In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

1. Reading SQLite Data

In [3]:
```
import sqlite3
```

In [4]:
```
con=sqlite3.connect(r'C:\Users\Ansh\Downloads\database.sqlite')
```

In [5]:
```
df=pd.read_sql_query('Select * from reviews', con)
```

In [6]:
```
df.shape
```

Out[6]:
```
(568454, 10)
```

2. Data Preparation

In [8]:
```
#Removing Invalid Rows and Duplicate Rows
```

In [9]:
```
type(df)
```

Out[9]:
```
pandas.core.frame.DataFrame
```

In [10]:
```
df.head()
```

Out[10]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summa
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Go Qua Dog Fo
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not Advertis
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delig says it
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cou Medici
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great ta

In [11]:
```
df.columns
```

Out[11]:
```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

In [12]:
```
# HelpfulnessDenominator will always be greater than or equal to HelpfulnessNumerator
```

In [13]:
```
df['HelpfulnessNumerator'] > df['HelpfulnessDenominator']
```

Out[13]:
```
0         False
1         False
2         False
3         False
4         False
          ...
568449    False
568450    False
568451    False
568452    False
568453    False
Length: 568454, dtype: bool
```

In [14]:
```python
# All the invalid rows
```

In [15]:
```python
df[df['HelpfulnessNumerator'] > df['HelpfulnessDenominator']]
```

Out[15]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
44736	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200
64421	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800

In [16]:
```python
df_valid = df[df['HelpfulnessNumerator'] <= df['HelpfulnessDenominator']]
```

In [17]:
```python
df_valid.shape
```

Out[17]:
```
(568452, 10)
```

In [18]:
```python
# Removing Duplicate Rows
```

In [19]:
```python
df_valid.columns
```

Out[19]:
```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

In [20]:
```python
df_valid.duplicated(['UserId', 'ProfileName' ,'Time' ,'Text'])
```

Out[20]:
```
0         False
1         False
2         False
3         False
4         False
          ...
568449    False
568450    False
568451    False
568452    False
568453    False
Length: 568452, dtype: bool
```

In [21]:
```python
# All Duplicate Rows
```

In [22]:
```python
df_valid[df_valid.duplicated(['UserId', 'ProfileName' ,'Time' ,'Text'])]
```

Out[22]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
29	30	B0001PB9FY	A3HDKO7OW0QNK4	Canadian Fan	1	1	5	11078208
574	575	B000G6RYNE	A3PJZ8TU8FDQ1K	Jared Castle	2	2	5	12317184
1973	1974	B0017165OG	A2EPNS38TTLZYN	tedebear	0	0	3	13126752
2309	2310	B0001VWE0M	AQM74O8Z4FMS0	Sunshine	0	0	2	11276064
2323	2324	B0001VWE0C	AQM74O8Z4FMS0	Sunshine	0	0	2	11276064
...	...	...	...	...	...	...	...	
568409	568410	B0018CLWM4	A2PE0AGWV6OPL7	Dark Water Mermaid	3	3	5	13096512
568410	568411	B0018CLWM4	A88HLWDCU57WG	R28	2	2	5	13329792
568411	568412	B0018CLWM4	AUX1HSY8FX55S	DAW	1	1	5	13195008
568412	568413	B0018CLWM4	AVZ2OZ479Q9E8	Ai Ling Chow	0	0	5	13364352
568413	568414	B0018CLWM4	AI3Y26HLPYW4L	kimosabe	1	2	2	13300416

174521 rows × 10 columns

In [23]:
```
data = df_valid.drop_duplicates(subset=['UserId', 'ProfileName' ,'Time' ,'Text'])
```

In [24]:
```
data.shape
```

Out[24]:
```
(393931, 10)
```

In [25]:
```
data.dtypes
```

```
Out[25]:  Id                          int64
          ProductId                  object
          UserId                     object
          ProfileName                object
          HelpfulnessNumerator        int64
          HelpfulnessDenominator      int64
          Score                       int64
          Time                        int64
          Summary                    object
          Text                       object
          dtype: object
```

```
In [26]:  # Converting 'Time' data type from int64 to date-time
```

```
In [27]:  data['Time'] = pd.to_datetime(data['Time'] , unit='s')
```

```
C:\Users\Ansh\AppData\Local\Temp\ipykernel_11960\2920101369.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#retu
rning-a-view-versus-a-copy
  data['Time'] = pd.to_datetime(data['Time'] , unit='s')
```

```
In [28]:  data['Time']
```

```
Out[28]:  0        2011-04-27
          1        2012-09-07
          2        2008-08-18
          3        2011-06-13
          4        2012-10-21
                      ...
          568449   2011-03-09
          568450   2012-03-09
          568451   2012-02-21
          568452   2012-03-13
          568453   2012-05-31
          Name: Time, Length: 393931, dtype: datetime64[ns]
```

3. Analysing to what Users Amazon can recommend more products

```
In [30]:  data.columns
```

```
Out[30]:  Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
                 'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
                dtype='object')
```

```
In [31]:  data['UserId'].nunique()
```

```
Out[31]:  256059
```

```
In [32]:  data.groupby(['UserId']).agg({'Summary':'count' , 'Text':'count' , 'Score':'mean' ,'ProductId':'count'  })
```

Out[32]:

	Summary	Text	Score	ProductId
UserId				
#oc-R103C0QSV1DF5E	1	1	5.000000	1
#oc-R109MU5OBBZ59U	1	1	5.000000	1
#oc-R10LFEMQEW6QGZ	1	1	5.000000	1
#oc-R10LT57ZGIB140	1	1	3.000000	1
#oc-R10UA029WVWIUI	1	1	1.000000	1
...	...	...	...	...
AZZV9PDNMCOZW	3	3	4.666667	3
AZZVNIMTTMJH6	1	1	5.000000	1
AZZY649VYAHQS	1	1	5.000000	1
AZZYCJOJLUDYR	1	1	5.000000	1
AZZZOVIBXHGDR	1	1	2.000000	1

256059 rows × 4 columns

```
In [33]:  req_df = data.groupby(['UserId']).agg({'Summary':'count' , 'Text':'count' , 'Score':'mean' ,'ProductId':'count'
```

```
In [34]:  req_df
```

Out[34]:

	Summary	Text	Score	ProductId
UserId				
AY12DBB0U420B	329	329	4.659574	329
A3OXHLG6DIBRW8	278	278	4.546763	278
A281NPSIMI1C2R	259	259	4.787645	259
A1YUL9PCJR3JTY	214	214	4.621495	214
A1Z54EM24Y40LL	211	211	4.383886	211
...	...	...	...	...
A2E80MDB9TCNGW	1	1	3.000000	1
A2E80RT3HOR35T	1	1	5.000000	1
A2E816C5N51F6X	1	1	5.000000	1
A2E81TVIUZI1IC	1	1	5.000000	1
AZZZOVIBXHGDR	1	1	2.000000	1

256059 rows × 4 columns

In [35]:
```
# Let's find out Top 10 users
```

In [36]:
```
req_df.index[0:10]
```

Out[36]:
```
Index(['AY12DBB0U420B', 'A3OXHLG6DIBRW8', 'A281NPSIMI1C2R', 'A1YUL9PCJR3JTY',
       'A1Z54EM24Y40LL', 'A2MUGFV2TDQ47K', 'A3D6OI36USYOU1', 'AZV26LP92E6WU',
       'AKMEY1BSHSDG7', 'A2GEZJHBV92EVR'],
      dtype='object', name='UserId')
```

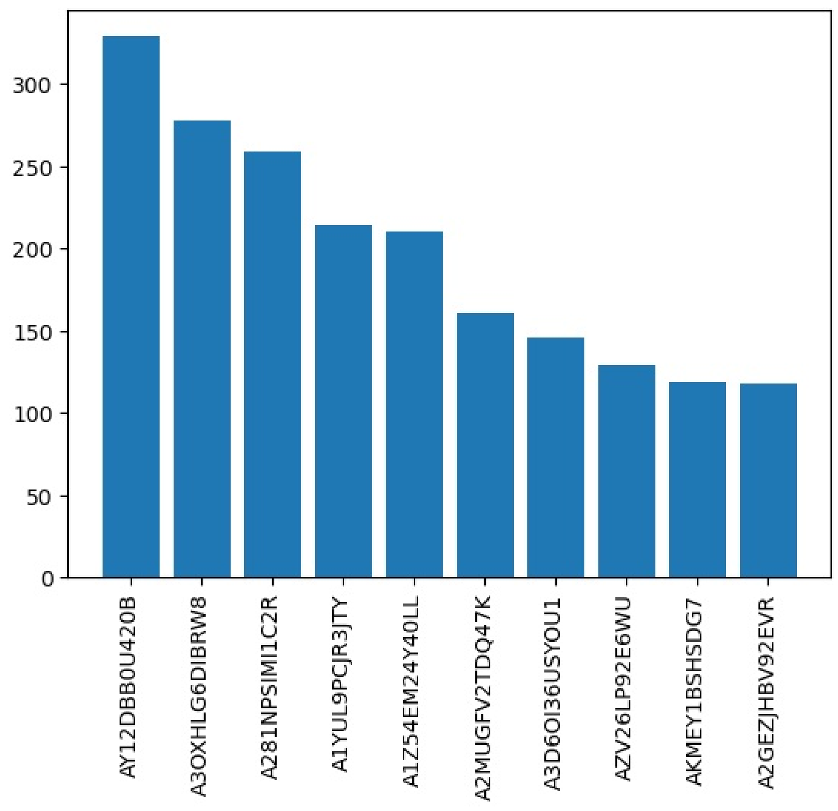
In [37]:
```
req_df.columns = ['Number_of_summaries' , 'num_text' , 'avg_score' , 'No_of_prods_purchased']
```

In [38]:
```
req_df['No_of_prods_purchased'][0:10].values
```

Out[38]:
```
array([329, 278, 259, 214, 211, 161, 146, 129, 119, 118], dtype=int64)
```

In [39]:
```
plt.bar(req_df.index[0:10] , req_df['No_of_prods_purchased'][0:10].values)
plt.xticks(rotation='vertical') #To avoid overlapping UserIds
```

Out[39]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [Text(0, 0, 'AY12DBB0U420B'),
  Text(1, 0, 'A3OXHLG6DIBRW8'),
  Text(2, 0, 'A281NPSIMI1C2R'),
  Text(3, 0, 'A1YUL9PCJR3JTY'),
  Text(4, 0, 'A1Z54EM24Y40LL'),
  Text(5, 0, 'A2MUGFV2TDQ47K'),
  Text(6, 0, 'A3D6OI36USYOU1'),
  Text(7, 0, 'AZV26LP92E6WU'),
  Text(8, 0, 'AKMEY1BSHSDG7'),
  Text(9, 0, 'A2GEZJHBV92EVR')])
```



Inference = These are the Top 10 users to whom we can recommend more products and there will be high probability that they will buy them

4. Finding which Products have good Reviews

In [42]:
```
data.columns
```

Out[42]:
```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

In [43]:
```
data['ProductId'].nunique()
```

Out[43]:
```
67624
```

In [44]:
```
#For Analysis, we will use most sold products
```

In [45]:
```
prod_count = data['ProductId'].value_counts().to_frame()
```

In [46]:
```
prod_count
```

Out[46]:

ProductId	count
B007JFMH8M	912
B002QWP89S	630
B003B3OOPA	622
B001EO5Q64	566
B0013NUGDE	558
...	...
B002DNX4GO	1
B000FM2YU2	1
B001M1VA32	1
B009858H6M	1
B001LR2CU2	1

67624 rows × 1 columns

In [47]:
```
prod_count[prod_count['count']>500]
```

Out[47]:

ProductId	count
B007JFMH8M	912
B002QWP89S	630
B003B3OOPA	622
B001EO5Q64	566
B0013NUGDE	558
B000KV61FC	556
B000UBD88A	542
B000NMJWZO	542
B005K4Q37A	541
B0090X8IPM	530
B005ZBZLT4	505

In [48]:
```
freq_ids=prod_count[prod_count['count']>500].index
```

In [49]:
```
freq_ids
```

Out[49]:
```
Index(['B007JFMH8M', 'B002QWP89S', 'B003B3OOPA', 'B001EO5Q64', 'B0013NUGDE',
       'B000KV61FC', 'B000UBD88A', 'B000NMJWZO', 'B005K4Q37A', 'B0090X8IPM',
       'B005ZBZLT4'],
      dtype='object', name='ProductId')
```

In [50]:
```
data['ProductId'].isin(freq_ids)
```

Out[50]:
```
0         False
1         False
2         False
3         False
4         False
          ...
568449    False
568450    False
568451    False
568452    False
568453    False
Name: ProductId, Length: 393931, dtype: bool
```

In [51]:
```
freq_prod_df=data[data['ProductId'].isin(freq_ids)]
```

In [52]:
```
freq_prod_df
```

Out[52]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Su
20982	20983	B002QWP89S	A21U4DR8M6I9QN	K. M Merrill "justine"	1	1	5	2011-10-18	a br c
20983	20984	B002QWP89S	A17TDUBB4Z1PEC	jaded_green	1	1	5	2011-10-14	C be
20984	20985	B002QWP89S	ABQH3WAWMSMBH	tenisbrat87	1	1	5	2011-09-28	
20985	20986	B002QWP89S	AVTY5M74VA1BJ	tarotqueen	1	1	5	2011-09-24	de g
20986	20987	B002QWP89S	A13TNN54ZEAUB1	dcz2221	1	1	5	2011-09-23	C
...	...	...	...	...	...	...	...	...	
563878	563879	B007JFMH8M	A366PSH7KFLRPB	TheRosySnail	0	0	5	2012-07-29	·
563879	563880	B007JFMH8M	A2KV6EYQPKJRR5	Kelley	0	0	5	2012-07-28	
563880	563881	B007JFMH8M	A3O7REI0OSV89M	Esme	0	0	4	2012-07-28	D
563881	563882	B007JFMH8M	A9JS5GQQ6GIQT	Syne	0	0	5	2012-07-28	
563882	563883	B007JFMH8M	AMAVEZAGCH52H	Tangela	0	0	5	2012-07-28	

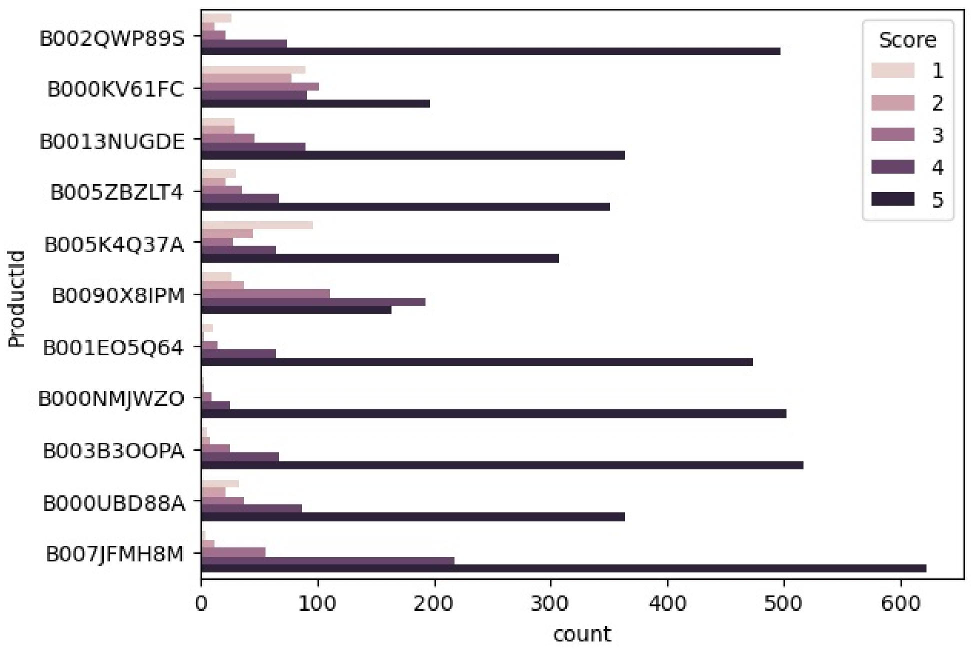
6504 rows × 10 columns

In [53]:
```
freq_prod_df.columns
```

Out[53]:
```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

In [54]:
```
sns.countplot(y = 'ProductId' , data = freq_prod_df , hue='Score')
```

Out[54]:
```
<Axes: xlabel='count', ylabel='ProductId'>
```



5. Difference between behaviour of Frequent and Non-Frequent Users

```
In [56]: #Frequent Users are those who have bought products 50 times or more
         #Not-Frequent users are those who have bought products less than 50 times
```

```
In [57]: data.columns
```

```
Out[57]: Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
                'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
               dtype='object')
```

```
In [58]: x = data['UserId'].value_counts()
```

```
In [59]: x
```

```
Out[59]: UserId
         AY12DBB0U420B     329
         A3OXHLG6DIBRW8    278
         A281NPSIMI1C2R    259
         A1YUL9PCJR3JTY    214
         A1Z54EM24Y40LL    211
                          ...
         AAQPR1MSRXKTU       1
         AG081Z6PZSF7P       1
         ALA84XWMTQBFT       1
         A1G9DK8EUR36JC      1
         A3LGQPJCZVL9UC      1
         Name: count, Length: 256059, dtype: int64
```

```
In [60]: data.head()
```

Out[60]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	2011-04-27	Good Quality Dog Food	s
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	2012-09-07	Not as Advertised	la F
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	2008-08-18	"Delight" says it all	cc a
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	2011-06-13	Cough Medicine	If in
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	2012-10-21	Great taffy	t

In [61]:
```
x['AY12DBB0U420B']
```

Out[61]:
```
329
```

In [62]:
```
data['user_type'] = data['UserId'].apply(lambda user : "Frequent" if x[user]>50 else "Not Frequent")
```

```
C:\Users\Ansh\AppData\Local\Temp\ipykernel_11960\3239024313.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#retu
rning-a-view-versus-a-copy
  data['user_type'] = data['UserId'].apply(lambda user : "Frequent" if x[user]>50 else "Not Frequent")
```

In [63]:
```
data
```

Out[63]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	2011-04-27	Q
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	2012-09-07	A
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	2008-08-18	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	2011-06-13	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	2012-10-21	(
...	...	...	...	...	...	...	...	...	
568449	568450	B001EO7N10	A28KG5XORO54AY	Lettie D. Carter	0	0	5	2011-03-09	\
568450	568451	B003S1WTCU	A3I8AFVPEE8KI5	R. Sawyer	0	0	2	2012-03-09	dis
568451	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	2	2	5	2012-02-21	I ou
568452	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	1	1	5	2012-03-13	Tra re
568453	568454	B001LR2CU2	A3LGQPJCZVL9UC	srfell17	0	0	5	2012-05-31	

393931 rows × 11 columns

In [64]:
```
not_freq_df = data[data['user_type']=='Not Frequent']
freq_df = data[data['user_type']=='Frequent']
```

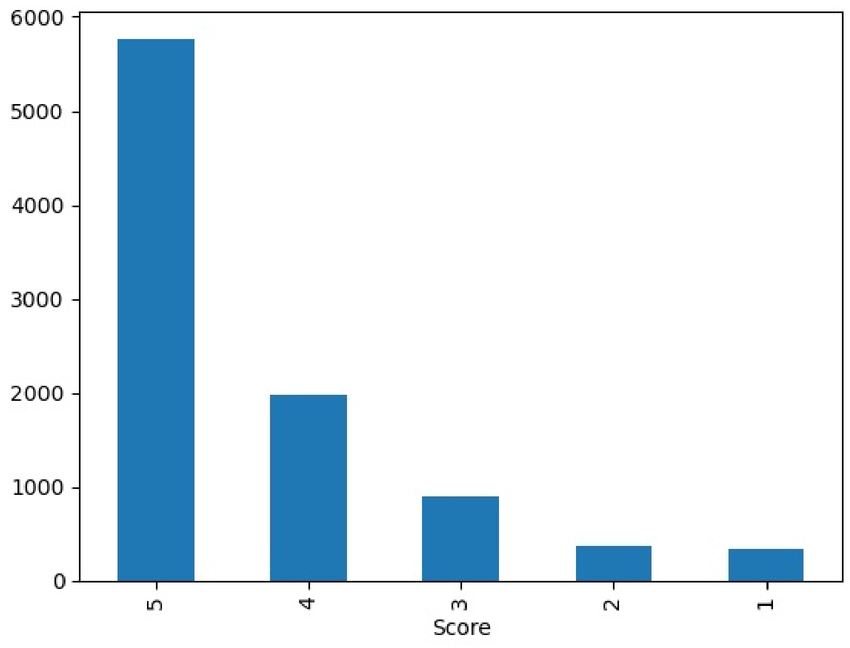
In [65]:
```
freq_df['Score'].value_counts()
```

Out[65]:
```
Score
5    5765
4    1979
3     897
2     368
1     349
Name: count, dtype: int64
```

In [66]:
```
#PLotting above results using bar-plot
```

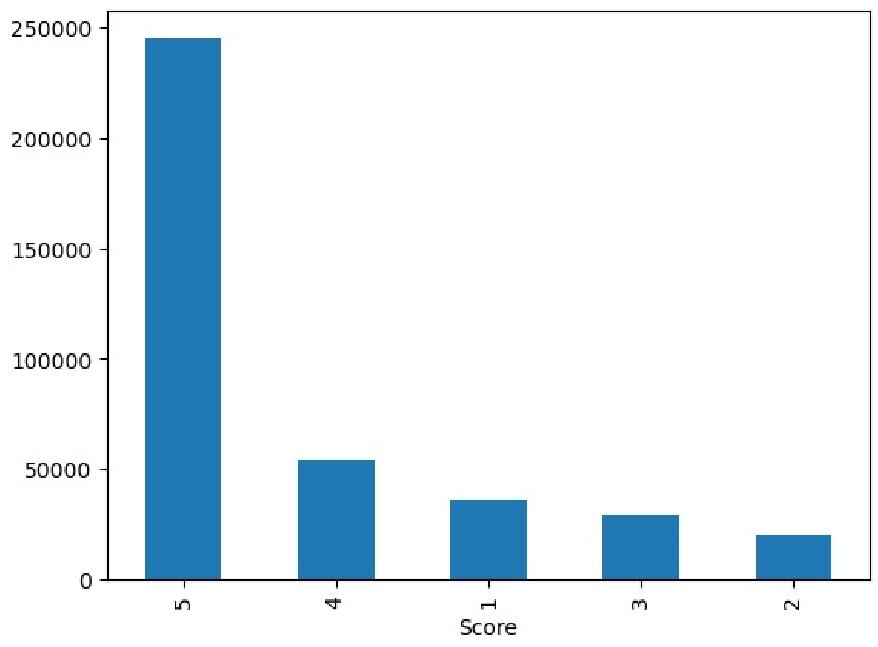
In [67]:
```
freq_df['Score'].value_counts().plot(kind='bar')
```

Out[67]:
```
<Axes: xlabel='Score'>
```



```
In [68]: not_freq_df['Score'].value_counts().plot(kind='bar')
```

```
Out[68]: <Axes: xlabel='Score'>
```



Inference = The distribution of ratings among frequent reviewers is similar to that of all reviews. However, we can see that frequent reviewers give less 5-star reviews and less 1-star review.

6. Finding out if frequent Users are verbose

```
In [71]: data.columns
```

```
Out[71]: Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
                'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text',
                'user_type'],
               dtype='object')
```

```
In [72]: data[['UserId' , 'ProductId' , 'Text']]
```

Out[72]:

	UserId	ProductId	Text
0	A3SGXH7AUHU8GW	B001E4KFG0	I have bought several of the Vitality canned d...
1	A1D87F6ZCVE5NK	B00813GRG4	Product arrived labeled as Jumbo Salted Peanut...
2	ABXLMWJIXXAIN	B000LQOCH0	This is a confection that has been around a fe...
3	A395BORC6FGVXV	B000UA0QIQ	If you are looking for the secret ingredient i...
4	A1UQRSCLF8GW1T	B006K2ZZ7K	Great taffy at a great price. There was a wid...
...	...	...	...
568449	A28KG5XORO54AY	B001EO7N10	Great for sesame chicken..this is a good if no...
568450	A3I8AFVPEE8KI5	B003S1WTCU	I'm disappointed with the flavor. The chocolat...
568451	A121AA1GQV751Z	B004I613EE	These stars are small, so you can give 10-15 o...
568452	A3IBEVCTXKNOH	B004I613EE	These are the BEST treats for training and rew...
568453	A3LGQPJCZVL9UC	B001LR2CU2	I am very satisfied ,product is as advertised,...

393931 rows × 3 columns

In [73]:
```
data['Text'][0]
```

Out[73]:
```
'I have bought several of the Vitality canned dog food products and have found them all to be of good quality. 
The product looks more like a stew than a processed meat and it smells better. My Labrador is finicky and she a
ppreciates this product better than  most.'
```

In [74]:
```
len(data['Text'][0].split(' '))
```

Out[74]:
```
49
```

In [75]:
```
def calculate_length(text):
    return len(text.split(' '))
```

In [76]:
```
data['Text_length'] = data['Text'].apply(calculate_length)
```

```
C:\Users\Ansh\AppData\Local\Temp\ipykernel_11960\1567403007.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#retu
rning-a-view-versus-a-copy
  data['Text_length'] = data['Text'].apply(calculate_length)
```

In [77]:
```
## let's separate dataframe for both "frequent_viewers" & for "not_frequent_viewers"
```

In [78]:
```
data['user_type'].unique()
```

Out[78]:
```
array(['Not Frequent', 'Frequent'], dtype=object)
```

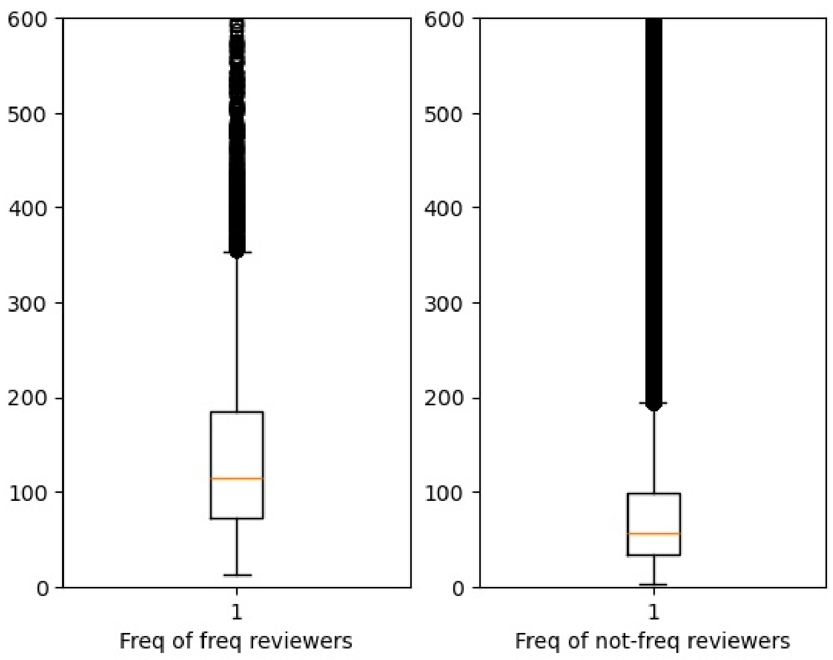
In [79]:
```
not_freq_data = data[data['user_type']=='Not Frequent']
freq_data = data[data['user_type']=='Frequent']
```

In [80]:
```
fig = plt.figure()

ax1 = fig.add_subplot(121)
ax1.boxplot(freq_data['Text_length'])
ax1.set_xlabel('Freq of freq reviewers')
ax1.set_ylim(0,600) ## setting limit on y-axis..

ax2 = fig.add_subplot(122)
ax2.boxplot(not_freq_data['Text_length'])
ax2.set_xlabel('Freq of not-freq reviewers')
ax2.set_ylim(0,600)
```

Out[80]:
```
(0.0, 600.0)
```



Inference = The distributions of word counts for frequent and non-frequent reviews shows that non-frequent reviewers have a large amount of reviews of low word count. On the other hand, the largest concentration of word count is higher for frequent reviewers than for non-frequent reviews.

7. Sentiment Analysis

```
In [156…  #Sentiment Analysis is the computational task of automatically determining what feelings a writer is expressing
```

```
In [160…  from textblob import TextBlob
```

```
In [162…  data['Summary'][0]
Out[162…  'Good Quality Dog Food'
```

```
In [164…  TextBlob('Good Quality Dog Food').sentiment.polarity
Out[164…  0.7
```

```
In [166…  sample = data[0:50000]
```

```
In [168…  polarity = []

          for text in sample['Summary']:
              try:
                  polarity.append(TextBlob(text).sentiment.polarity)
              except:
                  polarity.append(0)
```

```
In [170…  len(polarity)
Out[170…  50000
```

```
In [172…  sample['polarity'] = polarity
```

```
C:\Users\Ansh\AppData\Local\Temp\ipykernel_11960\4182253960.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#retu
rning-a-view-versus-a-copy
  sample['polarity'] = polarity
```

In [174…

```python
sample.head()
```

Out[174…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	2011-04-27	Good Quality Dog Food	s
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	2012-09-07	Not as Advertised	la F
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	2008-08-18	"Delight" says it all	cc a
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	2011-06-13	Cough Medicine	If in
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	2012-10-21	Great taffy	t

In [176…

```python
s_negative= sample[sample['polarity']<0]

s_positive= sample[sample['polarity']>0]
```

In [178…

```python
from collections import Counter
```

In [182…

```python
Counter(s_negative['Summary']).most_common(10) # Most used negative keywords
```

Out[182…

```
[('Disappointed', 44),
 ('Disappointing', 32),
 ('Bland', 18),
 ('Awful', 17),
 ('Not what I expected', 17),
 ('Terrible', 15),
 ('Horrible', 15),
 ('disappointed', 15),
 ('Disgusting', 12),
 ('not good', 11)]
```

In [184…

```python
Counter(s_positive['Summary']).most_common(10) # Most used positive keywords
```

Out[184…

```
[('Delicious!', 208),
 ('Delicious', 204),
 ('Great product', 100),
 ('Excellent', 85),
 ('Love it!', 81),
 ('Great', 81),
 ('Great Product', 77),
 ('Great!', 70),
 ('Good stuff', 51),
 ('Awesome', 50)]
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js