=products
training_data['current_date']=current_date
```

[573]:
```
p1_master=[]
p2_master=[]
p3_master=[]
master_dataset=pd.DataFrame()
for user in tqdm(training_data['Member'].unique()):
    df=pd.DataFrame(training_data[training_data['Member']==user].
 ↪sort_values(by='current_date',ascending=False))
    df.reset_index(inplace=True)
    df.drop(df.columns[0],axis=1,inplace=True)
    p1=[]
    p2=[]
    p3=[]
```

```
    for i,row in df.iterrows():
        try:
            p1.append(df.loc[i+1]['current_date'])
            p2.append(df.loc[i+2]['current_date'])
            p3.append(df.loc[i+3]['current_date'])
        except:
            p1.append(0)
            p2.append(0)
            p3.append(0)
    p1=p1[:-2]
    p2=p2[:-1]
    df['p1']=p1
    df['p2']=p2
    df['p3']=p3

    master_dataset=pd.concat([master_dataset,df],ignore_index=True)
```

```
100%|      | 106/106 [00:05<00:00, 21.56it/s]
```

master_dataset.to_csv('train.csv') # saving the intermediate result for faster re-execution

[621]:
```
master_dataset=pd.read_csv('train.csv')
```

[622]:
```
master_dataset.drop(list(master_dataset[master_dataset['p1']==0].
↪index),axis=0,inplace=True)
```

[623]:
```
master_dataset.drop(['Order','Member','products'],axis=1,inplace=True)
y=master_dataset['current_date']
x=master_dataset.drop(['current_date'],axis=1)
```

[624]:
```
x.drop(x.columns[0],axis=1,inplace=True)
```

[722]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

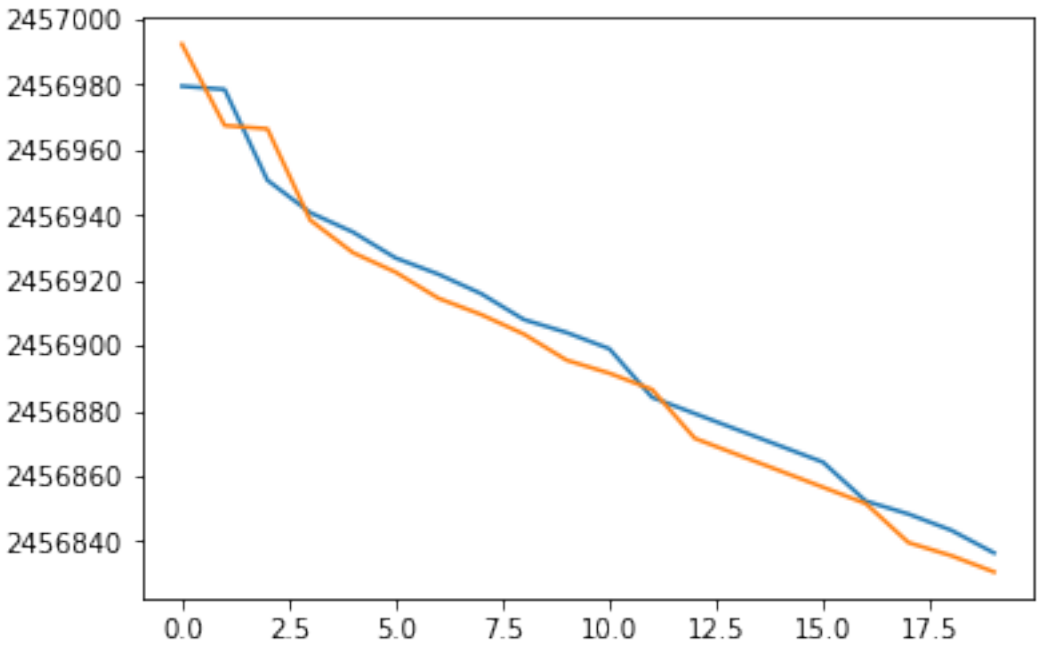
[726]:
```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)
lr.score(x_test,y_test)
```

[726]:
```
0.9950350111188072
```

[727]:
```
y_pred=lr.predict(x)
```

[728]:
```
plt.plot(y_pred[0:20])
plt.plot(y[0:20])
```

[728]: [<matplotlib.lines.Line2D at 0x7f51eb03bb70>]



10.0.1 Predicting the next purchase date

[641]:
```
julian.from_jd(lr.predict([x.iloc[0]]))
```

[641]:
```
datetime.datetime(2014, 11, 18, 3, 41, 17, 49489)
```

10.0.2 RMS of the model

[659]:
```
from sklearn.metrics import mean_squared_error
import math

mse=mean_squared_error(y_pred,y)
print('RMS : ',math.sqrt(mse))
```

```
RMS :  16.57876259986778
```

10.0.3 Predicting the next order date for each customer

[666]:
```
training_data.head()
```

[666]:

	Order	Member	products \
0	6468572	M09736	[Other Sauces, Cashews, Other Dals, Namkeen, S…
1	6486475	M09736	[Utensil Scrub Pads, Other Rice Products, Toor…
2	6504964	M09736	[Urad Dal, Boiled Rice, Jaggery, Other Dals, R…

```
3  6529569  M09736  [Sugar, Jaggery, Root Vegetables, Cakes, Urad …
4  6549521  M09736  [Banana, Cashews, Other Rice Products, Raw Pea…

   current_date
0     2456922.5
1     2456928.5
2     2456914.5
3     2456756.5
4     2456909.5
```

[674]:
```
p1_master=[]
p2_master=[]
p3_master=[]

prediction_data=pd.DataFrame()
for user in tqdm(training_data['Member'].unique()):
    member=[]
    df=pd.DataFrame(training_data[training_data['Member']==user].
 ↪sort_values(by='current_date',ascending=False))
    df.reset_index(inplace=True)
    df.drop(df.columns[0],axis=1,inplace=True)
    p1=[]
    p2=[]
    p3=[]
    for i,row in df.iterrows():
        member.append(user)
        try:
            p1.append(df.loc[i+1]['current_date'])
            p2.append(df.loc[i+2]['current_date'])
            p3.append(df.loc[i+3]['current_date'])
        except:
            p1.append(0)
            p2.append(0)
            p3.append(0)
    p1=p1[:-2]
    p2=p2[:-1]
    df['p1']=p1
    df['p2']=p2
    df['p3']=p3
    df['Member']=member
    prediction_data=pd.concat([prediction_data,df],ignore_index=True)
```

```
100%|      | 106/106 [00:05<00:00, 20.79it/s]
```

[700]:
```
members=[]
next_date=[]
for member in tqdm(training_data['Member'].unique()):
```

```
    # Making prediction based on the last order placed
    p1=prediction_data[prediction_data['Member']==member]['current_date'].max()
    p2=prediction_data[prediction_data['Member']==member]['p1'].max()
    p3=prediction_data[prediction_data['Member']==member]['p2'].max()
    members.append(member)
    next_date.append(julian.from_jd(lr.predict([[p1,p2,p3]])).date())
prediction=pd.DataFrame()
prediction['Member']=members
prediction['Predicted Next Order']=next_date
```

```
100%|      | 106/106 [00:00<00:00, 182.58it/s]
```

[702]:
```
prediction.head()
```

[702]:
```
   Member Predicted Next Order
0  M09736           2014-12-13
1  M39021           2014-10-13
2  M47229           2014-11-19
3  M76390           2014-12-15
4  M77779           2014-09-30
```

[704]:
```
prediction.to_csv('next_order_prediction.csv')
```

11 Understanding the SKU Column

[646]:
```
bb[bb['SKU']==bb['SKU'].unique()[1]]
```

[646]:
```
        Member    Order       SKU       Date Description  lastdate  Month  Day  \
1       M09736  6468572  15669800 2014-09-22     Cashews       NaN      9   22
40      M09736  6529569  15669800 2014-04-09     Cashews       NaN      4    9
44      M09736  6549521  15669800 2014-09-09     Cashews       NaN      9    9
2999    M78720  7425894  15669800 2013-09-14     Cashews       NaN      9   14
26716   M35649  7632815  15669800 2013-05-18     Cashews       NaN      5   18
27146   M36366  6423338  15669800 2014-09-28     Cashews       NaN      9   28
27175   M36366  6533376  15669800 2014-06-09     Cashews       NaN      6    9
27336   M36366  7730741  15669800 2013-02-17     Cashews       NaN      2   17
44616   M48938  8220406  15669800 2014-09-02     Cashews       NaN      9    2

        Year             Top1      Top2             Top3             Top4  \
1       2014           Banana     Beans       Other Dals   Root Vegetables
40      2014           Banana     Beans       Other Dals   Root Vegetables
44      2014           Banana     Beans       Other Dals   Root Vegetables
2999    2013            Chips    Banana          Namkeen  Other Vegetables
26716   2013  Root Vegetables    Banana      Whole Spices       Other Dals
27146   2014          Namkeen  Raw Rice             Beans            Chips
27175   2014          Namkeen  Raw Rice             Beans            Chips
27336   2013          Namkeen  Raw Rice             Beans            Chips
```

```
44616  2014       Other Dals  Brinjals  Other Vegetables   Root Vegetables

       Cluster  current_julian_date
1            0            2456922.5
40           0            2456756.5
44           0            2456909.5
2999         0            2456549.5
26716        1            2456430.5
27146        1            2456928.5
27175        1            2456817.5
27336        1            2456340.5
44616        3            2456902.5
```

[731]:
```
list(bb[bb['Order']==6468572]['Description'].values)
```

[731]:
```
array(['Other Sauces', 'Cashews', 'Other Dals', 'Namkeen', 'Sugar',
       'Banana', 'Sugar Cubes', 'Other Sweets', 'Other Dals',
       'Other Rice Products', 'Other Rice Products', 'Namkeen'],
      dtype=object)
```

[740]:
```
# Combining the orders together, as a single order is now split into multiple␣
 ↪rows
orders=[]
sku=[]
description=[]
member=[]
for order in tqdm(bb['Order'].unique()):
    sku.append(list(set(bb[bb['Order']==order]['SKU'].values)))
    orders.append(order)
    description.append(list(set(bb[bb['Order']==order]['Description'].values)))
    member.append((list(set(bb[bb['Order']==order]['Member'].values))[0]))
```

```
100%|      | 8275/8275 [00:15<00:00, 535.12it/s]
```

[741]:
```
sku_df=pd.DataFrame()
sku_df['Member']=member
sku_df['Order']=orders
sku_df['SKU']=sku
sku_df['Descriptions']=description
```

[744]:
```
sku_df.head()
```

[744]:
```
   Member    Order                                                SKU  \
0  M09736  6468572  [15669856, 34989440, 21409124, 15669830, 75697…
1  M09736  6486475  [15669789, 7585573, 15669830, 34991046, 156698…
2  M09736  6504964  [15669767, 34934493, 15669865, 15669965, 15669…
3  M09736  6529569  [7585573, 15669767, 15669800, 15669865, 156698…
4  M09736  6549521  [34989440, 21409124, 15669861, 15669800, 15669…
```

```
                                                Descriptions
0   [Cashews, Sugar Cubes, Banana, Other Dals, Oth…
1   [Utensil Scrub Pads, Boiled Rice, Other Rice P…
2   [Raisins, Urad Dal, Boiled Rice, Almonds, Othe…
3   [Cashews, Urad Dal, Raisins, Healthy Snacks, M…
4   [Cashews, Sugar Cubes, Banana, Other Rice Prod…
```

11.0.1 Creating an SKU Description mapping

[708]:
```
unique_sku=bb['SKU'].unique()
```

[713]:
```
sku_dictionary={sku : bb[bb['SKU']==sku]['Description'].values[0]  for sku in␣
↪unique_sku}
```

Stuck here

[ ]:
```

```