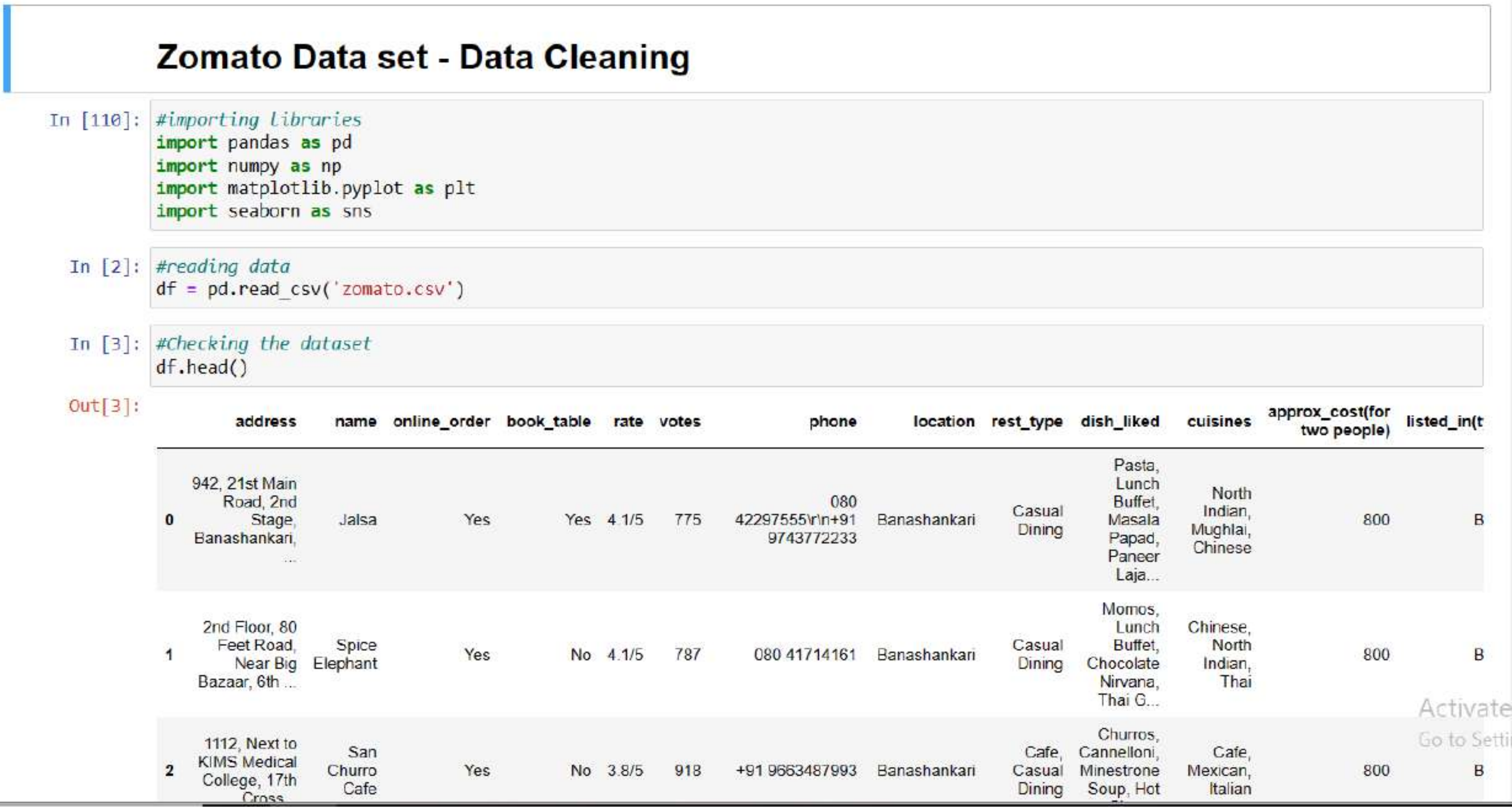


Zomato Data set - Data Cleaning

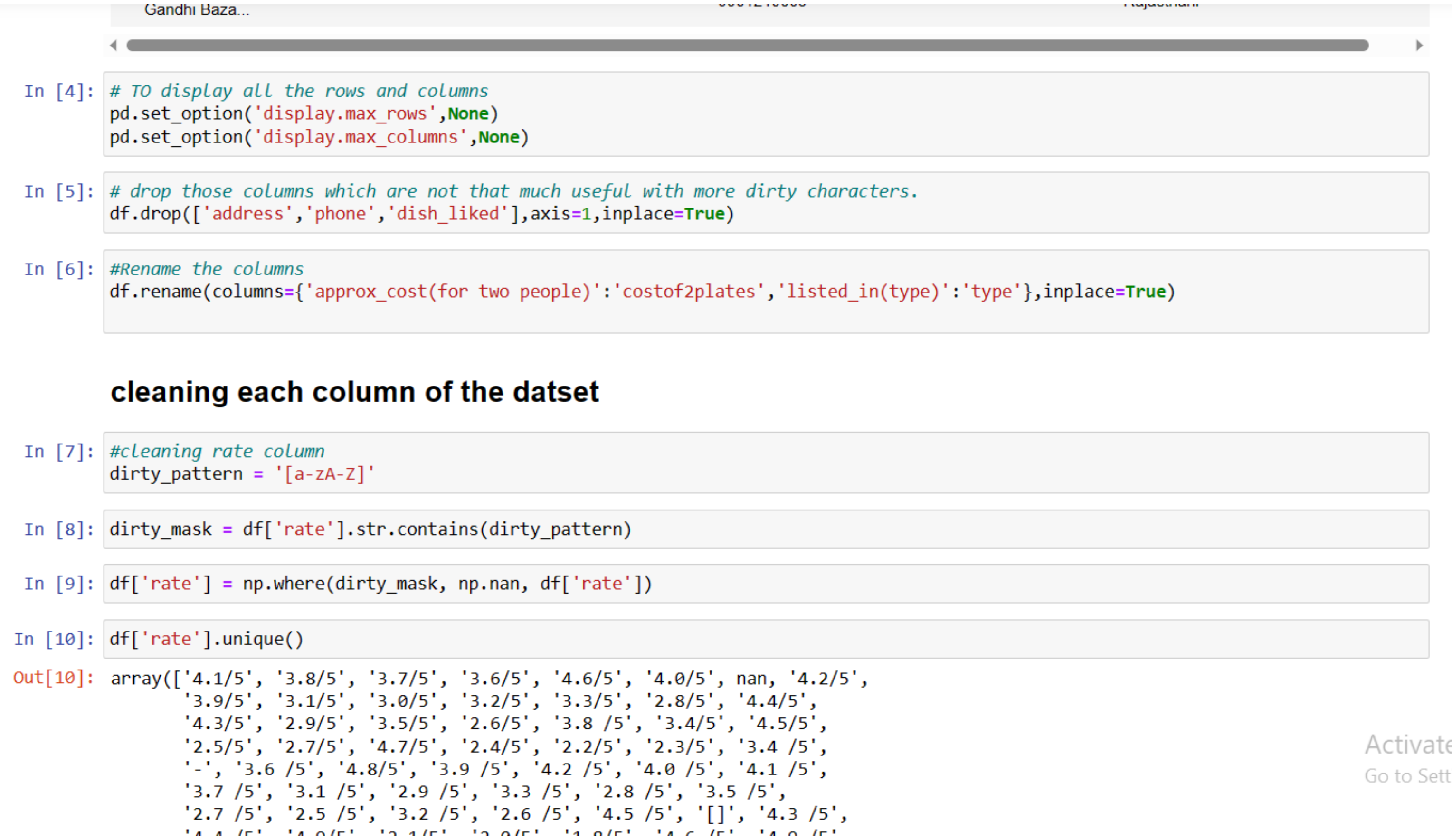
In [110]:
```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#reading data
df = pd.read_csv('zomato.csv')
```

In [3]:
```
#Checking the dataset
df.head()
```

Out[3]:

	address	name	online_order	book_table	rate	votes	phone	location	rest_type	dish_liked	cuisines	approx_cost(for two people)	listed_in(t
0	942, 21st Main Road, 2nd Stage, Banashankari, ...	Jalsa	Yes	Yes	4.1/5	775	080 42297555\r\n+91 9743772233	Banashankari	Casual Dining	Pasta, Lunch Buffet, Masala Papad, Paneer Laja...	North Indian, Mughlai, Chinese	800	B
1	2nd Floor, 80 Feet Road, Near Big Bazaar, 6th ...	Spice Elephant	Yes	No	4.1/5	787	080 41714161	Banashankari	Casual Dining	Momos, Lunch Buffet, Chocolate Nirvana, Thai G...	Chinese, North Indian, Thai	800	B
2	1112, Next to KIMS Medical College, 17th Cross	San Churro Cafe	Yes	No	3.8/5	918	+91 9663487993	Banashankari	Cafe, Casual Dining	Churros, Cannelloni, Minestrone Soup, Hot	Cafe, Mexican, Italian	800	B



Gandhi Baza...

```
In [4]: # TO display all the rows and columns
        pd.set_option('display.max_rows',None)
        pd.set_option('display.max_columns',None)
```

```
In [5]: # drop those columns which are not that much useful with more dirty characters.
        df.drop(['address','phone','dish_liked'],axis=1,inplace=True)
```

```
In [6]: #Rename the columns
        df.rename(columns={'approx_cost(for two people)':'costof2plates','listed_in(type)':'type'},inplace=True)
```

cleaning each column of the datset

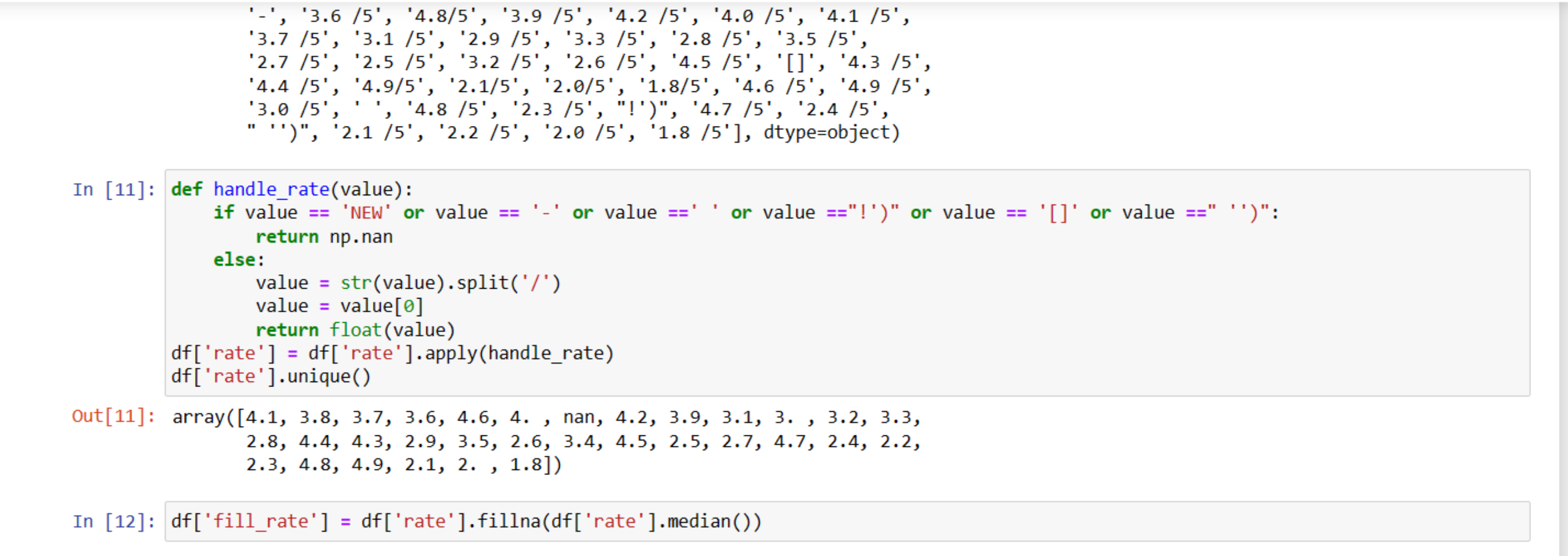
```
In [7]: #cleaning rate column
        dirty_pattern = '[a-zA-Z]'
```

```
In [8]: dirty_mask = df['rate'].str.contains(dirty_pattern)
```

```
In [9]: df['rate'] = np.where(dirty_mask, np.nan, df['rate'])
```

```
In [10]: df['rate'].unique()
```

```
Out[10]: array(['4.1/5', '3.8/5', '3.7/5', '3.6/5', '4.6/5', '4.0/5', nan, '4.2/5',
                '3.9/5', '3.1/5', '3.0/5', '3.2/5', '3.3/5', '2.8/5', '4.4/5',
                '4.3/5', '2.9/5', '3.5/5', '2.6/5', '3.8 /5', '3.4/5', '4.5/5',
                '2.5/5', '2.7/5', '4.7/5', '2.4/5', '2.2/5', '2.3/5', '3.4 /5',
                '-', '3.6 /5', '4.8/5', '3.9 /5', '4.2 /5', '4.0 /5', '4.1 /5',
                '3.7 /5', '3.1 /5', '2.9 /5', '3.3 /5', '2.8 /5', '3.5 /5',
                '2.7 /5', '2.5 /5', '3.2 /5', '2.6 /5', '4.5 /5', '[]', '4.3 /5',
                '4.4 /5', '4.0/5', '2.1/5', '2.0/5', '1.8/5', '4.6 /5', '4.9 /5',
```

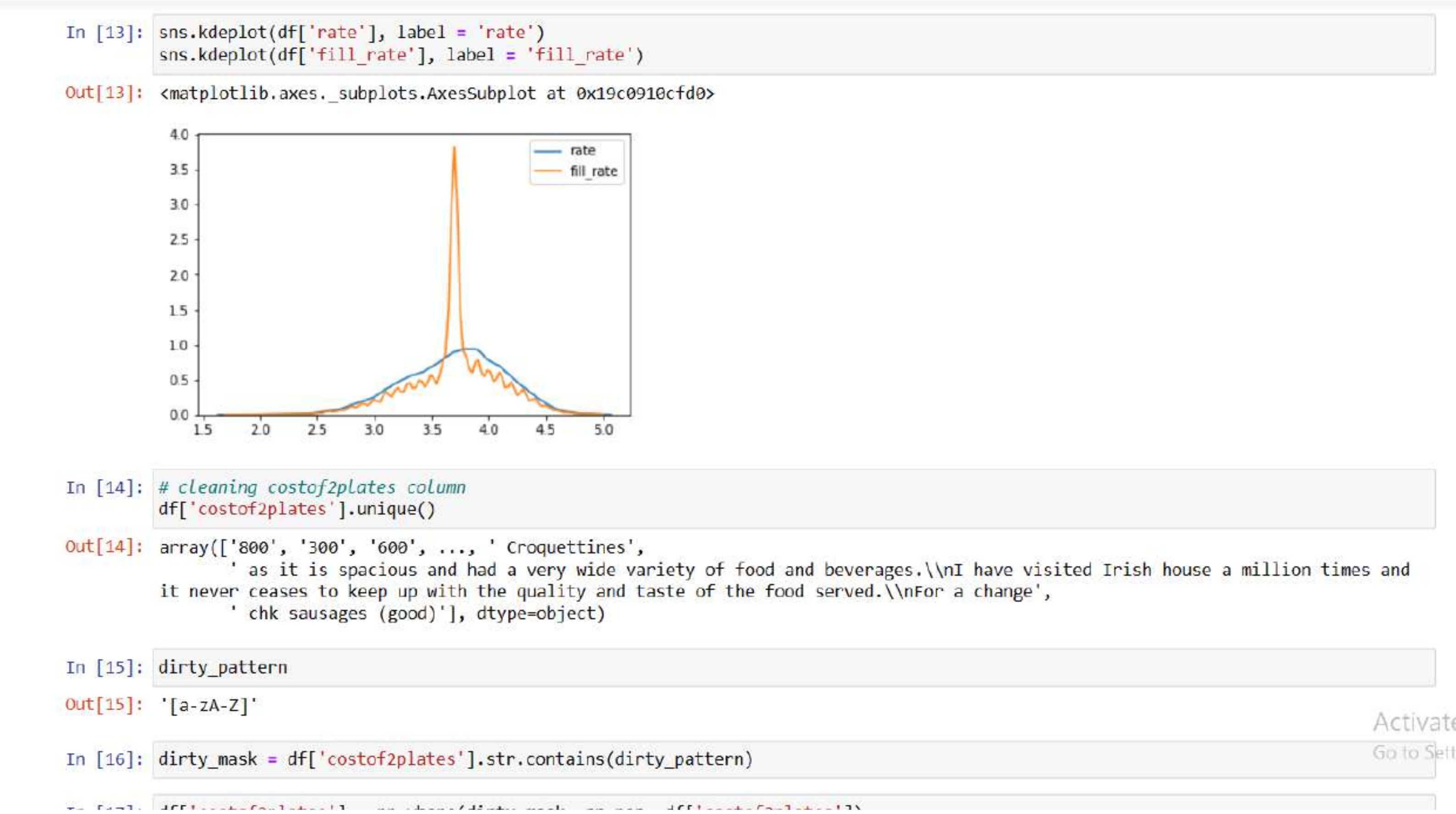


```
       '-', '3.6 /5', '4.8/5', '3.9 /5', '4.2 /5', '4.0 /5', '4.1 /5',
       '3.7 /5', '3.1 /5', '2.9 /5', '3.3 /5', '2.8 /5', '3.5 /5',
       '2.7 /5', '2.5 /5', '3.2 /5', '2.6 /5', '4.5 /5', '[]', '4.3 /5',
       '4.4 /5', '4.9/5', '2.1/5', '2.0/5', '1.8/5', '4.6 /5', '4.9 /5',
       '3.0 /5', ' ', '4.8 /5', '2.3 /5', "!')", '4.7 /5', '2.4 /5',
       " '')", '2.1 /5', '2.2 /5', '2.0 /5', '1.8 /5'], dtype=object)
```

```
In [11]: def handle_rate(value):
             if value == 'NEW' or value == '-' or value ==' ' or value =="!')" or value == '[]' or value ==" '')":
                 return np.nan
             else:
                 value = str(value).split('/')
                 value = value[0]
                 return float(value)
         df['rate'] = df['rate'].apply(handle_rate)
         df['rate'].unique()
```

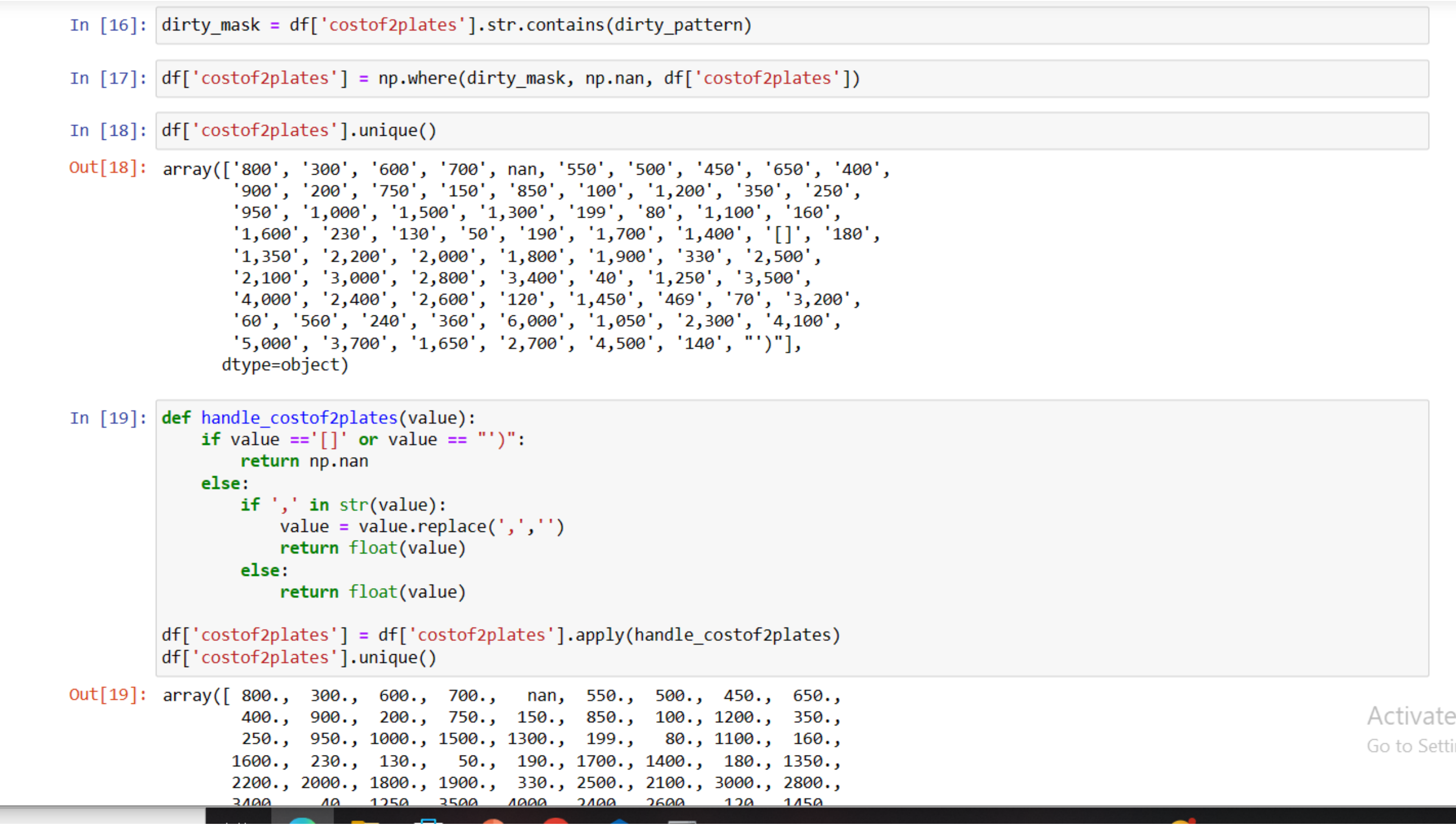
```
Out[11]: array([4.1, 3.8, 3.7, 3.6, 4.6, 4. , nan, 4.2, 3.9, 3.1, 3. , 3.2, 3.3,
                2.8, 4.4, 4.3, 2.9, 3.5, 2.6, 3.4, 4.5, 2.5, 2.7, 4.7, 2.4, 2.2,
                2.3, 4.8, 4.9, 2.1, 2. , 1.8])
```

```
In [12]: df['fill_rate'] = df['rate'].fillna(df['rate'].median())
```



```
In [13]: sns.kdeplot(df['rate'], label = 'rate')
         sns.kdeplot(df['fill_rate'], label = 'fill_rate')

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x19c0910cfd0>
```

```
In [14]: # cleaning costof2plates column
         df['costof2plates'].unique()

Out[14]: array(['800', '300', '600', ..., ' Croquettines',
                ' as it is spacious and had a very wide variety of food and beverages.\\nI have visited Irish house a million times and
         it never ceases to keep up with the quality and taste of the food served.\\nFor a change',
                ' chk sausages (good)'], dtype=object)

In [15]: dirty_pattern

Out[15]: '[a-zA-Z]'

In [16]: dirty_mask = df['costof2plates'].str.contains(dirty_pattern)

In [17]: df['costof2plates'] = np.where(dirty_mask, np.nan, df['costof2plates'])
```



```
In [16]: dirty_mask = df['costof2plates'].str.contains(dirty_pattern)
```

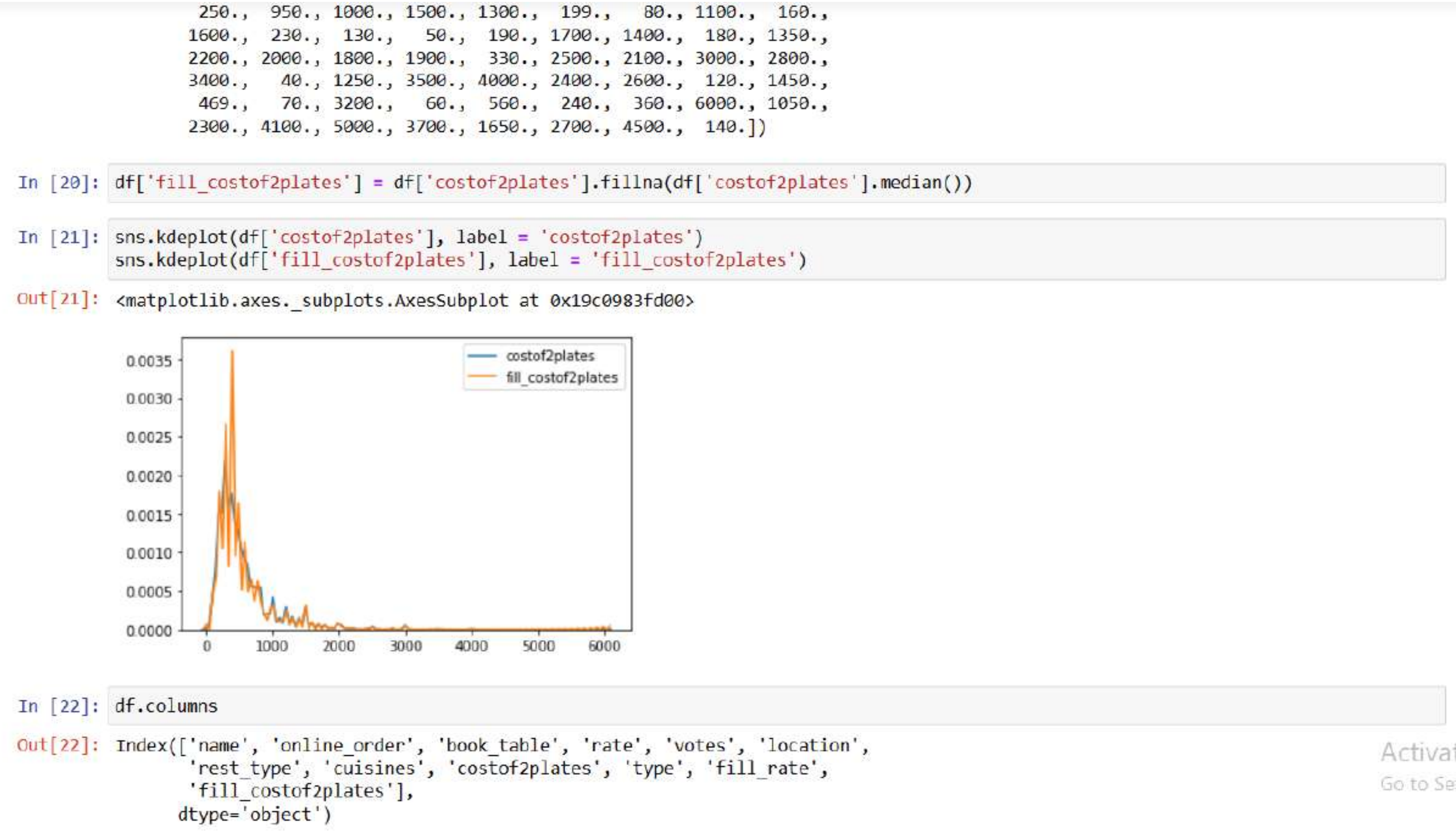
```
In [17]: df['costof2plates'] = np.where(dirty_mask, np.nan, df['costof2plates'])
```

```
In [18]: df['costof2plates'].unique()
```

```
Out[18]: array(['800', '300', '600', '700', nan, '550', '500', '450', '650', '400',
                '900', '200', '750', '150', '850', '100', '1,200', '350', '250',
                '950', '1,000', '1,500', '1,300', '199', '80', '1,100', '160',
                '1,600', '230', '130', '50', '190', '1,700', '1,400', '[]', '180',
                '1,350', '2,200', '2,000', '1,800', '1,900', '330', '2,500',
                '2,100', '3,000', '2,800', '3,400', '40', '1,250', '3,500',
                '4,000', '2,400', '2,600', '120', '1,450', '469', '70', '3,200',
                '60', '560', '240', '360', '6,000', '1,050', '2,300', '4,100',
                '5,000', '3,700', '1,650', '2,700', '4,500', '140', "')"],
               dtype=object)
```

```
In [19]: def handle_costof2plates(value):
             if value =='[]' or value == "')":
                 return np.nan
             else:
                 if ',' in str(value):
                     value = value.replace(',','')
                     return float(value)
                 else:
                     return float(value)

         df['costof2plates'] = df['costof2plates'].apply(handle_costof2plates)
         df['costof2plates'].unique()
```

```
Out[19]: array([ 800.,  300.,  600.,  700.,   nan,  550.,  500.,  450.,  650.,
                 400.,  900.,  200.,  750.,  150.,  850.,  100., 1200.,  350.,
                 250.,  950., 1000., 1500., 1300.,  199.,   80., 1100.,  160.,
                1600.,  230.,  130.,   50.,  190., 1700., 1400.,  180., 1350.,
                2200., 2000., 1800., 1900.,  330., 2500., 2100., 3000., 2800.,
                3400.,   40., 1250., 3500., 4000., 2400., 2600.,  120., 1450.
```



```
         250.,  950., 1000., 1500., 1300.,  199.,   80., 1100.,  160.,
        1600.,  230.,  130.,   50.,  190., 1700., 1400.,  180., 1350.,
        2200., 2000., 1800., 1900.,  330., 2500., 2100., 3000., 2800.,
        3400.,   40., 1250., 3500., 4000., 2400., 2600.,  120., 1450.,
         469.,   70., 3200.,   60.,  560.,  240.,  360., 6000., 1050.,
        2300., 4100., 5000., 3700., 1650., 2700., 4500.,  140.])
```

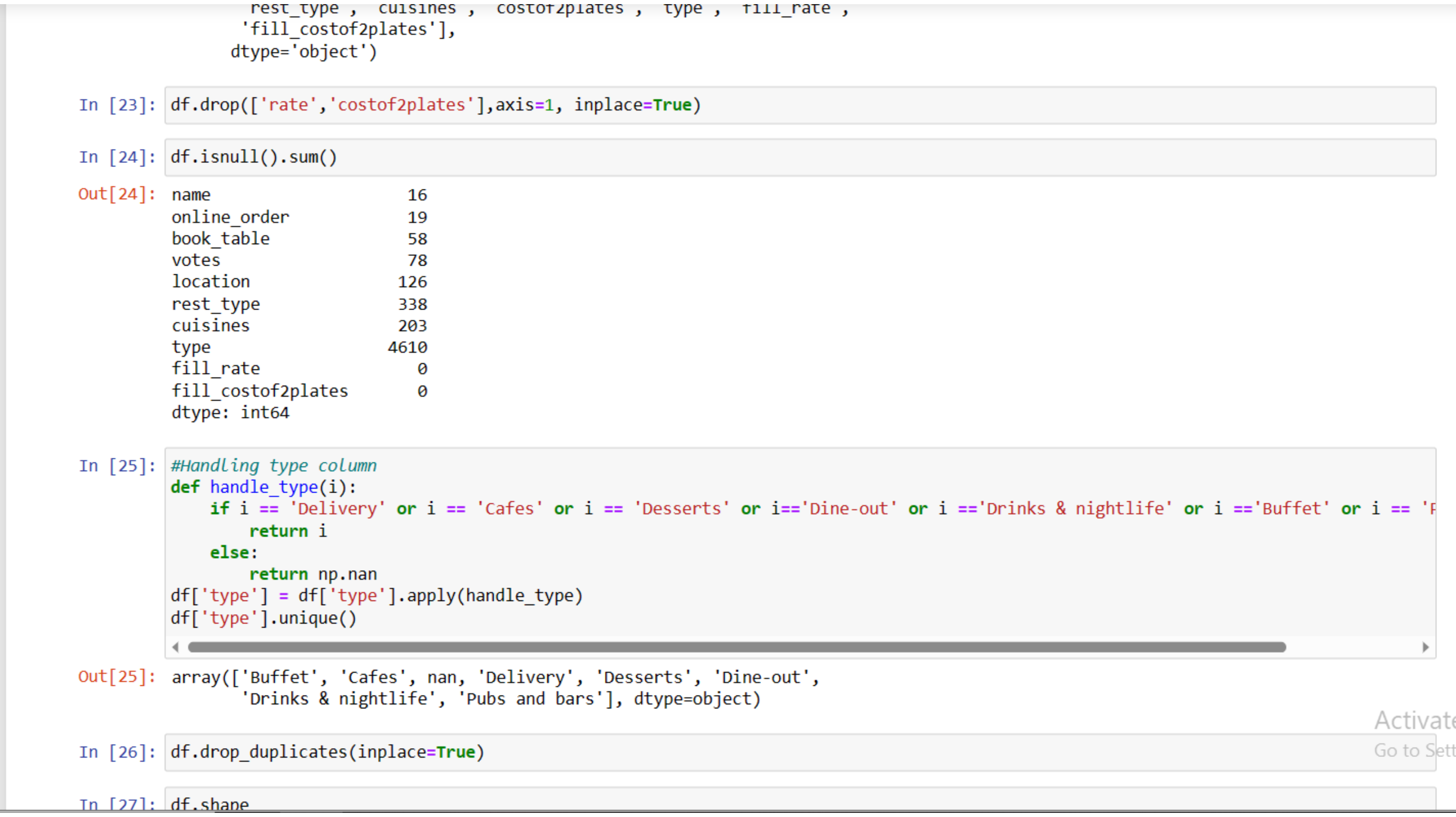
```
In [20]: df['fill_costof2plates'] = df['costof2plates'].fillna(df['costof2plates'].median())
```

```
In [21]: sns.kdeplot(df['costof2plates'], label = 'costof2plates')
         sns.kdeplot(df['fill_costof2plates'], label = 'fill_costof2plates')
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x19c0983fd00>
```

```
In [22]: df.columns
```

```
Out[22]: Index(['name', 'online_order', 'book_table', 'rate', 'votes', 'location',
                'rest_type', 'cuisines', 'costof2plates', 'type', 'fill_rate',
                'fill_costof2plates'],
               dtype='object')
```



```
        'rest_type', 'cuisines', 'costof2plates', 'type', 'fill_rate',
        'fill_costof2plates'],
       dtype='object')
```

In [23]:
```
df.drop(['rate','costof2plates'],axis=1, inplace=True)
```

In [24]:
```
df.isnull().sum()
```

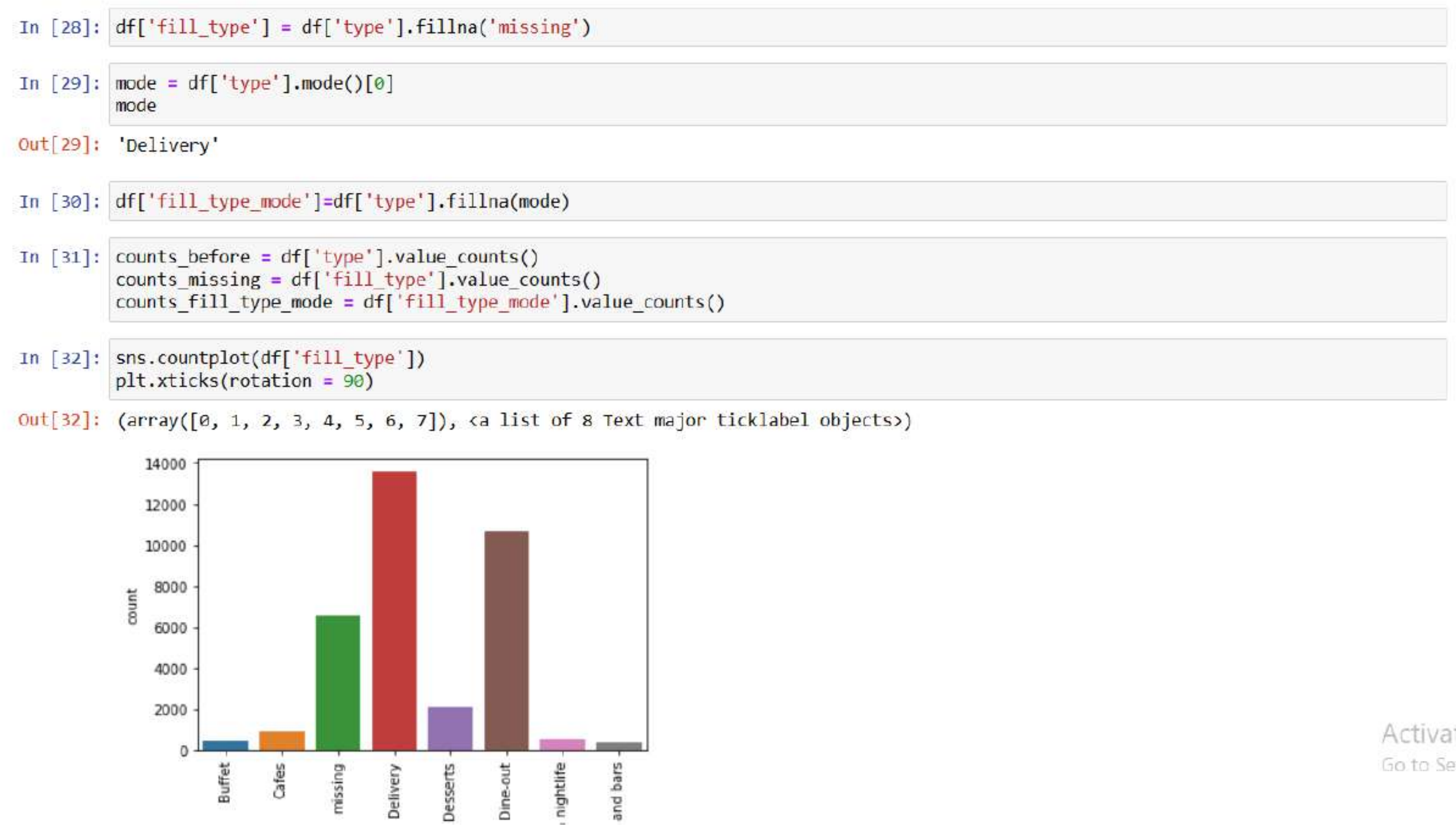
Out[24]:
```
name                    16
online_order            19
book_table              58
votes                   78
location               126
rest_type              338
cuisines               203
type                  4610
fill_rate                0
fill_costof2plates       0
dtype: int64
```

In [25]:
```
#Handling type column
def handle_type(i):
    if i == 'Delivery' or i == 'Cafes' or i == 'Desserts' or i=='Dine-out' or i =='Drinks & nightlife' or i =='Buffet' or i == 'P
        return i
    else:
        return np.nan
df['type'] = df['type'].apply(handle_type)
df['type'].unique()
```

Out[25]:
```
array(['Buffet', 'Cafes', nan, 'Delivery', 'Desserts', 'Dine-out',
       'Drinks & nightlife', 'Pubs and bars'], dtype=object)
```

In [26]:
```
df.drop_duplicates(inplace=True)
```

In [27]:
```
df.shape
```



```
In [28]: df['fill_type'] = df['type'].fillna('missing')
```

```
In [29]: mode = df['type'].mode()[0]
         mode
```

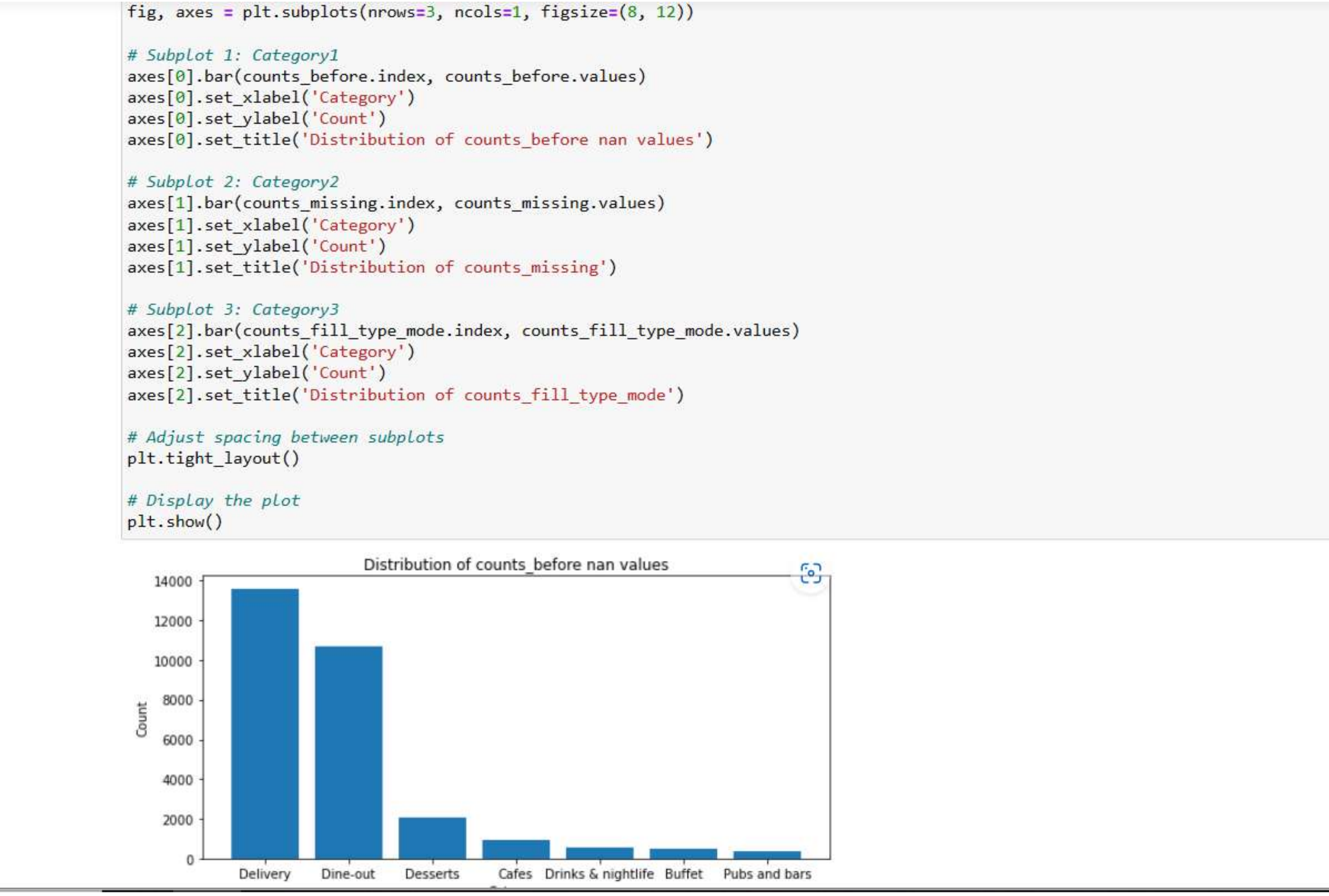
```
Out[29]: 'Delivery'
```

```
In [30]: df['fill_type_mode']=df['type'].fillna(mode)
```

```
In [31]: counts_before = df['type'].value_counts()
         counts_missing = df['fill_type'].value_counts()
         counts_fill_type_mode = df['fill_type_mode'].value_counts()
```

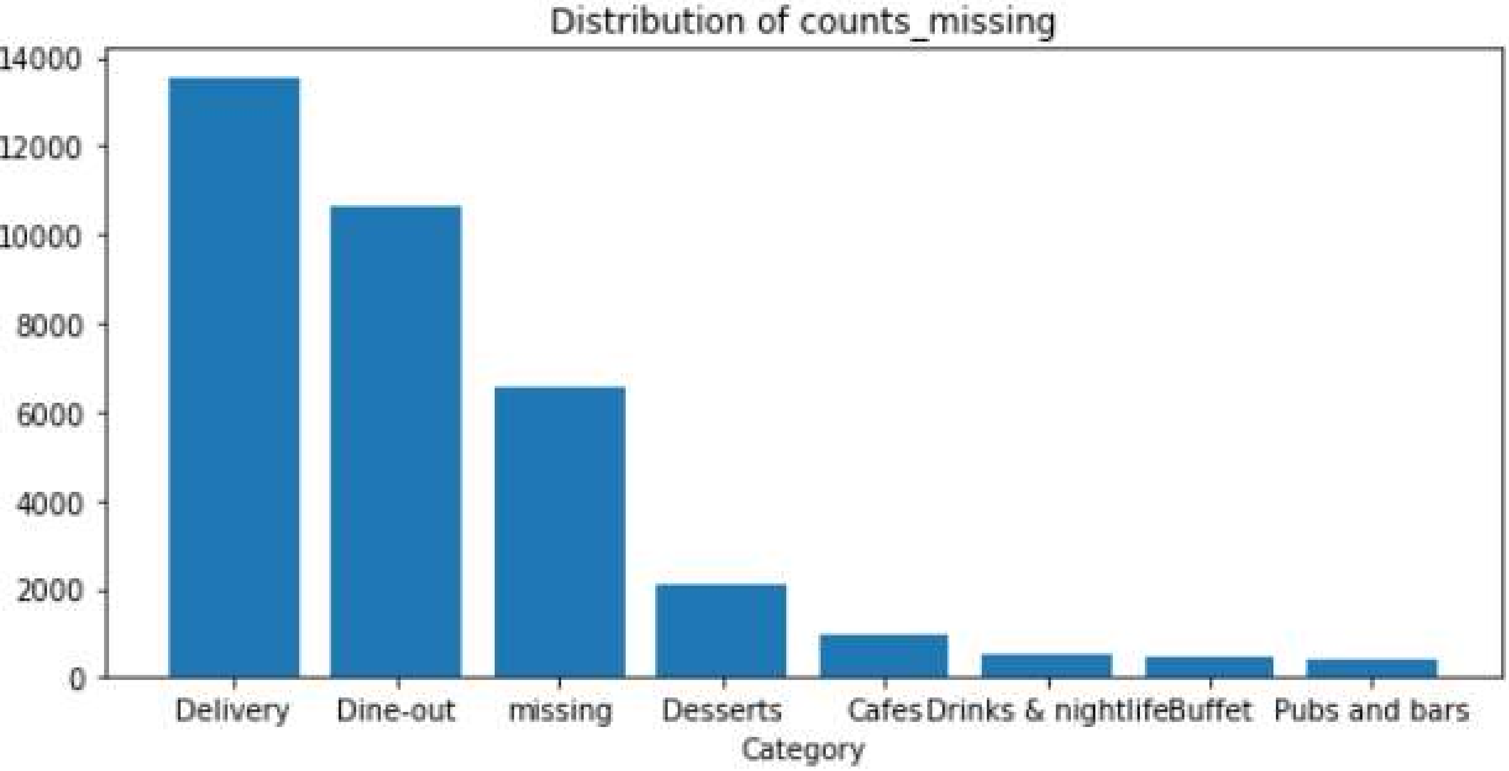
```
In [32]: sns.countplot(df['fill_type'])
         plt.xticks(rotation = 90)
```

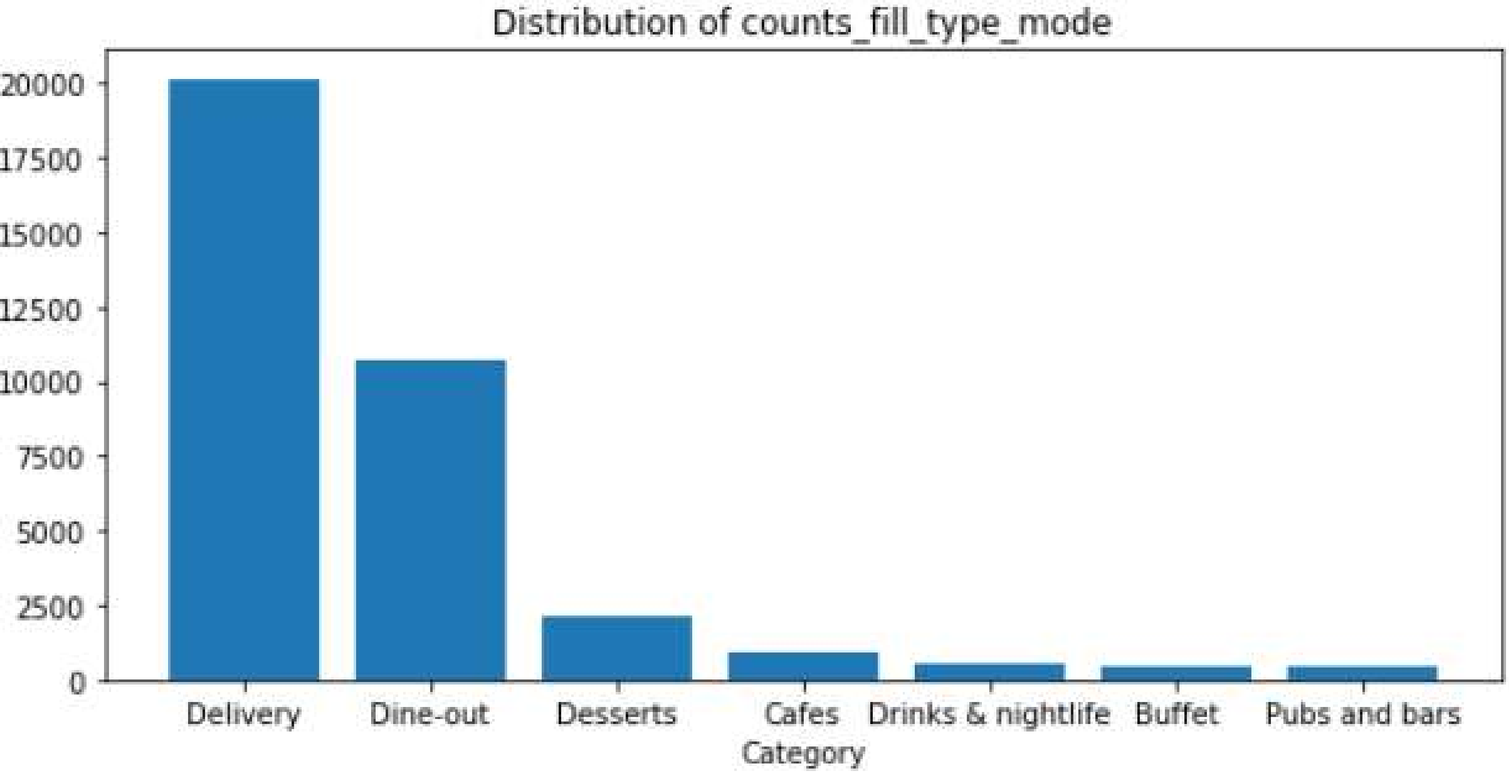
```
Out[32]: (array([0, 1, 2, 3, 4, 5, 6, 7]), <a list of 8 Text major ticklabel objects>)
```



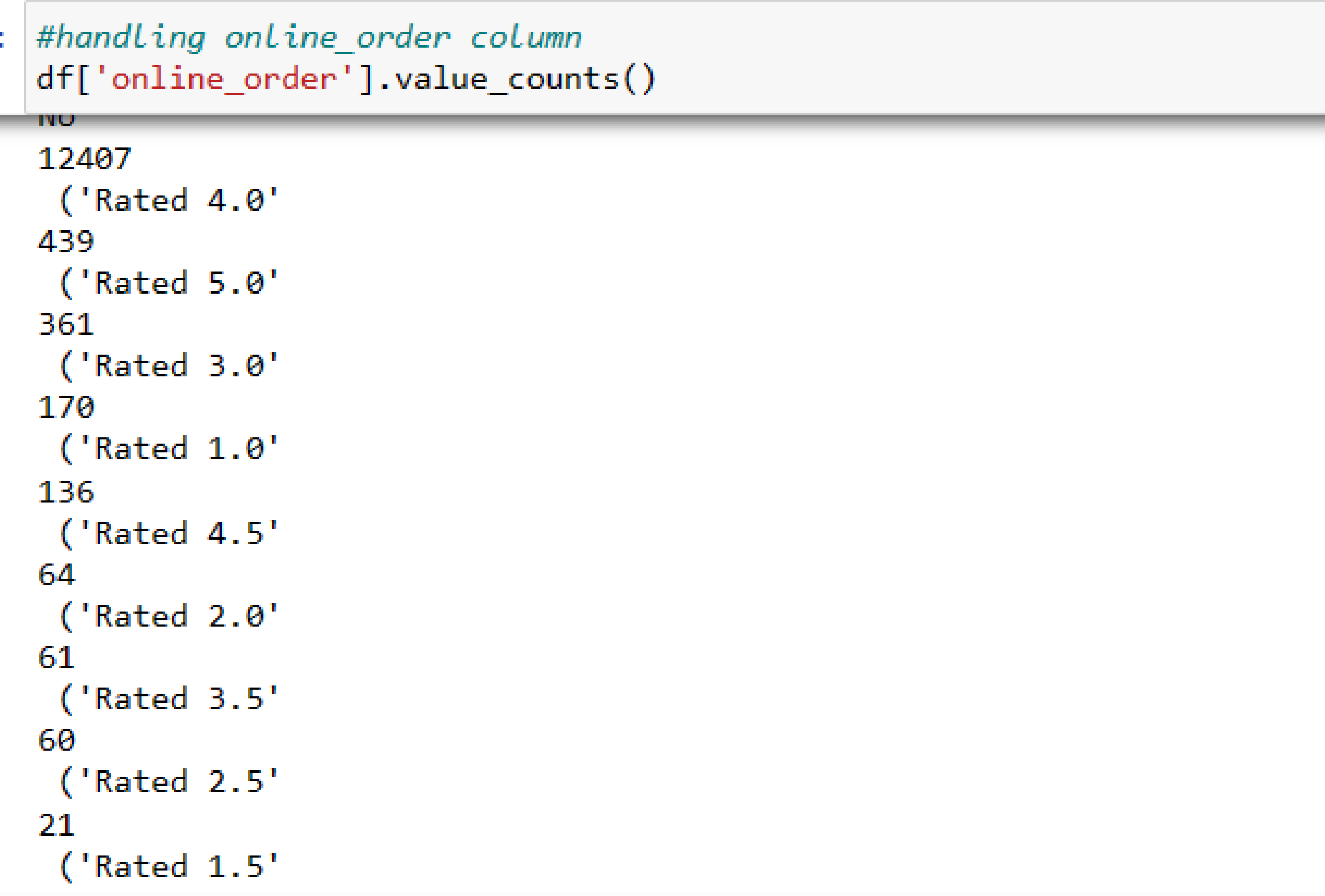
```
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(8, 12))

# Subplot 1: Category1
axes[0].bar(counts_before.index, counts_before.values)
axes[0].set_xlabel('Category')
axes[0].set_ylabel('Count')
axes[0].set_title('Distribution of counts_before nan values')

# Subplot 2: Category2
axes[1].bar(counts_missing.index, counts_missing.values)
axes[1].set_xlabel('Category')
axes[1].set_ylabel('Count')
axes[1].set_title('Distribution of counts_missing')

# Subplot 3: Category3
axes[2].bar(counts_fill_type_mode.index, counts_fill_type_mode.values)
axes[2].set_xlabel('Category')
axes[2].set_ylabel('Count')
axes[2].set_title('Distribution of counts_fill_type_mode')

# Adjust spacing between subplots
plt.tight_layout()

# Display the plot
plt.show()
```

Delivery
Dine-out
Desserts
Cafes
Drinks & nightlife
Buffet
Pubs and bars
Category


Distribution of counts_missing
14000
12000
10000
8000
6000
4000
2000
0
Delivery
Dine-out
missing
Desserts
Cafes
Drinks & nightlife
Buffet
Pubs and bars
Category

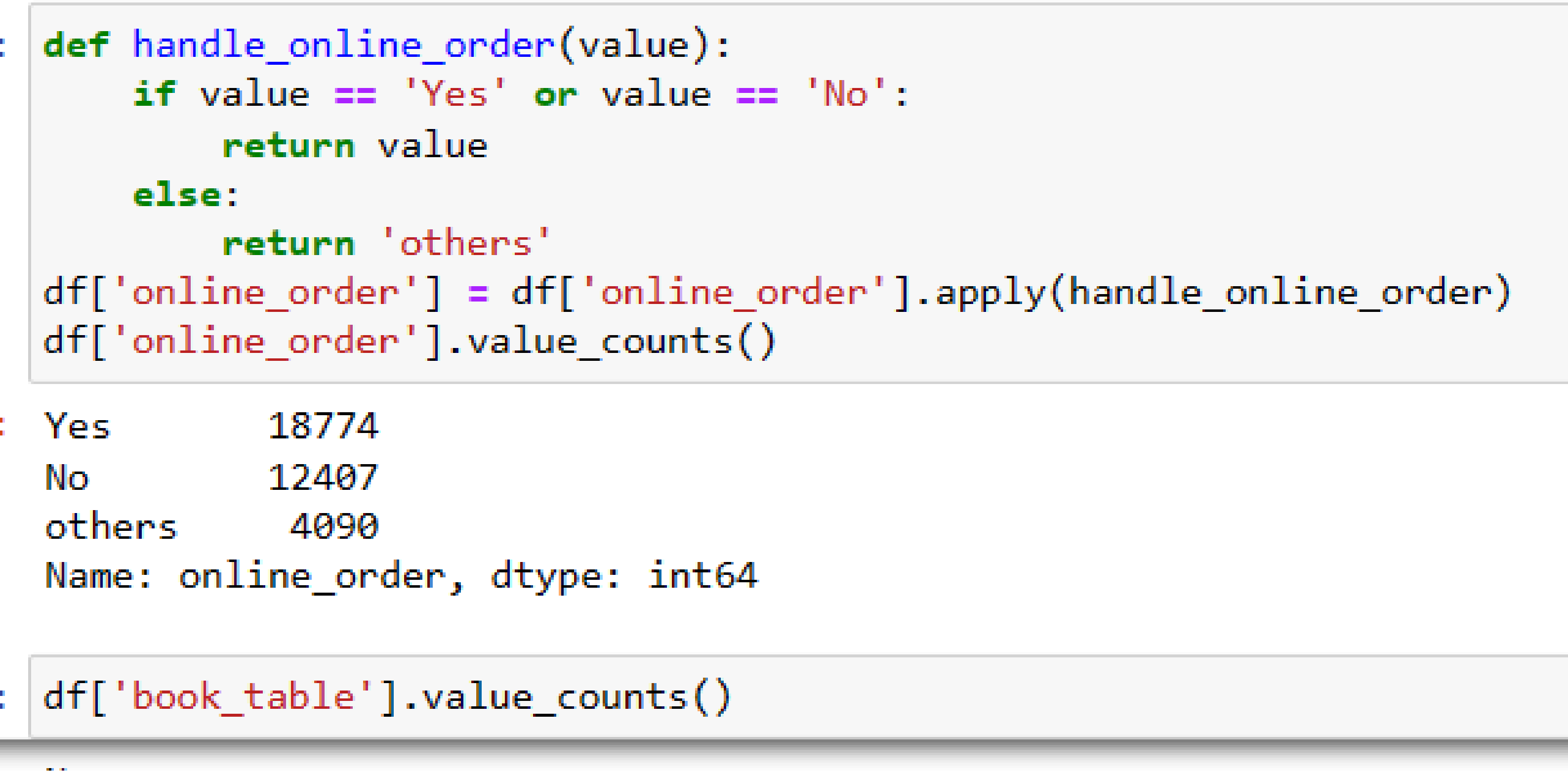

Distribution of counts_fill_type_mode
20000
17500
15000
12500
10000
7500
5000
2500
0
Delivery
Dine-out
Desserts
Cafes
Drinks & nightlife
Buffet
Pubs and bars
Category

```
columns
```



```python
#handling online_order column
df['online_order'].value_counts()
```

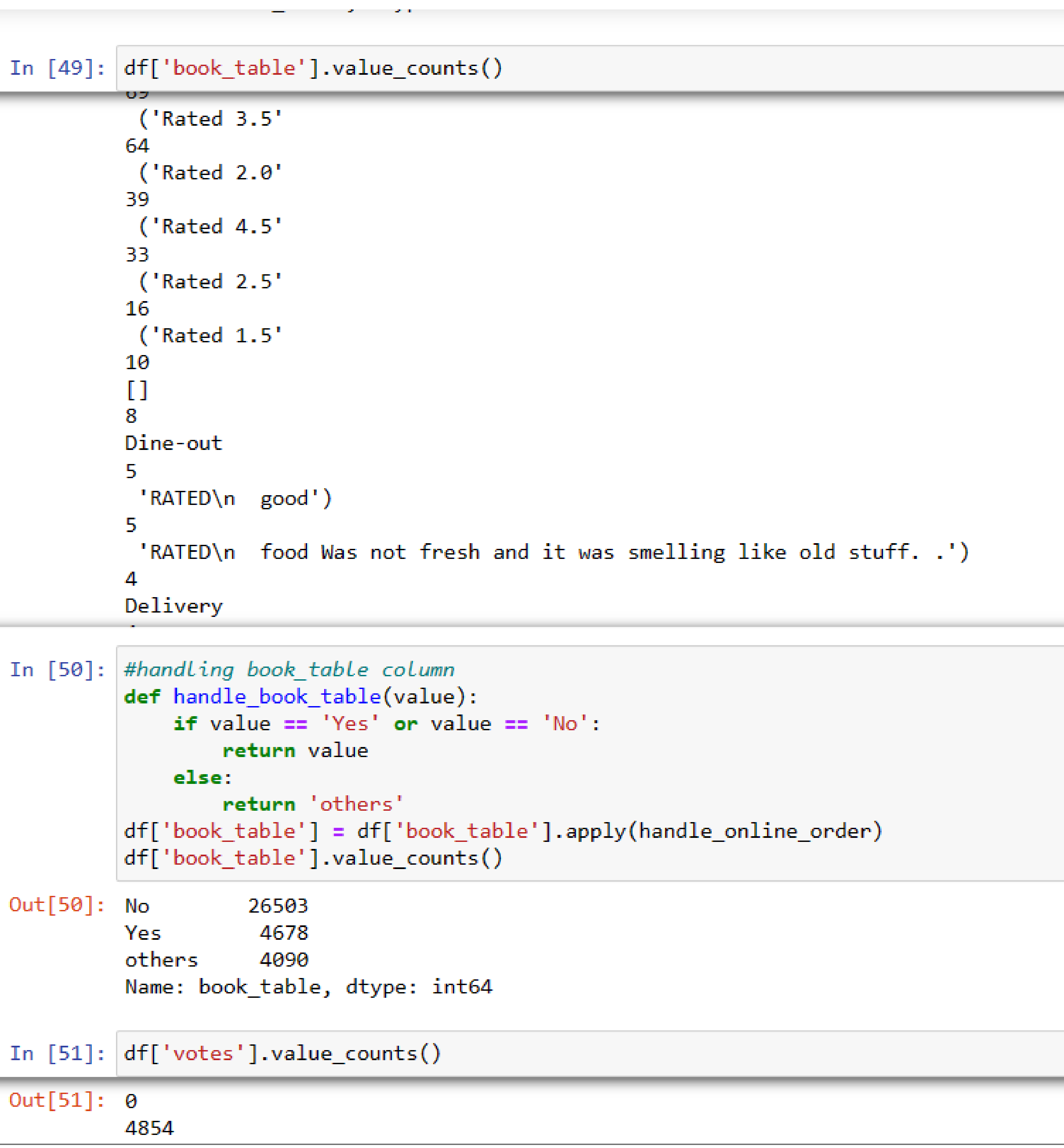
```
No
12407
 ('Rated 4.0'
439
 ('Rated 5.0'
361
 ('Rated 3.0'
170
 ('Rated 1.0'
136
 ('Rated 4.5'
64
 ('Rated 2.0'
61
 ('Rated 3.5'
60
 ('Rated 2.5'
21
 ('Rated 1.5'
```



```python
def handle_online_order(value):
    if value == 'Yes' or value == 'No':
        return value
    else:
        return 'others'
df['online_order'] = df['online_order'].apply(handle_online_order)
df['online_order'].value_counts()
```

```
Yes       18774
No        12407
others     4090
Name: online_order, dtype: int64
```

```python
df['book_table'].value_counts()
```



```
In [49]: df['book_table'].value_counts()
```

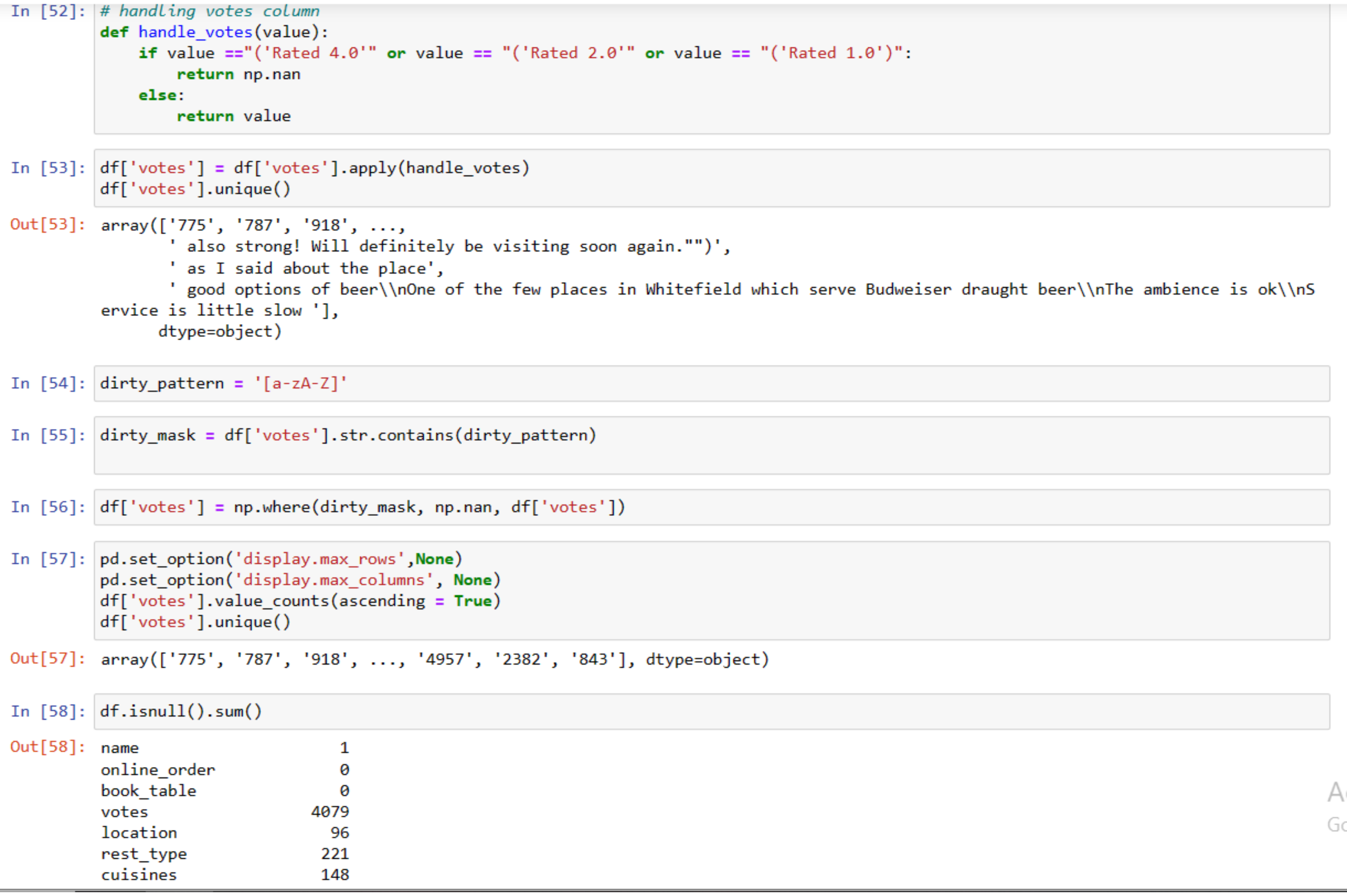
```
         69
          ('Rated 3.5'
         64
          ('Rated 2.0'
         39
          ('Rated 4.5'
         33
          ('Rated 2.5'
         16
          ('Rated 1.5'
         10
         []
         8
         Dine-out
         5
          'RATED\n  good')
         5
          'RATED\n  food Was not fresh and it was smelling like old stuff. .')
         4
         Delivery
```

```
In [50]: #handling book_table column
         def handle_book_table(value):
             if value == 'Yes' or value == 'No':
                 return value
             else:
                 return 'others'
         df['book_table'] = df['book_table'].apply(handle_online_order)
         df['book_table'].value_counts()
```

```
Out[50]: No         26503
         Yes         4678
         others      4090
         Name: book_table, dtype: int64
```

```
In [51]: df['votes'].value_counts()
```

```
Out[51]: 0
         4854
```



```
In [52]: # handling votes column
         def handle_votes(value):
             if value =="('Rated 4.0'" or value == "('Rated 2.0'" or value == "('Rated 1.0')":
                 return np.nan
             else:
                 return value
```

```
In [53]: df['votes'] = df['votes'].apply(handle_votes)
         df['votes'].unique()
```

```
Out[53]: array(['775', '787', '918', ...,
                ' also strong! Will definitely be visiting soon again."")',
                ' as I said about the place',
                ' good options of beer\\nOne of the few places in Whitefield which serve Budweiser draught beer\\nThe ambience is ok\\nS
         ervice is little slow '],
               dtype=object)
```

```
In [54]: dirty_pattern = '[a-zA-Z]'
```

```
In [55]: dirty_mask = df['votes'].str.contains(dirty_pattern)
```

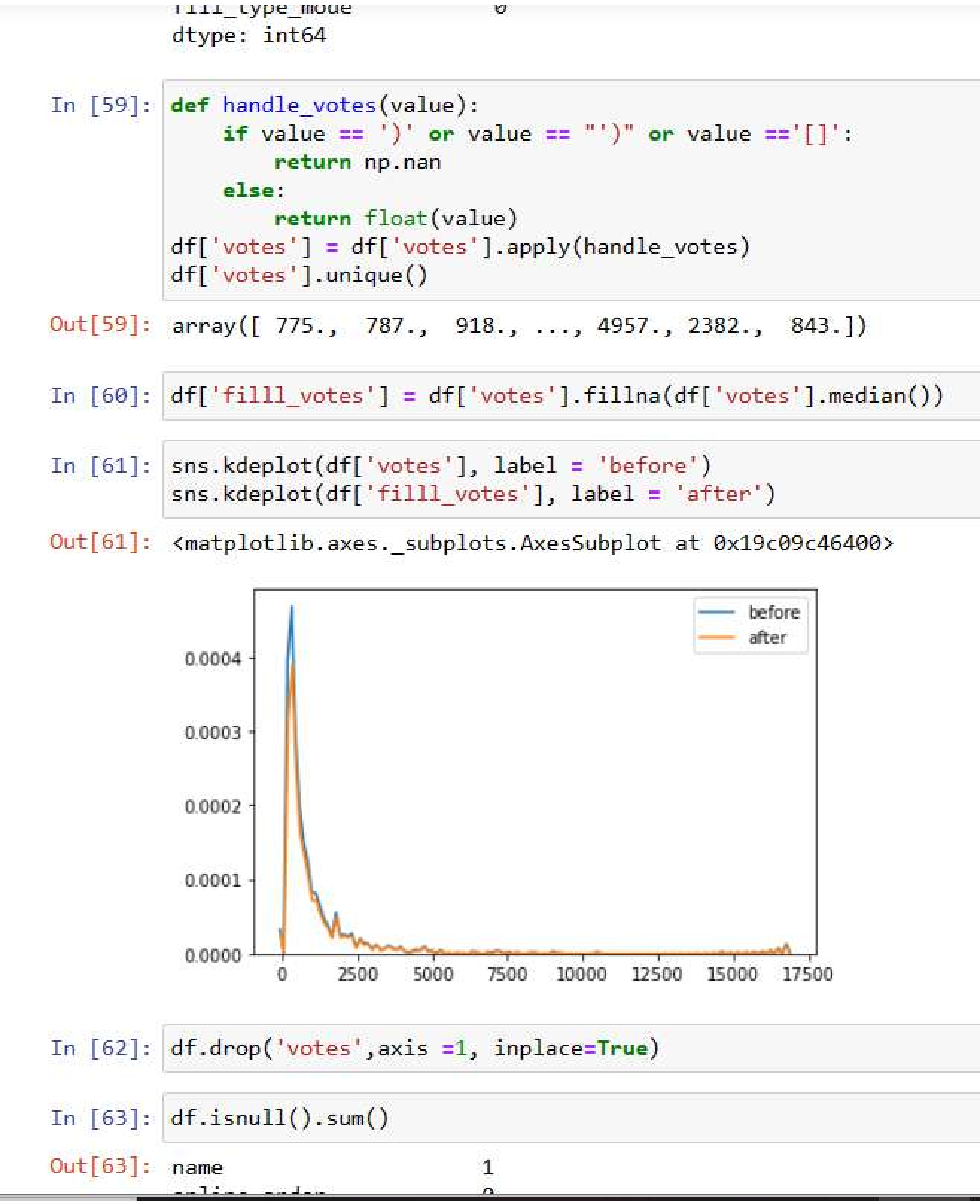
```
In [56]: df['votes'] = np.where(dirty_mask, np.nan, df['votes'])
```

```
In [57]: pd.set_option('display.max_rows',None)
         pd.set_option('display.max_columns', None)
         df['votes'].value_counts(ascending = True)
         df['votes'].unique()
```

```
Out[57]: array(['775', '787', '918', ..., '4957', '2382', '843'], dtype=object)
```

```
In [58]: df.isnull().sum()
```

```
Out[58]: name                1
         online_order        0
         book_table          0
         votes            4079
         location           96
         rest_type         221
         cuisines          148
```



```
fill_type_mode          0
dtype: int64
```

In [59]:
```
def handle_votes(value):
    if value == ')' or value == "')" or value =='[]':
        return np.nan
    else:
        return float(value)
df['votes'] = df['votes'].apply(handle_votes)
df['votes'].unique()
```

Out[59]:
```
array([ 775.,  787.,  918., ..., 4957., 2382.,  843.])
```

In [60]:
```
df['filll_votes'] = df['votes'].fillna(df['votes'].median())
```

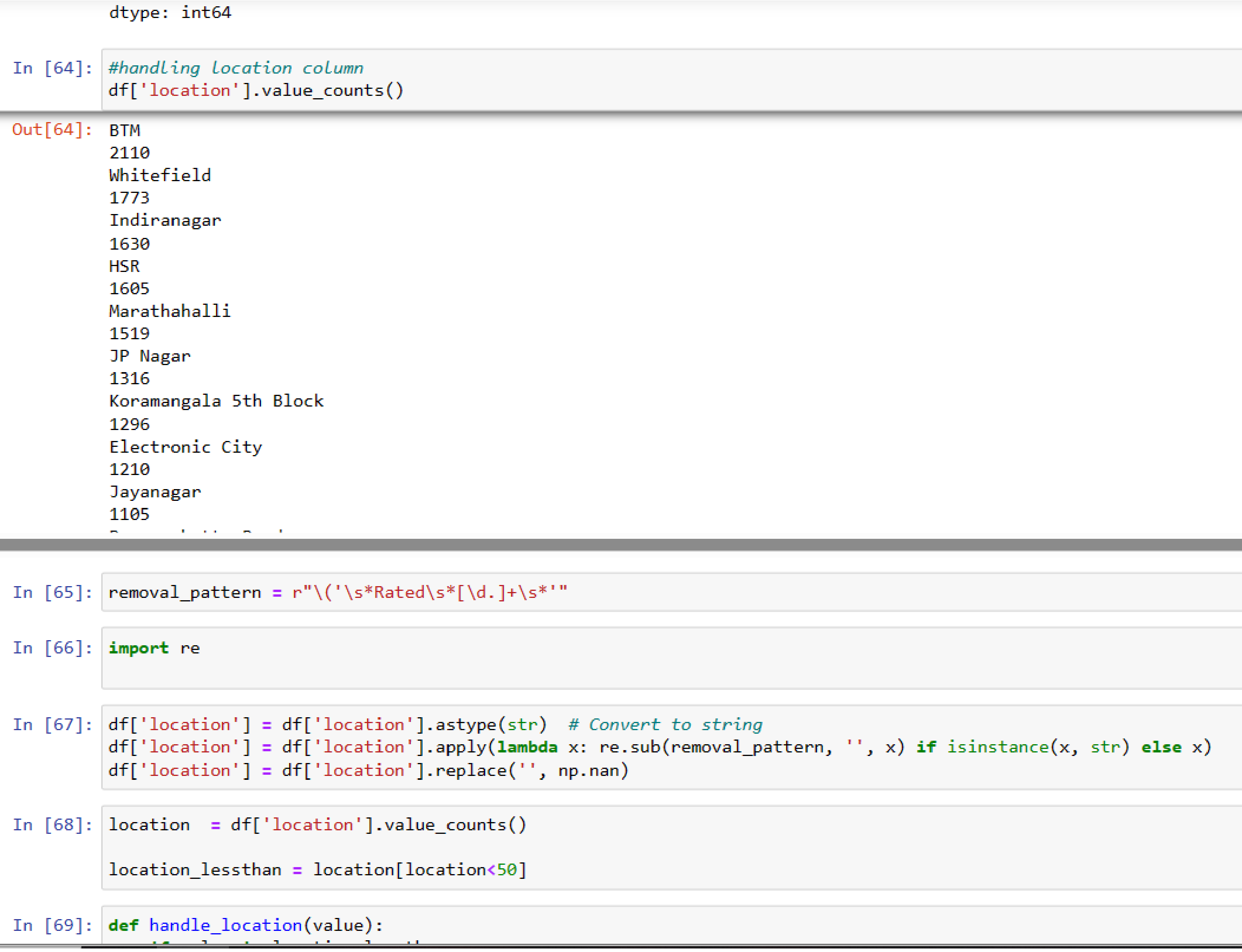
In [61]:
```
sns.kdeplot(df['votes'], label = 'before')
sns.kdeplot(df['filll_votes'], label = 'after')
```

Out[61]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x19c09c46400>
```

In [62]:
```
df.drop('votes',axis =1, inplace=True)
```

In [63]:
```
df.isnull().sum()
```

Out[63]:
```
name                    1
online_order            0
```



```
dtype: int64
```

```
In [64]: #handling location column
         df['location'].value_counts()
```

```
Out[64]: BTM
         2110
         Whitefield
         1773
         Indiranagar
         1630
         HSR
         1605
         Marathahalli
         1519
         JP Nagar
         1316
         Koramangala 5th Block
         1296
         Electronic City
         1210
         Jayanagar
         1105
```

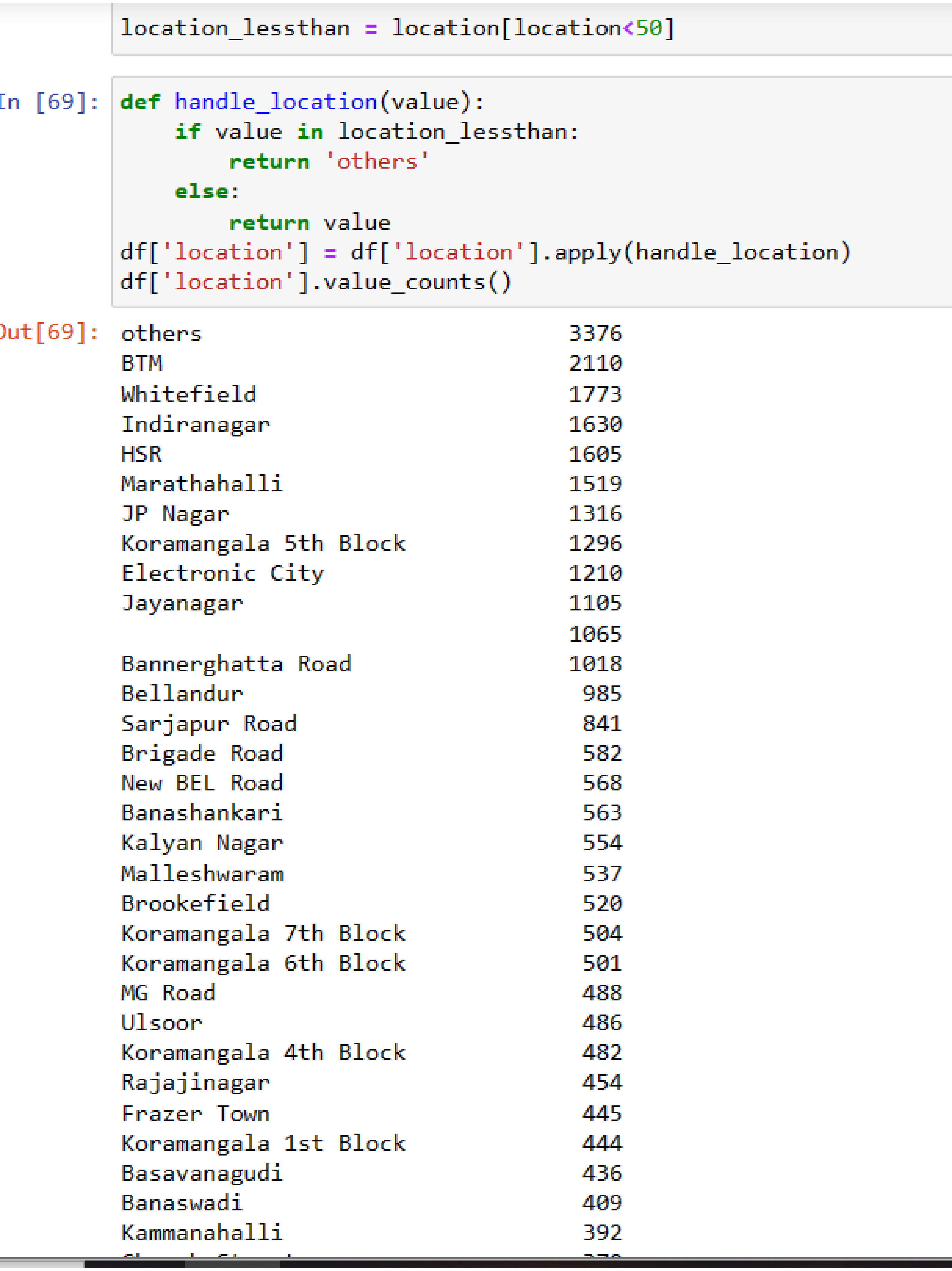
```
In [65]: removal_pattern = r"\('\s*Rated\s*[\d.]+\s*'"
```

```
In [66]: import re
```

```
In [67]: df['location'] = df['location'].astype(str)  # Convert to string
         df['location'] = df['location'].apply(lambda x: re.sub(removal_pattern, '', x) if isinstance(x, str) else x)
         df['location'] = df['location'].replace('', np.nan)
```

```
In [68]: location  = df['location'].value_counts()

         location_lessthan = location[location<50]
```

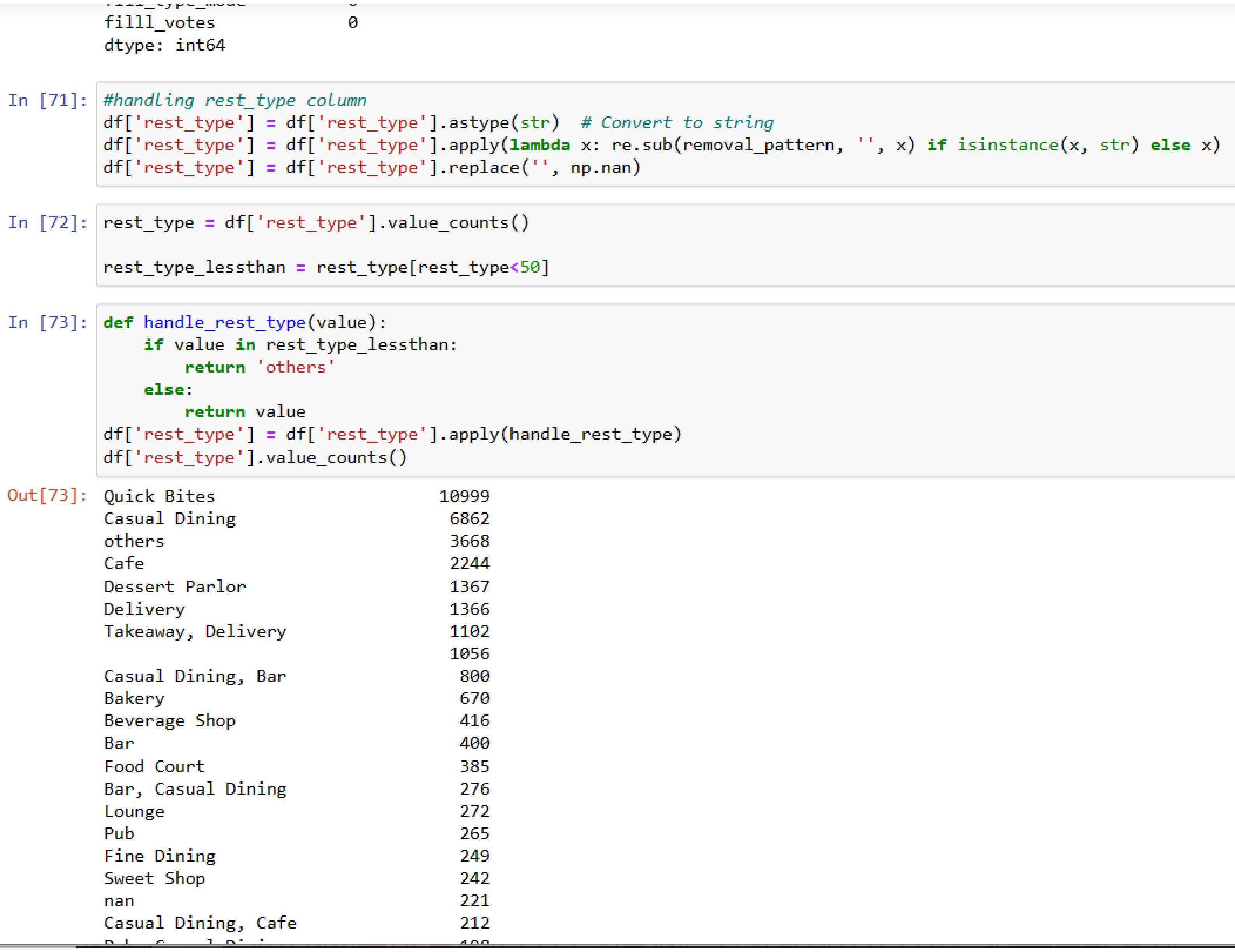
```
In [69]: def handle_location(value):
```



```
location_lesstham = location[location<50]
```

In [69]:
```
def handle_location(value):
    if value in location_lessthan:
        return 'others'
    else:
        return value
df['location'] = df['location'].apply(handle_location)
df['location'].value_counts()
```

Out[69]:
```
others                           3376
BTM                              2110
Whitefield                       1773
Indiranagar                      1630
HSR                              1605
Marathahalli                     1519
JP Nagar                         1316
Koramangala 5th Block            1296
Electronic City                  1210
Jayanagar                        1105
                                 1065
Bannerghatta Road                1018
Bellandur                         985
Sarjapur Road                     841
Brigade Road                      582
New BEL Road                      568
Banashankari                      563
Kalyan Nagar                      554
Malleshwaram                      537
Brookefield                       520
Koramangala 7th Block             504
Koramangala 6th Block             501
MG Road                           488
Ulsoor                            486
Koramangala 4th Block             482
Rajajinagar                       454
Frazer Town                       445
Koramangala 1st Block             444
Basavanagudi                      436
Banaswadi                         409
Kammanahalli                      392
```



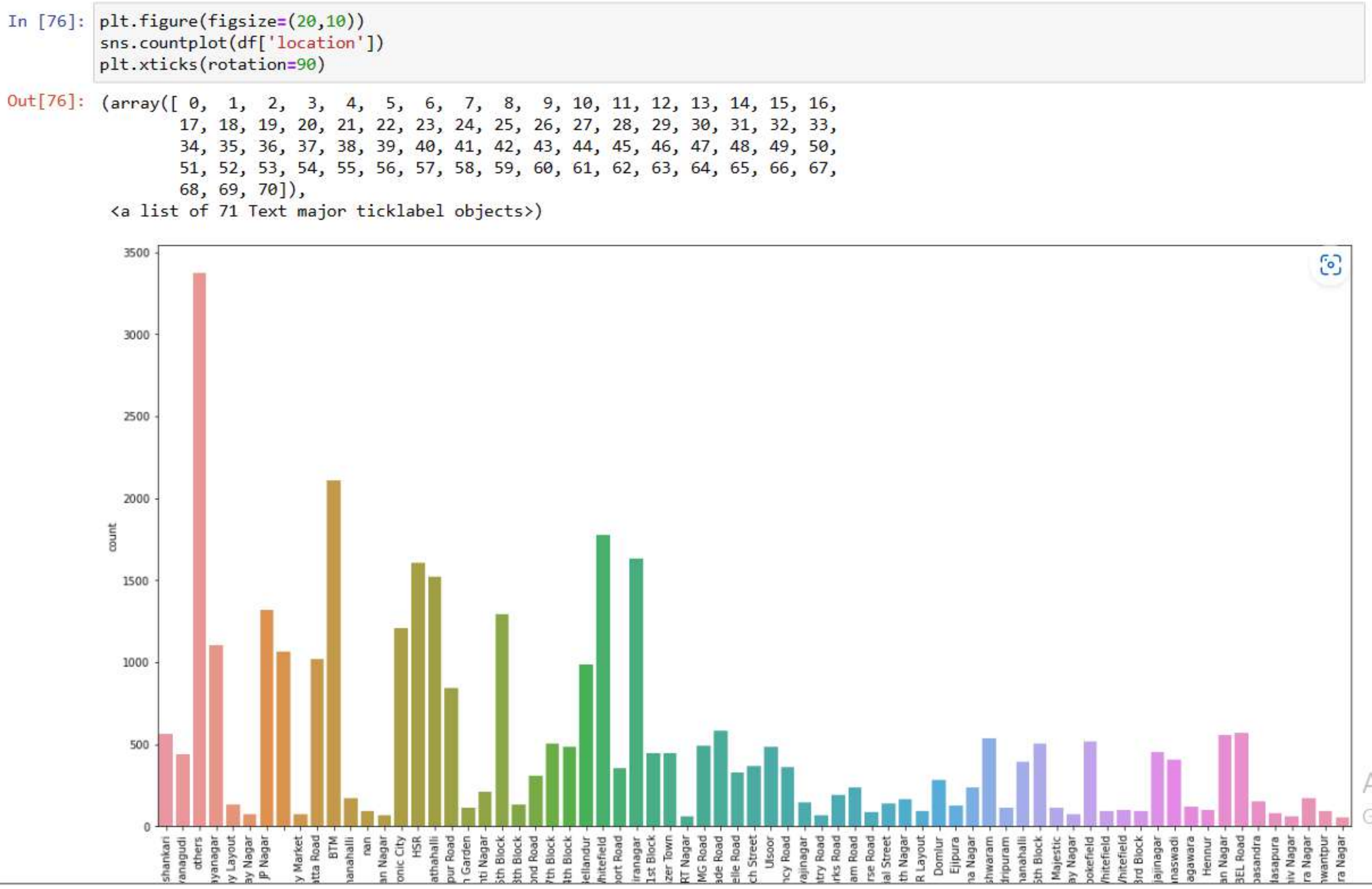
```
fill_type_mode        0
filll_votes           0
dtype: int64
```

In [71]:
```
#handling rest_type column
df['rest_type'] = df['rest_type'].astype(str)  # Convert to string
df['rest_type'] = df['rest_type'].apply(lambda x: re.sub(removal_pattern, '', x) if isinstance(x, str) else x)
df['rest_type'] = df['rest_type'].replace('', np.nan)
```

In [72]:
```
rest_type = df['rest_type'].value_counts()

rest_type_lessthan = rest_type[rest_type<50]
```

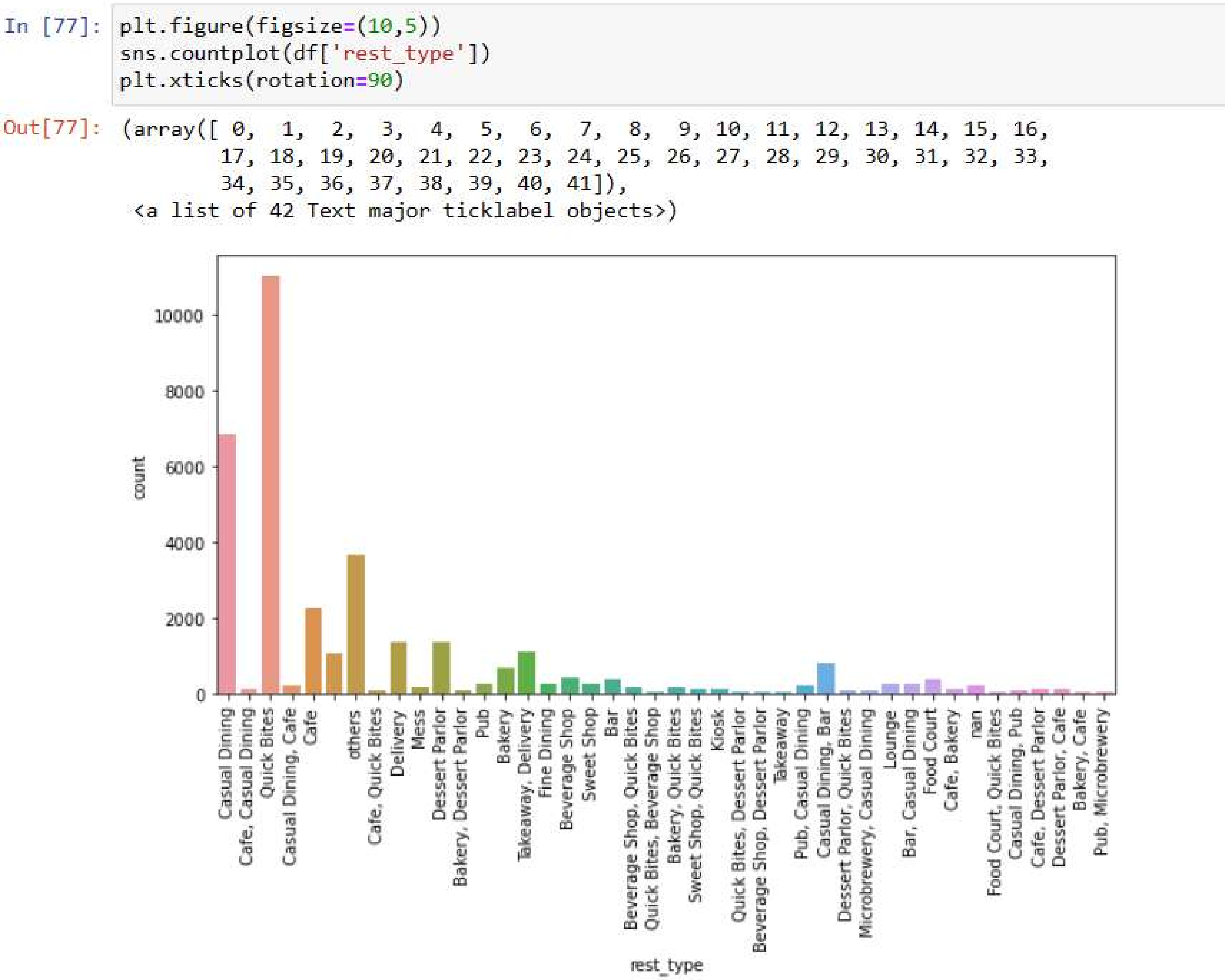
In [73]:
```
def handle_rest_type(value):
    if value in rest_type_lessthan:
        return 'others'
    else:
        return value
df['rest_type'] = df['rest_type'].apply(handle_rest_type)
df['rest_type'].value_counts()
```

Out[73]:
```
Quick Bites                      10999
Casual Dining                     6862
others                            3668
Cafe                              2244
Dessert Parlor                    1367
Delivery                          1366
Takeaway, Delivery                1102
                                  1056
Casual Dining, Bar                 800
Bakery                             670
Beverage Shop                      416
Bar                                400
Food Court                         385
Bar, Casual Dining                 276
Lounge                             272
Pub                                265
Fine Dining                        249
Sweet Shop                         242
nan                                221
Casual Dining, Cafe                212
Pub, Casual Dining                 198
```



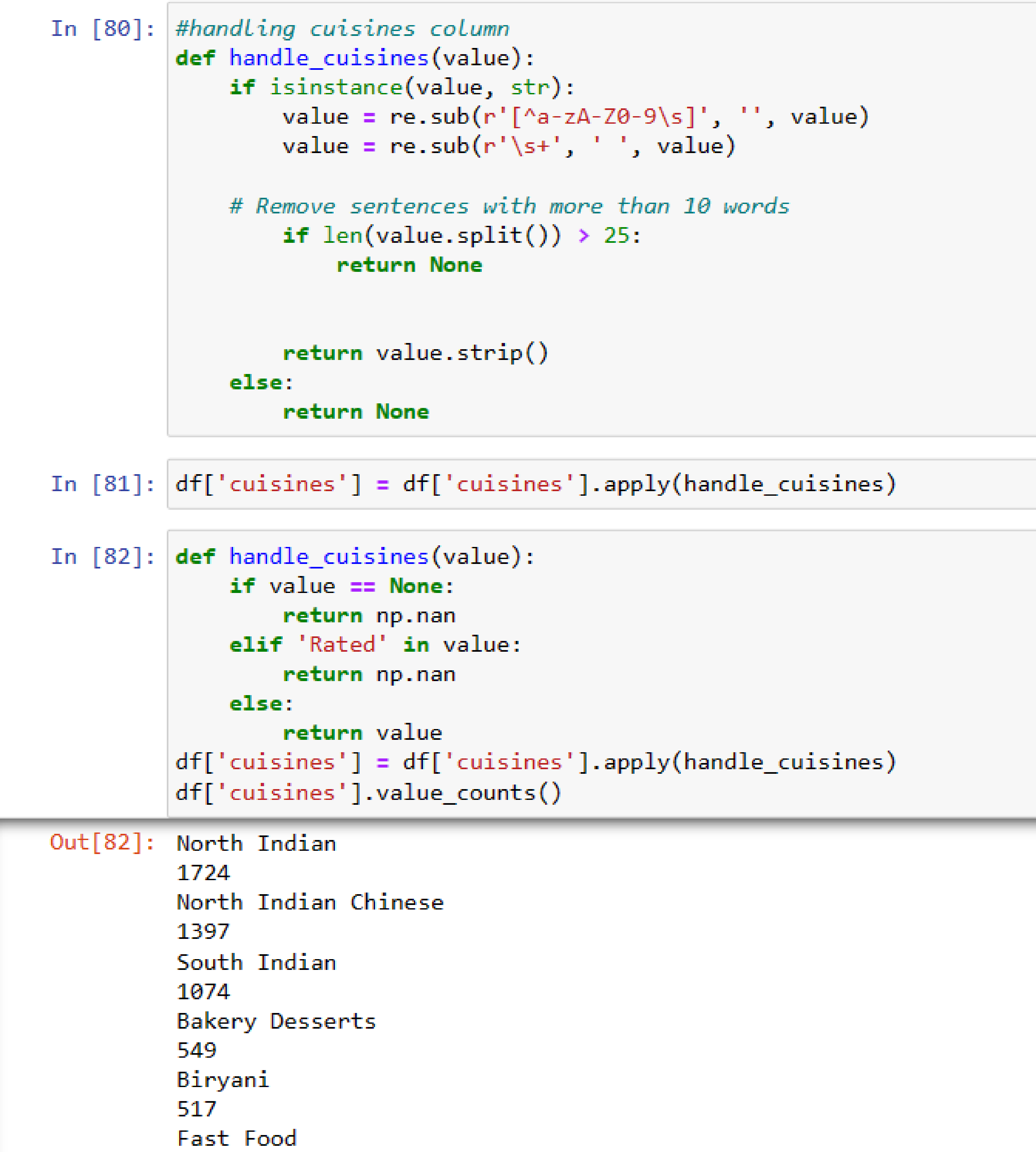
```
In [76]: plt.figure(figsize=(20,10))
         sns.countplot(df['location'])
         plt.xticks(rotation=90)
```

```
Out[76]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
                 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
                 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
                 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
                 68, 69, 70]),
          <a list of 71 Text major ticklabel objects>)
```



```
In [77]: plt.figure(figsize=(10,5))
         sns.countplot(df['rest_type'])
         plt.xticks(rotation=90)
```

```
Out[77]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
                 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
                 34, 35, 36, 37, 38, 39, 40, 41]),
          <a list of 42 Text major ticklabel objects>)
```

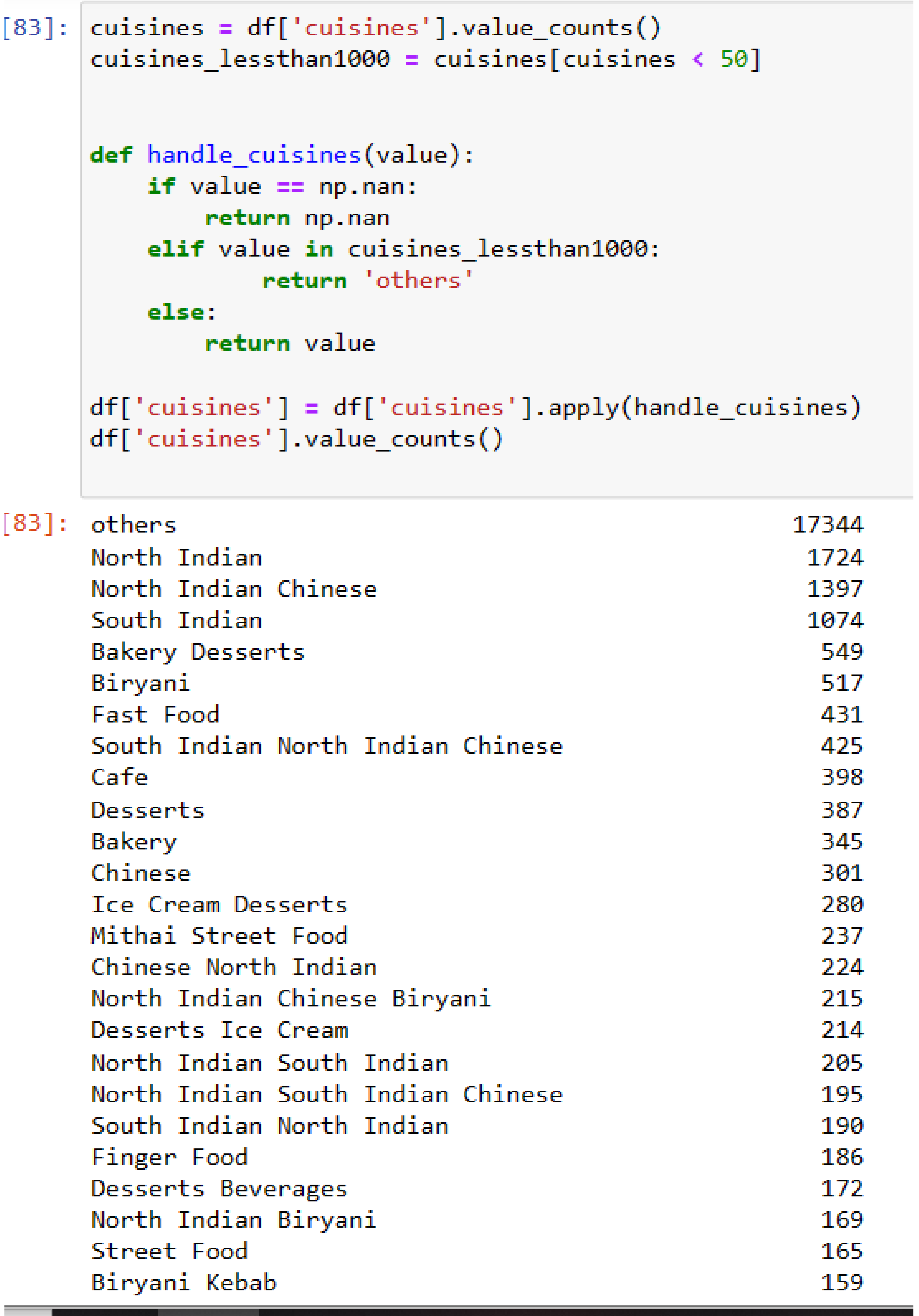


```
In [80]: #handling cuisines column
         def handle_cuisines(value):
             if isinstance(value, str):
                 value = re.sub(r'[^a-zA-Z0-9\s]', '', value)
                 value = re.sub(r'\s+', ' ', value)

             # Remove sentences with more than 10 words
                 if len(value.split()) > 25:
                     return None


                 return value.strip()
             else:
                 return None
```

```
In [81]: df['cuisines'] = df['cuisines'].apply(handle_cuisines)
```

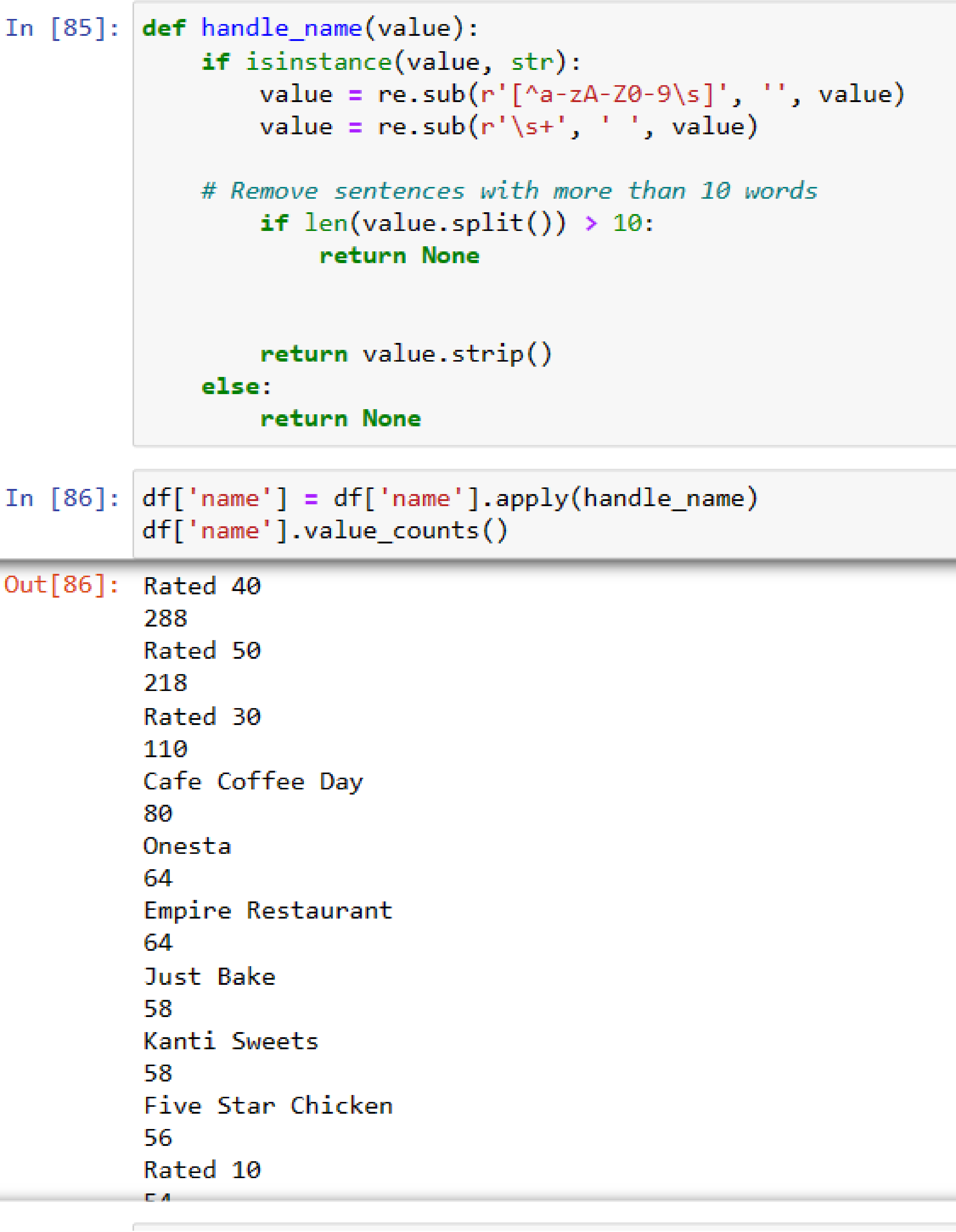
```
In [82]: def handle_cuisines(value):
             if value == None:
                 return np.nan
             elif 'Rated' in value:
                 return np.nan
             else:
                 return value
         df['cuisines'] = df['cuisines'].apply(handle_cuisines)
         df['cuisines'].value_counts()
```

```
Out[82]: North Indian
         1724
         North Indian Chinese
         1397
         South Indian
         1074
         Bakery Desserts
         549
         Biryani
         517
         Fast Food
```



```
[83]: cuisines = df['cuisines'].value_counts()
      cuisines_lessthan1000 = cuisines[cuisines < 50]


      def handle_cuisines(value):
          if value == np.nan:
              return np.nan
          elif value in cuisines_lessthan1000:
                  return 'others'
          else:
              return value

      df['cuisines'] = df['cuisines'].apply(handle_cuisines)
      df['cuisines'].value_counts()
```

```
[83]: others                                    17344
      North Indian                               1724
      North Indian Chinese                       1397
      South Indian                               1074
      Bakery Desserts                             549
      Biryani                                     517
      Fast Food                                   431
      South Indian North Indian Chinese           425
      Cafe                                        398
      Desserts                                    387
      Bakery                                      345
      Chinese                                     301
      Ice Cream Desserts                          280
      Mithai Street Food                          237
      Chinese North Indian                        224
      North Indian Chinese Biryani                215
      Desserts Ice Cream                          214
      North Indian South Indian                   205
      North Indian South Indian Chinese           195
      South Indian North Indian                   190
      Finger Food                                 186
      Desserts Beverages                          172
      North Indian Biryani                        169
      Street Food                                 165
      Biryani Kebab                               159
```



In [85]:

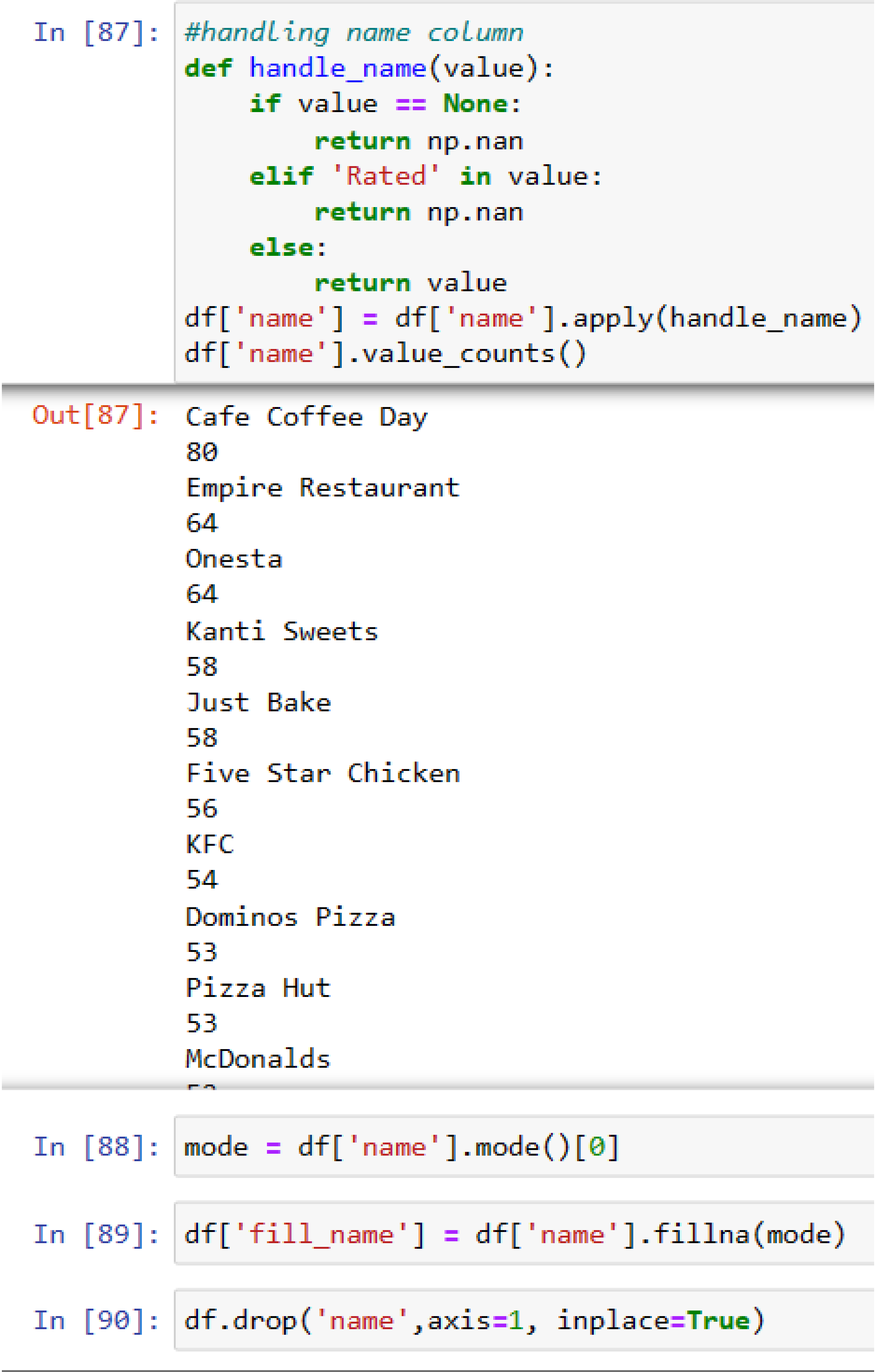
```python
def handle_name(value):
    if isinstance(value, str):
        value = re.sub(r'[^a-zA-Z0-9\s]', '', value)
        value = re.sub(r'\s+', ' ', value)

    # Remove sentences with more than 10 words
        if len(value.split()) > 10:
            return None


        return value.strip()
    else:
        return None
```

In [86]:

```python
df['name'] = df['name'].apply(handle_name)
df['name'].value_counts()
```

Out[86]:

```
Rated 40
288
Rated 50
218
Rated 30
110
Cafe Coffee Day
80
Onesta
64
Empire Restaurant
64
Just Bake
58
Kanti Sweets
58
Five Star Chicken
56
Rated 10
54
```



In [87]:

```
#handling name column
def handle_name(value):
    if value == None:
        return np.nan
    elif 'Rated' in value:
        return np.nan
    else:
        return value
df['name'] = df['name'].apply(handle_name)
df['name'].value_counts()
```

Out[87]:

```
Cafe Coffee Day
80
Empire Restaurant
64
Onesta
64
Kanti Sweets
58
Just Bake
58
Five Star Chicken
56
KFC
54
Dominos Pizza
53
Pizza Hut
53
McDonalds
```

In [88]:

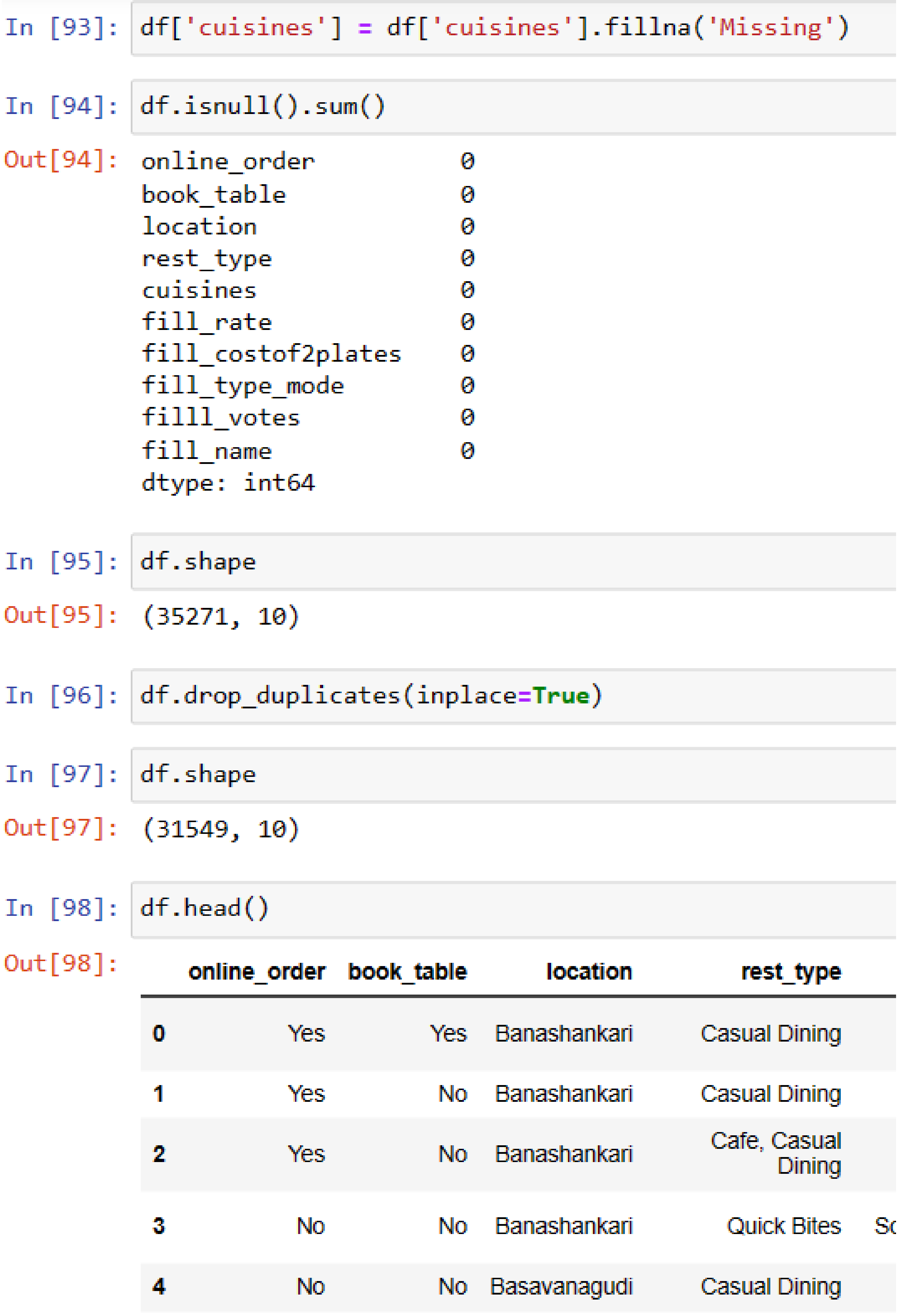
```
mode = df['name'].mode()[0]
```

In [89]:

```
df['fill_name'] = df['name'].fillna(mode)
```

In [90]:

```
df.drop('name',axis=1, inplace=True)
```



In [93]:
```
df['cuisines'] = df['cuisines'].fillna('Missing')
```

In [94]:
```
df.isnull().sum()
```

Out[94]:
```
online_order        0
book_table          0
location            0
rest_type           0
cuisines            0
fill_rate           0
fill_costof2plates  0
fill_type_mode      0
filll_votes         0
fill_name           0
dtype: int64
```

In [95]:
```
df.shape
```

Out[95]:
```
(35271, 10)
```

In [96]:
```
df.drop_duplicates(inplace=True)
```

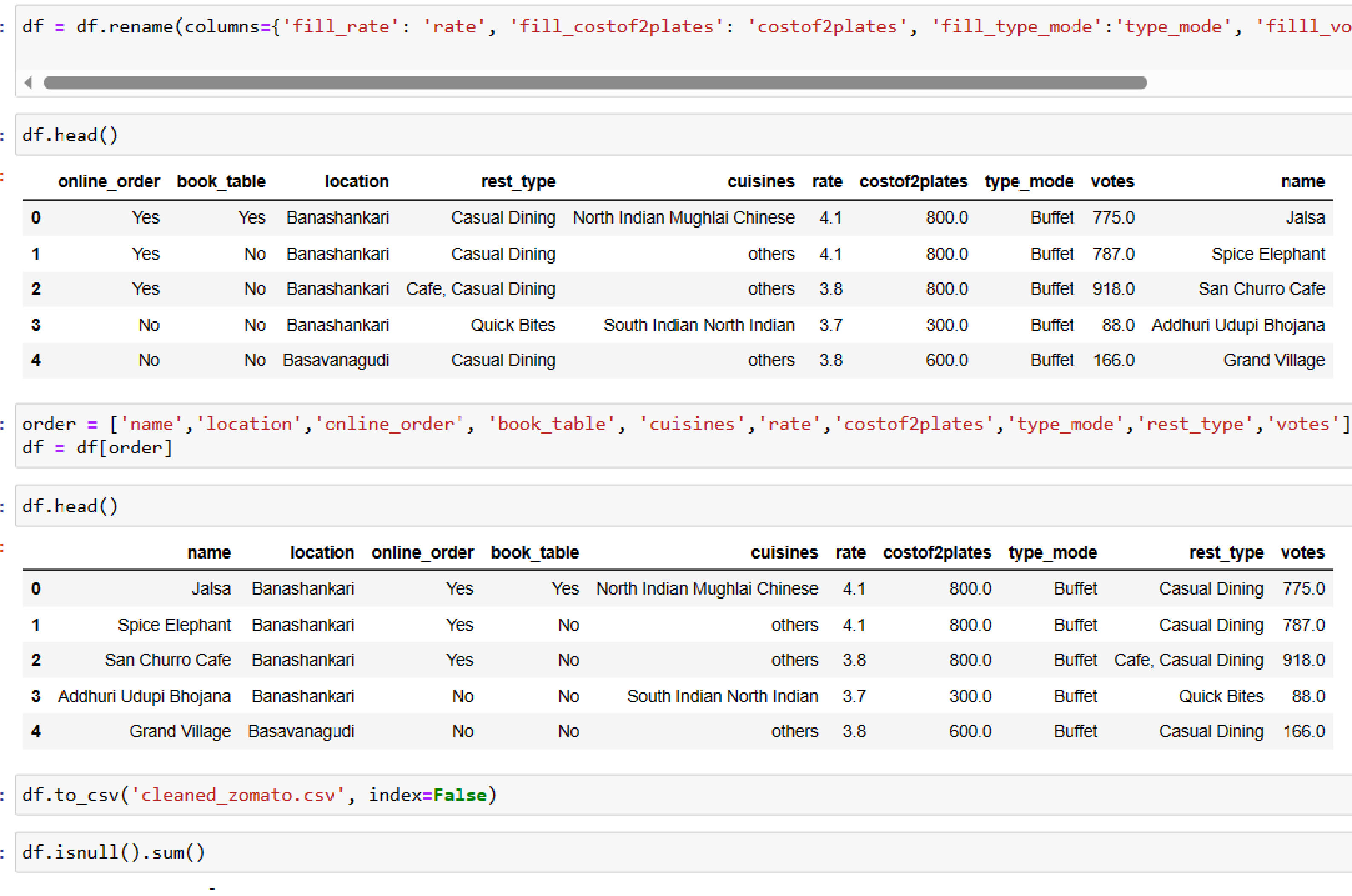
In [97]:
```
df.shape
```

Out[97]:
```
(31549, 10)
```

In [98]:
```
df.head()
```

Out[98]:

	online_order	book_table	location	rest_type	
0	Yes	Yes	Banashankari	Casual Dining	
1	Yes	No	Banashankari	Casual Dining	
2	Yes	No	Banashankari	Cafe, Casual Dining	
3	No	No	Banashankari	Quick Bites	Sc
4	No	No	Basavanagudi	Casual Dining	



```
df = df.rename(columns={'fill_rate': 'rate', 'fill_costof2plates': 'costof2plates', 'fill_type_mode':'type_mode', 'filll_vo
```

```
df.head()
```

	online_order	book_table	location	rest_type	cuisines	rate	costof2plates	type_mode	votes	name
0	Yes	Yes	Banashankari	Casual Dining	North Indian Mughlai Chinese	4.1	800.0	Buffet	775.0	Jalsa
1	Yes	No	Banashankari	Casual Dining	others	4.1	800.0	Buffet	787.0	Spice Elephant
2	Yes	No	Banashankari	Cafe, Casual Dining	others	3.8	800.0	Buffet	918.0	San Churro Cafe
3	No	No	Banashankari	Quick Bites	South Indian North Indian	3.7	300.0	Buffet	88.0	Addhuri Udupi Bhojana
4	No	No	Basavanagudi	Casual Dining	others	3.8	600.0	Buffet	166.0	Grand Village

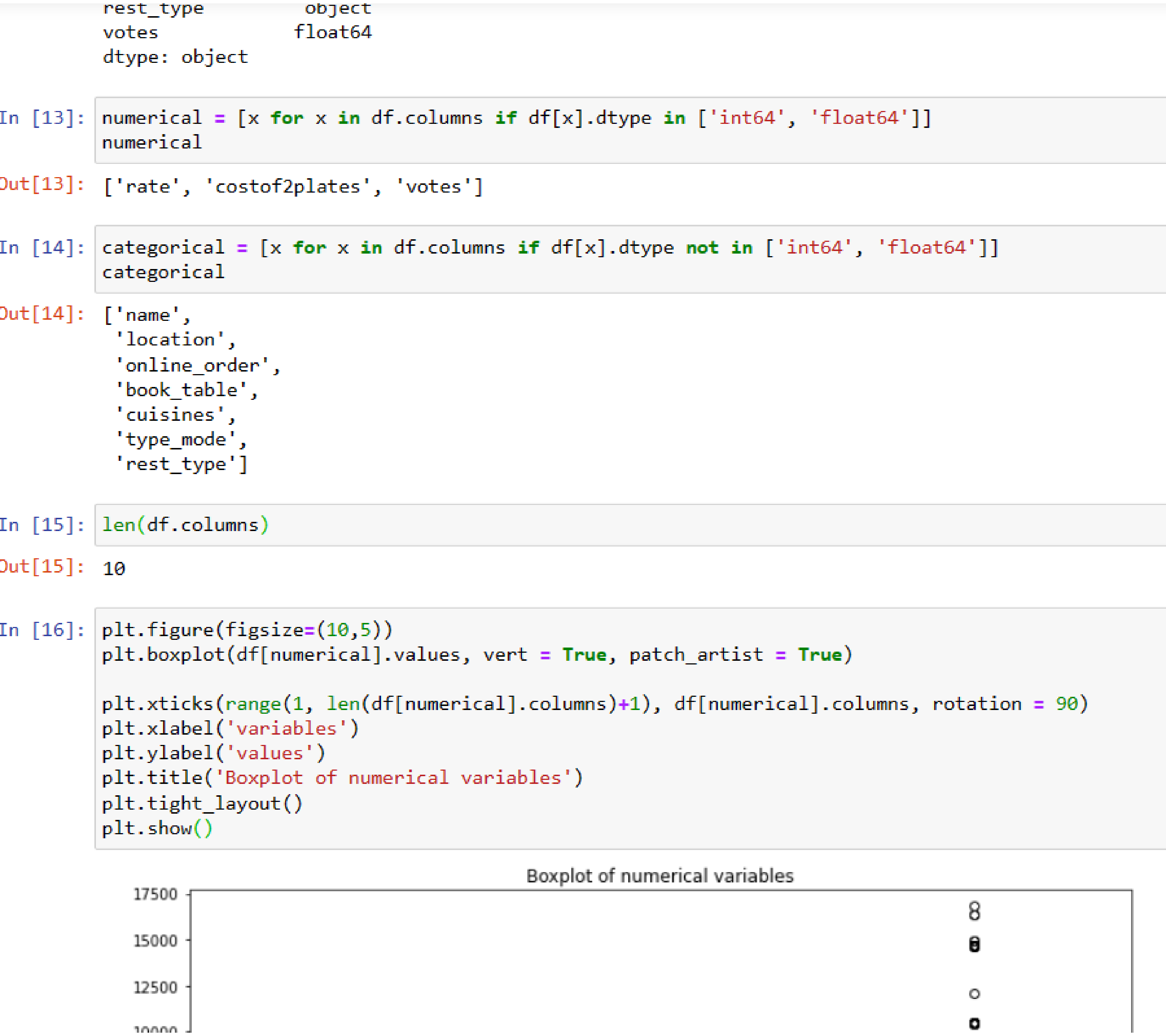
```
order = ['name','location','online_order', 'book_table', 'cuisines','rate','costof2plates','type_mode','rest_type','votes']
df = df[order]
```

```
df.head()
```

	name	location	online_order	book_table	cuisines	rate	costof2plates	type_mode	rest_type	votes
0	Jalsa	Banashankari	Yes	Yes	North Indian Mughlai Chinese	4.1	800.0	Buffet	Casual Dining	775.0
1	Spice Elephant	Banashankari	Yes	No	others	4.1	800.0	Buffet	Casual Dining	787.0
2	San Churro Cafe	Banashankari	Yes	No	others	3.8	800.0	Buffet	Cafe, Casual Dining	918.0
3	Addhuri Udupi Bhojana	Banashankari	No	No	South Indian North Indian	3.7	300.0	Buffet	Quick Bites	88.0
4	Grand Village	Basavanagudi	No	No	others	3.8	600.0	Buffet	Casual Dining	166.0

```
df.to_csv('cleaned_zomato.csv', index=False)
```

```
df.isnull().sum()
```



```
rest_type          object
votes             float64
dtype: object
```

In [13]:
```
numerical = [x for x in df.columns if df[x].dtype in ['int64', 'float64']]
numerical
```

Out[13]:
```
['rate', 'costof2plates', 'votes']
```

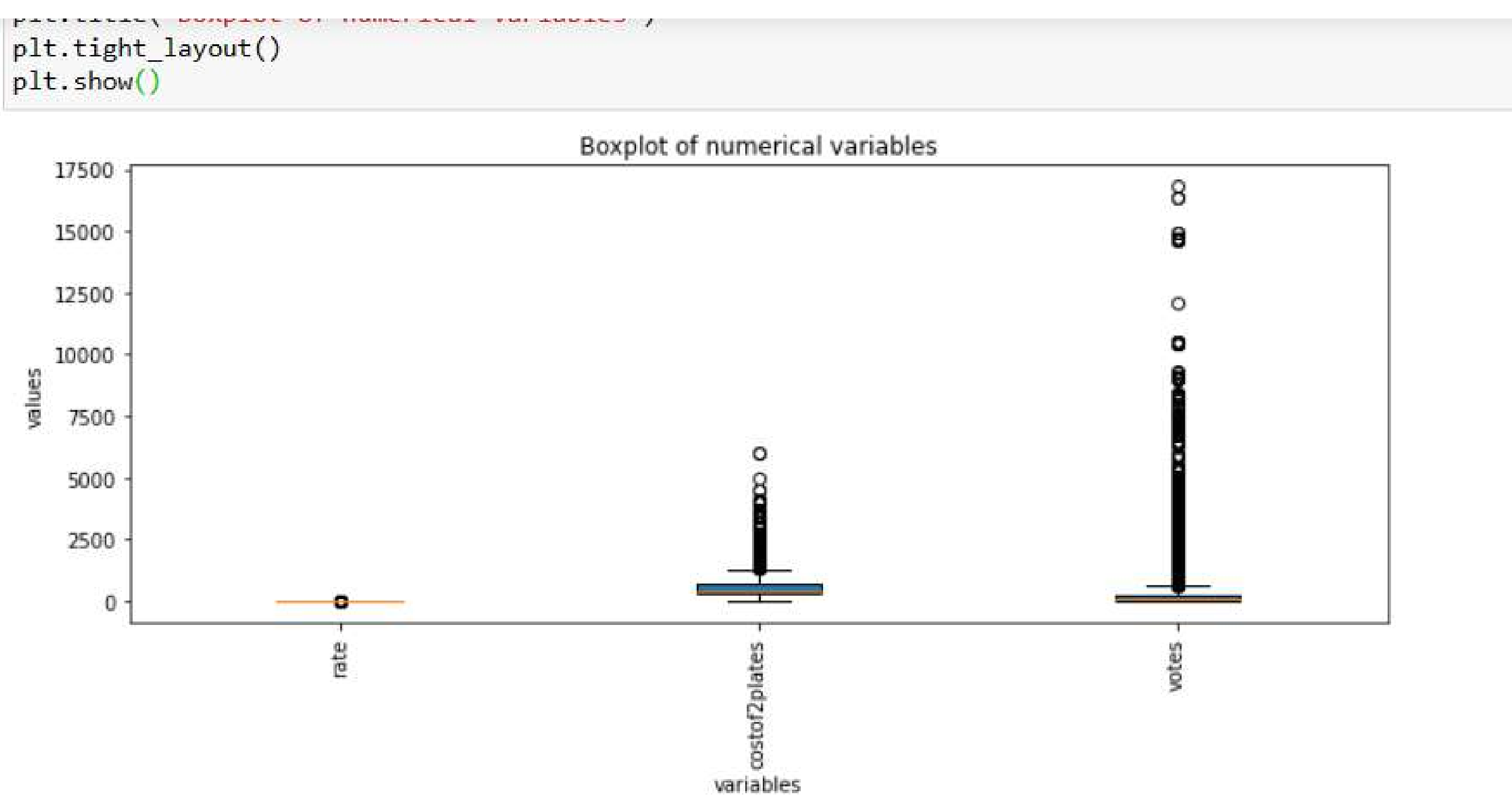
In [14]:
```
categorical = [x for x in df.columns if df[x].dtype not in ['int64', 'float64']]
categorical
```

Out[14]:
```
['name',
 'location',
 'online_order',
 'book_table',
 'cuisines',
 'type_mode',
 'rest_type']
```

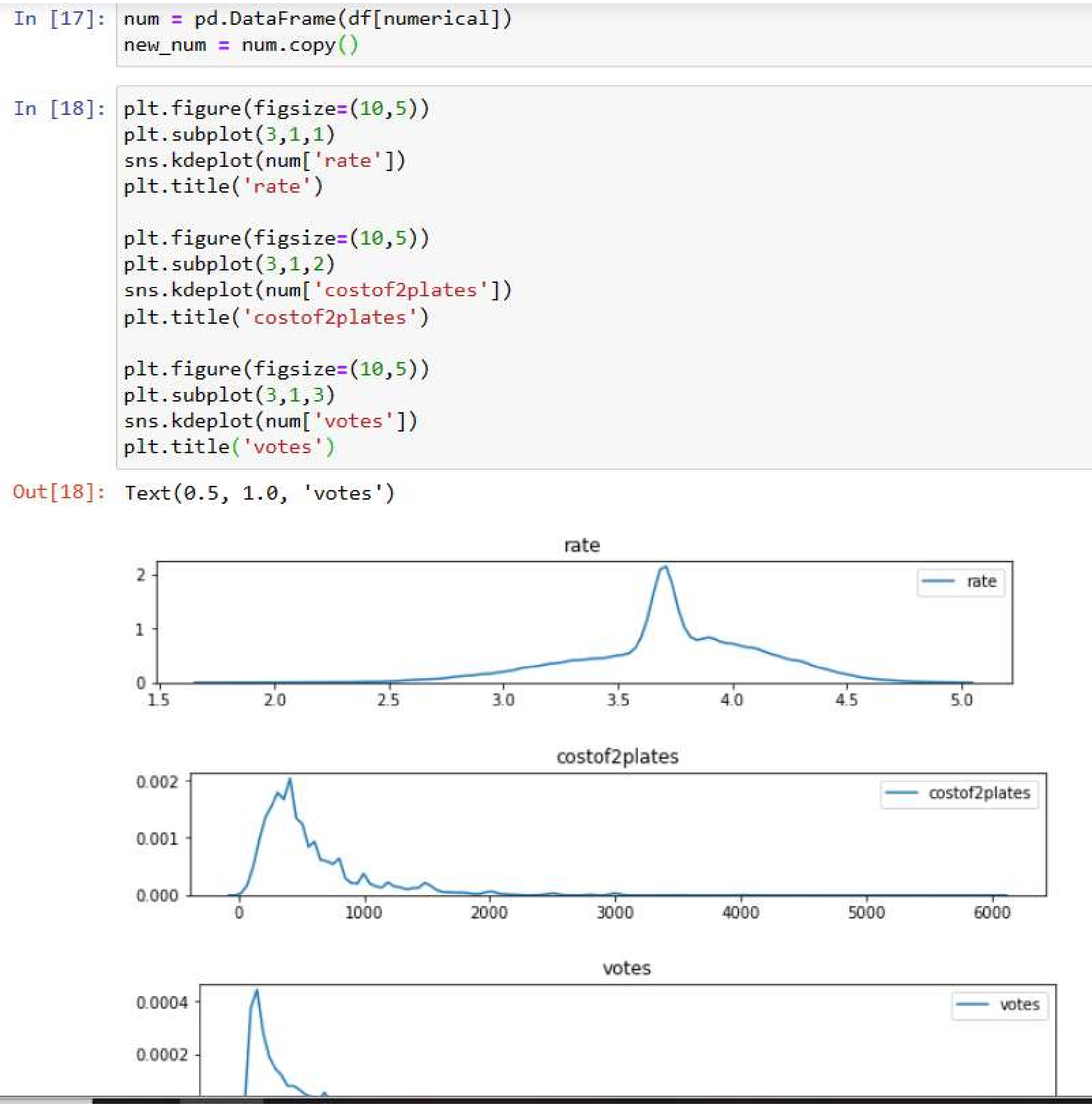
In [15]:
```
len(df.columns)
```

Out[15]:
```
10
```

In [16]:
```
plt.figure(figsize=(10,5))
plt.boxplot(df[numerical].values, vert = True, patch_artist = True)

plt.xticks(range(1, len(df[numerical].columns)+1), df[numerical].columns, rotation = 90)
plt.xlabel('variables')
plt.ylabel('values')
plt.title('Boxplot of numerical variables')
plt.tight_layout()
plt.show()
```



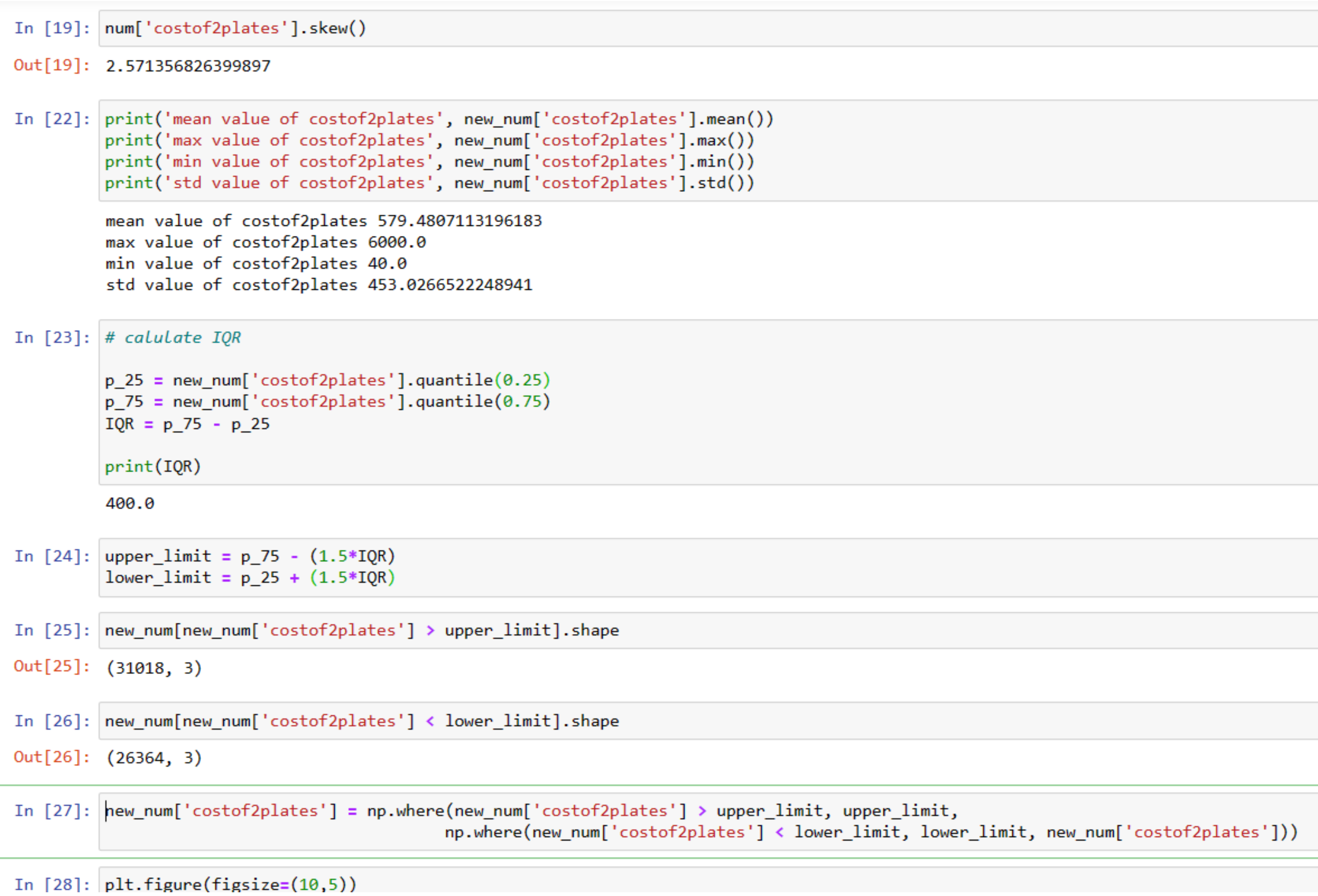
```
plt.title('Boxplot of numerical variables')
plt.tight_layout()
plt.show()
```



```
In [17]: num = pd.DataFrame(df[numerical])
         new_num = num.copy()
```

```
In [18]: plt.figure(figsize=(10,5))
         plt.subplot(3,1,1)
         sns.kdeplot(num['rate'])
         plt.title('rate')

         plt.figure(figsize=(10,5))
         plt.subplot(3,1,2)
         sns.kdeplot(num['costof2plates'])
         plt.title('costof2plates')

         plt.figure(figsize=(10,5))
         plt.subplot(3,1,3)
         sns.kdeplot(num['votes'])
         plt.title('votes')
```

```
Out[18]: Text(0.5, 1.0, 'votes')
```



```
In [19]: num['costof2plates'].skew()

Out[19]: 2.571356826399897
```

```
In [22]: print('mean value of costof2plates', new_num['costof2plates'].mean())
         print('max value of costof2plates', new_num['costof2plates'].max())
         print('min value of costof2plates', new_num['costof2plates'].min())
         print('std value of costof2plates', new_num['costof2plates'].std())

         mean value of costof2plates 579.4807113196183
         max value of costof2plates 6000.0
         min value of costof2plates 40.0
         std value of costof2plates 453.0266522248941
```

```
In [23]: # calulate IQR

         p_25 = new_num['costof2plates'].quantile(0.25)
         p_75 = new_num['costof2plates'].quantile(0.75)
         IQR = p_75 - p_25

         print(IQR)

         400.0
```

```
In [24]: upper_limit = p_75 - (1.5*IQR)
         lower_limit = p_25 + (1.5*IQR)
```

```
In [25]: new_num[new_num['costof2plates'] > upper_limit].shape

Out[25]: (31018, 3)
```

```
In [26]: new_num[new_num['costof2plates'] < lower_limit].shape

Out[26]: (26364, 3)
```

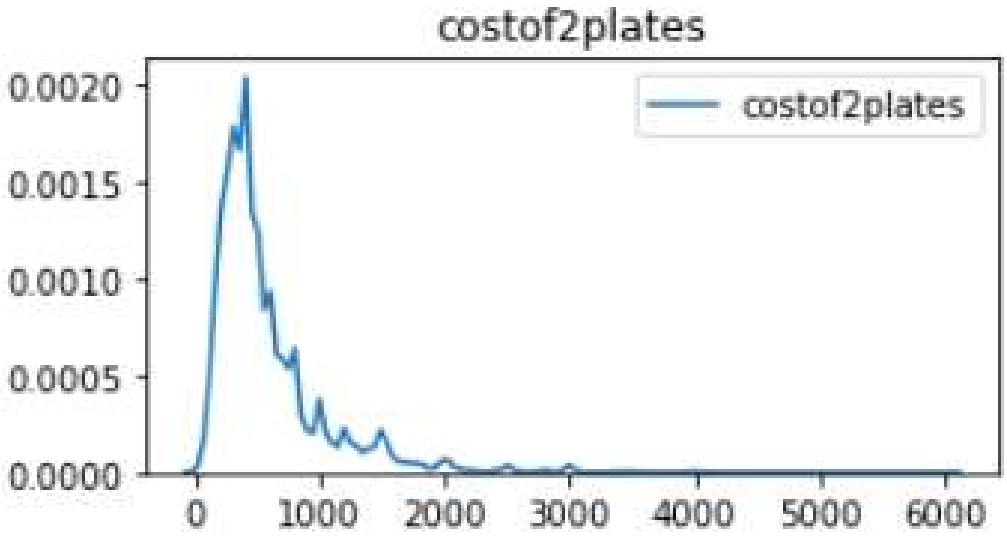
```
In [27]: new_num['costof2plates'] = np.where(new_num['costof2plates'] > upper_limit, upper_limit,
                                    np.where(new_num['costof2plates'] < lower_limit, lower_limit, new_num['costof2plates']))
```

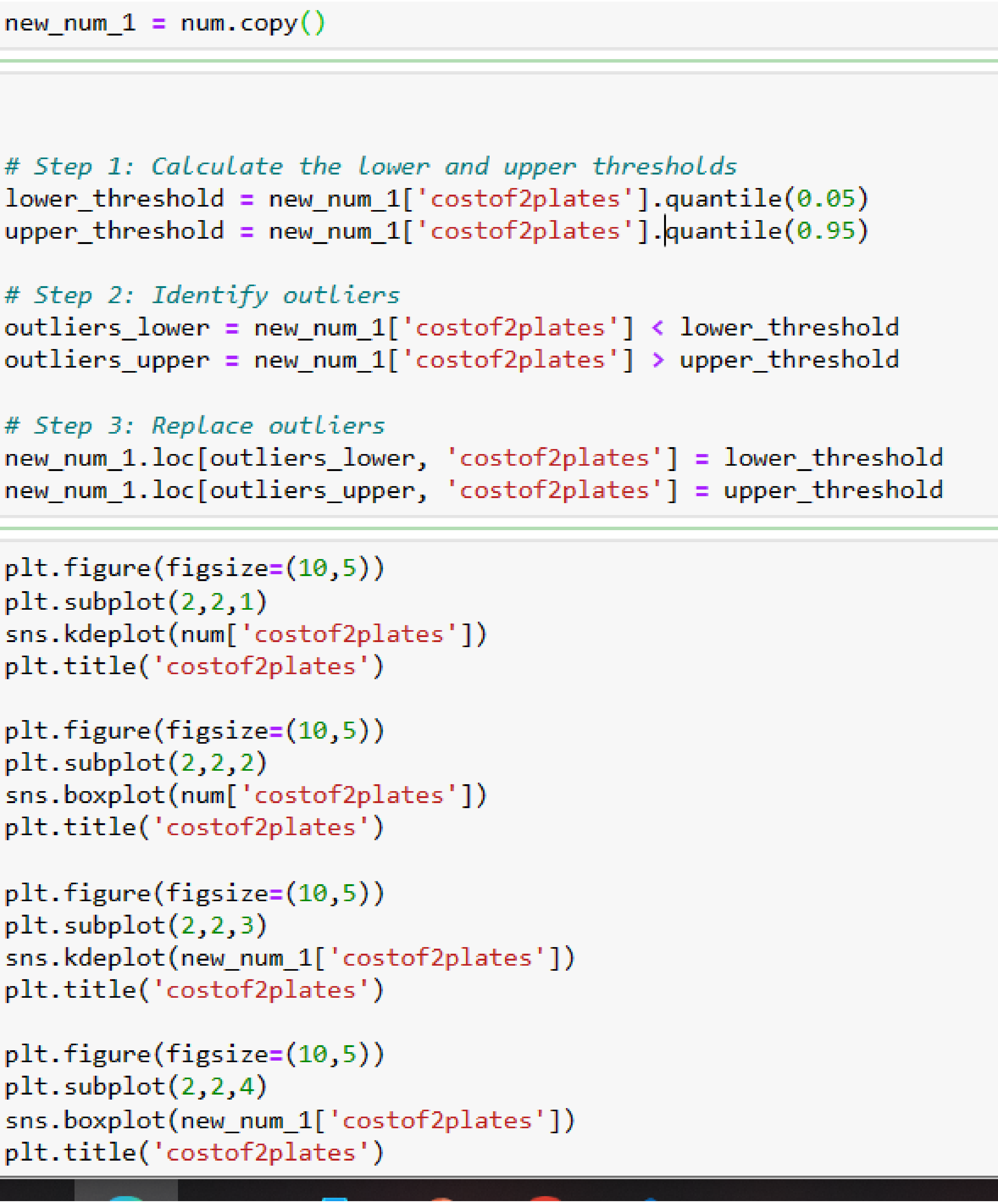
```
In [28]: plt.figure(figsize=(10,5))
```

In [28]:
```
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.kdeplot(num['costof2plates'])
plt.title('costof2plates')

plt.figure(figsize=(10,5))
plt.subplot(2,2,2)
sns.boxplot(num['costof2plates'])
plt.title('costof2plates')

plt.figure(figsize=(10,5))
plt.subplot(2,2,3)
sns.kdeplot(new_num['costof2plates'])
plt.title('costof2plates')

plt.figure(figsize=(10,5))
plt.subplot(2,2,4)
sns.boxplot(new_num['costof2plates'])
plt.title('costof2plates')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\se
ng density estimation.
  warnings.warn(msg, UserWarning)
```

Out[28]:
```
Text(0.5, 1.0, 'costof2plates')
```

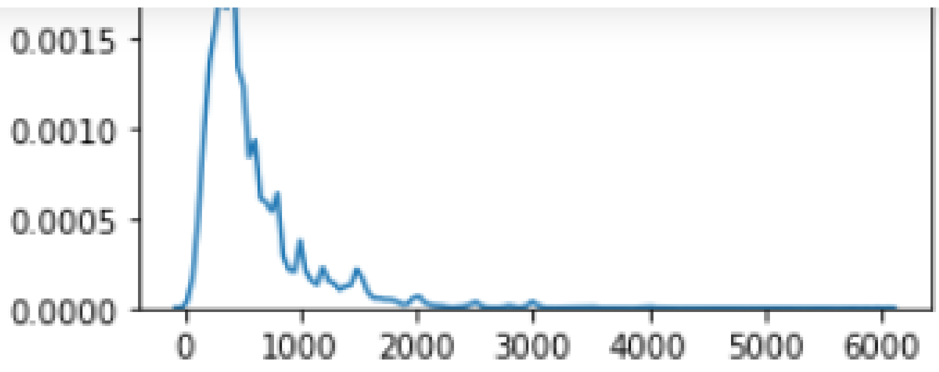


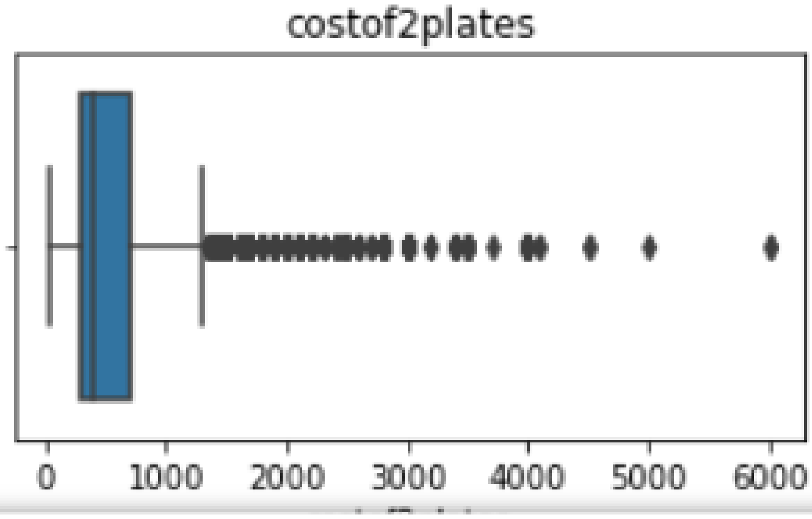


```
new_num_1 = num.copy()
```

```
# Step 1: Calculate the lower and upper thresholds
lower_threshold = new_num_1['costof2plates'].quantile(0.05)
upper_threshold = new_num_1['costof2plates'].quantile(0.95)

# Step 2: Identify outliers
outliers_lower = new_num_1['costof2plates'] < lower_threshold
outliers_upper = new_num_1['costof2plates'] > upper_threshold

# Step 3: Replace outliers
new_num_1.loc[outliers_lower, 'costof2plates'] = lower_threshold
new_num_1.loc[outliers_upper, 'costof2plates'] = upper_threshold
```

```
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.kdeplot(num['costof2plates'])
plt.title('costof2plates')

plt.figure(figsize=(10,5))
plt.subplot(2,2,2)
sns.boxplot(num['costof2plates'])
plt.title('costof2plates')

plt.figure(figsize=(10,5))
plt.subplot(2,2,3)
sns.kdeplot(new_num_1['costof2plates'])
plt.title('costof2plates')

plt.figure(figsize=(10,5))
plt.subplot(2,2,4)
sns.boxplot(new_num_1['costof2plates'])
plt.title('costof2plates')
```





2]:

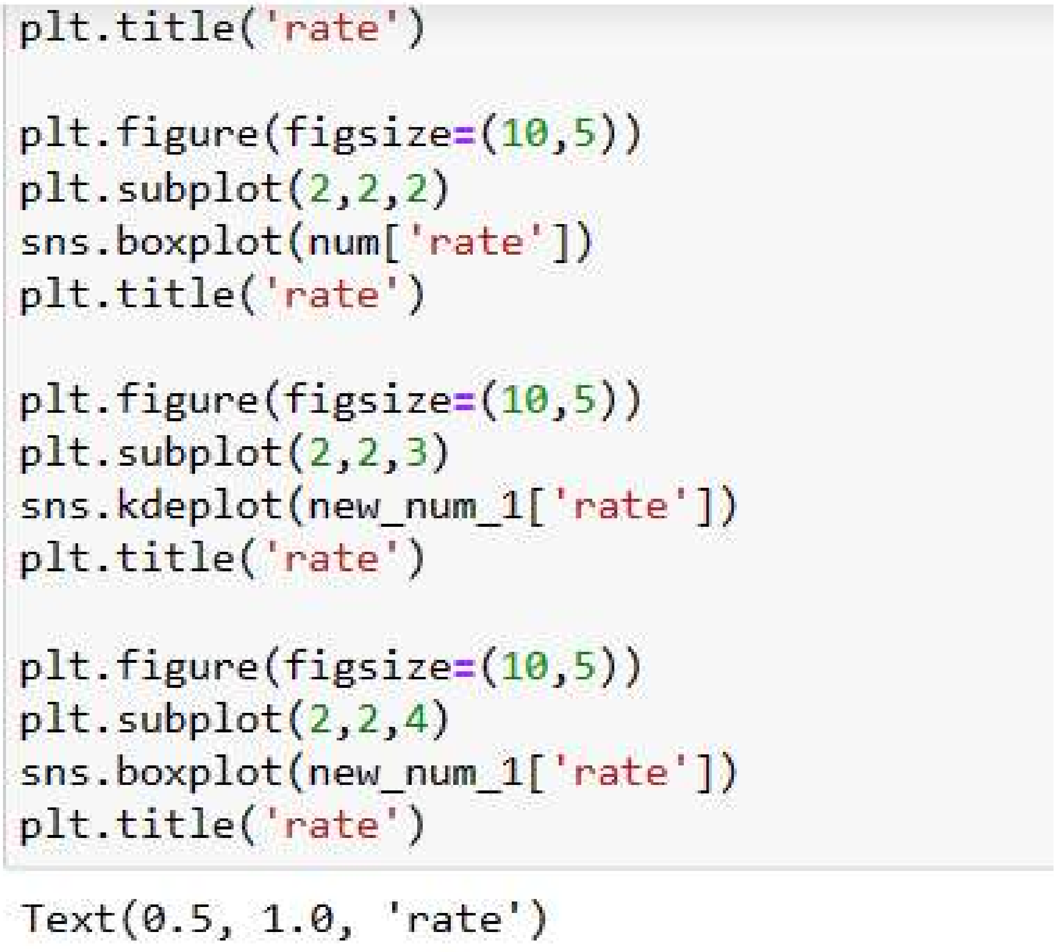
```
# handling outliers in rate column
```

3]:

```

# Step 1: Calculate the lower and upper thresholds
lower_threshold = new_num_1['rate'].quantile(0.05)
upper_threshold = new_num_1['rate'].quantile(0.95)

# Step 2: Identify outliers
outliers_lower = new_num_1['rate'] < lower_threshold
outliers_upper = new_num_1['rate'] > upper_threshold

# Step 3: Replace outliers
new_num_1.loc[outliers_lower, 'rate'] = lower_threshold
new_num_1.loc[outliers_upper, 'rate'] = upper_threshold
```

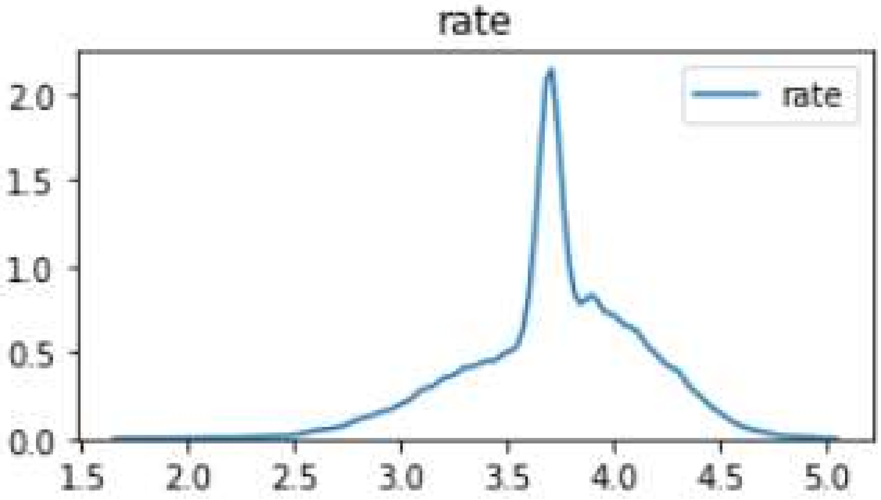
4]:

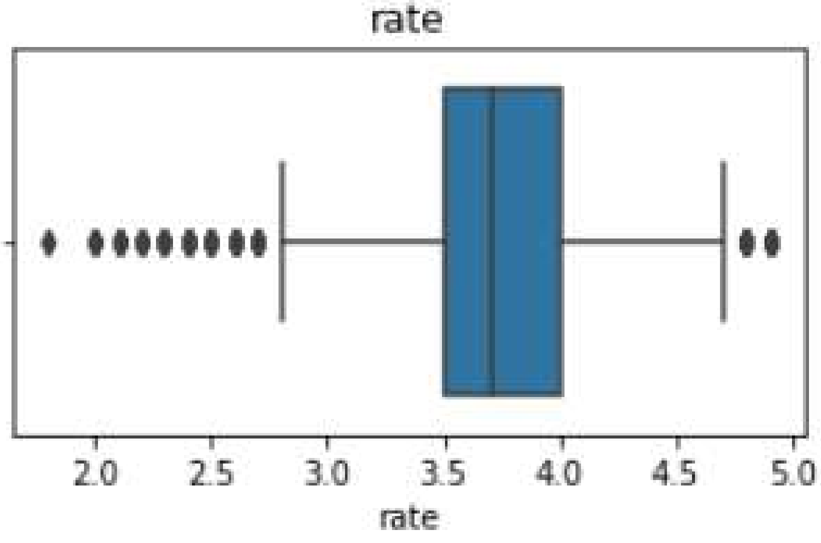
```
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.kdeplot(num['rate'])
plt.title('rate')

plt.figure(figsize=(10,5))
plt.subplot(2,2,2)
```



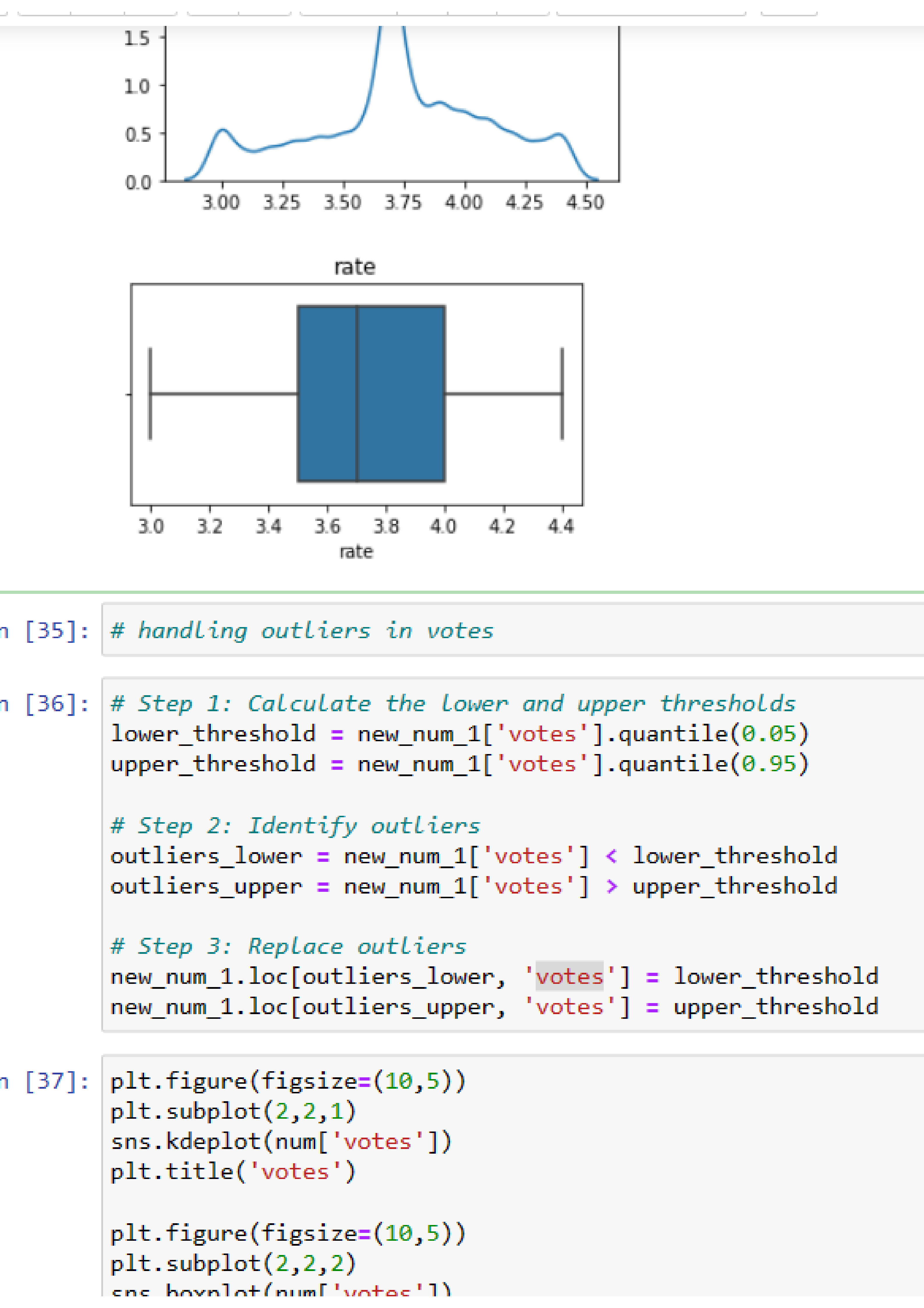
```
plt.title('rate')

plt.figure(figsize=(10,5))
plt.subplot(2,2,2)
sns.boxplot(num['rate'])
plt.title('rate')

plt.figure(figsize=(10,5))
plt.subplot(2,2,3)
sns.kdeplot(new_num_1['rate'])
plt.title('rate')

plt.figure(figsize=(10,5))
plt.subplot(2,2,4)
sns.boxplot(new_num_1['rate'])
plt.title('rate')
```

```
Text(0.5, 1.0, 'rate')
```





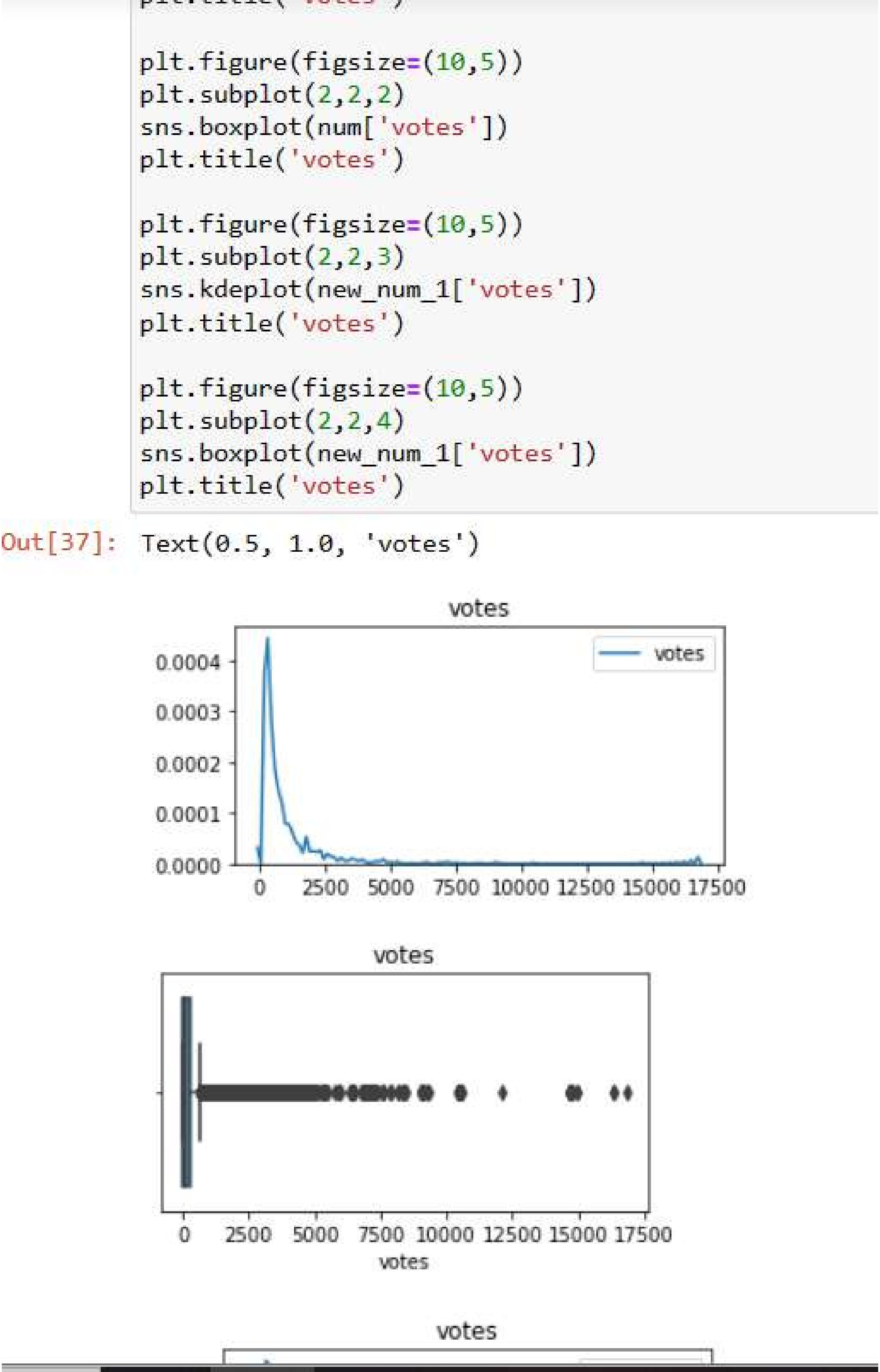
rate



```
n [35]: # handling outliers in votes
```

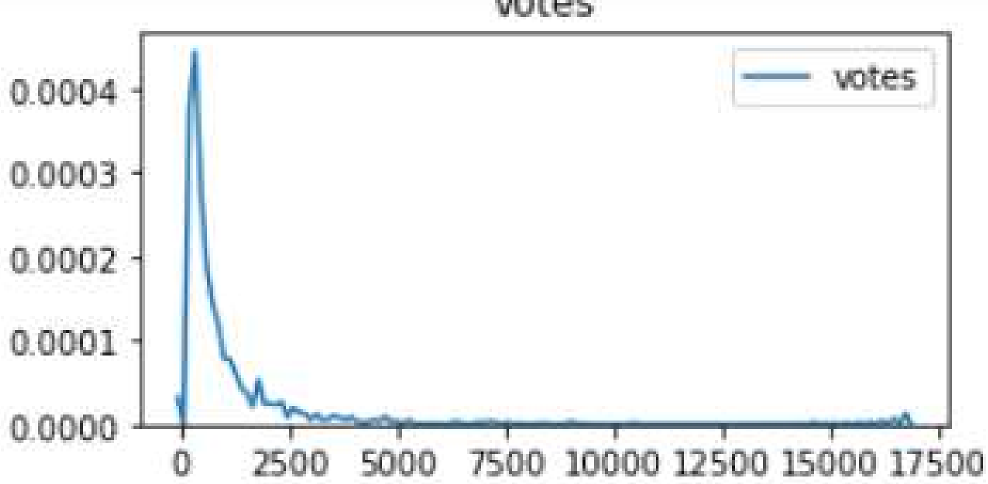
```
n [36]: # Step 1: Calculate the lower and upper thresholds
        lower_threshold = new_num_1['votes'].quantile(0.05)
        upper_threshold = new_num_1['votes'].quantile(0.95)

        # Step 2: Identify outliers
        outliers_lower = new_num_1['votes'] < lower_threshold
        outliers_upper = new_num_1['votes'] > upper_threshold

        # Step 3: Replace outliers
        new_num_1.loc[outliers_lower, 'votes'] = lower_threshold
        new_num_1.loc[outliers_upper, 'votes'] = upper_threshold
```

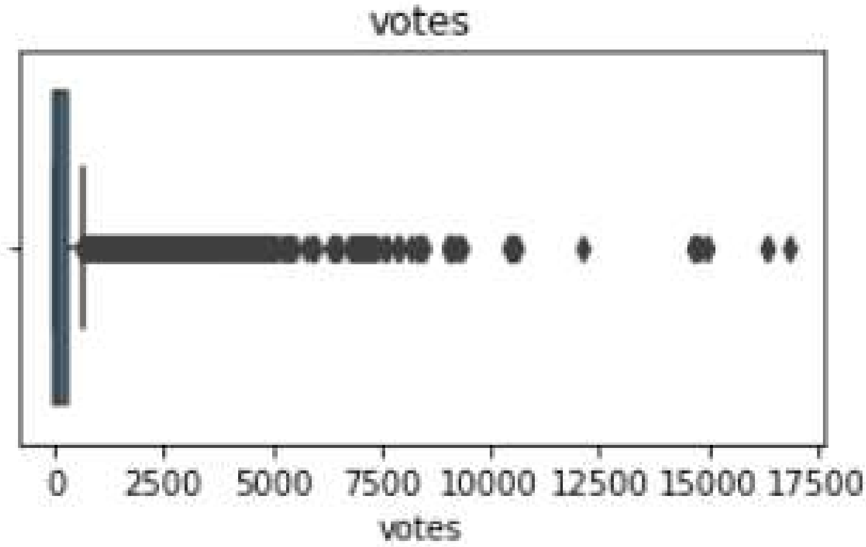
```
n [37]: plt.figure(figsize=(10,5))
        plt.subplot(2,2,1)
        sns.kdeplot(num['votes'])
        plt.title('votes')

        plt.figure(figsize=(10,5))
        plt.subplot(2,2,2)
        sns.boxplot(num['votes'])
```

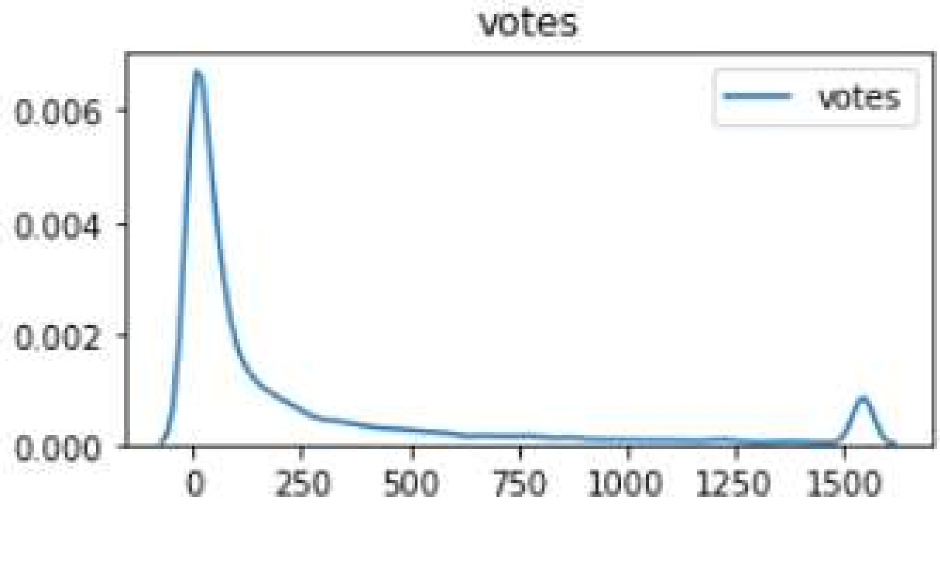


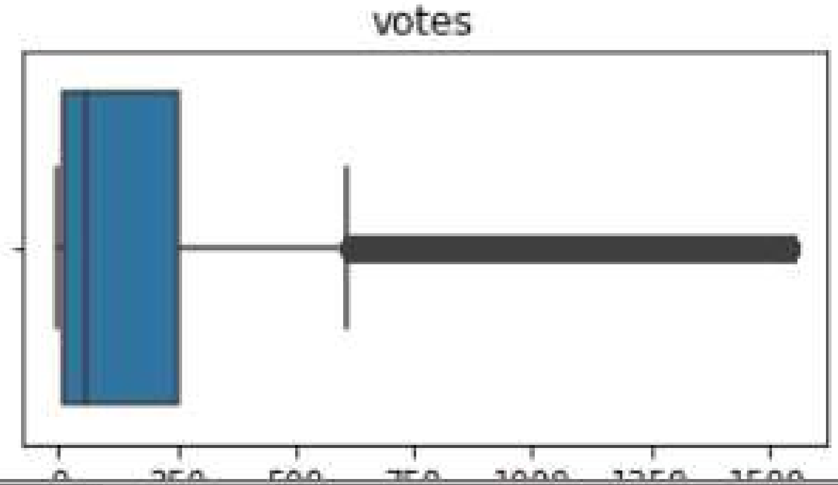
```
plt.title('votes')

plt.figure(figsize=(10,5))
plt.subplot(2,2,2)
sns.boxplot(num['votes'])
plt.title('votes')

plt.figure(figsize=(10,5))
plt.subplot(2,2,3)
sns.kdeplot(new_num_1['votes'])
plt.title('votes')

plt.figure(figsize=(10,5))
plt.subplot(2,2,4)
sns.boxplot(new_num_1['votes'])
plt.title('votes')
```

```
Out[37]: Text(0.5, 1.0, 'votes')
```


votes
votes
0.0004
0.0003
0.0002
0.0001
0.0000
0
2500
5000
7500
10000
12500
15000
17500


votes
0
2500
5000
7500
10000
12500
15000
17500
votes


votes
votes
0.006
0.004
0.002
0.000
0
250
500
750
1000
1250
1500


votes