In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

import numpy as np
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [2]:

```
# path to data containing information about sales
DATA_PATH = 'data/data.csv'
```

In [3]:

```
# Display all of the columns when data are shown
pd.set_option('display.max_columns', 60)

# Set up graphs
plt.rcParams['figure.figsize'] = (16, 8)
plt.rcParams['figure.dpi'] = 80
plt.rcParams['axes.facecolor'] = 'FFFFFF'
plt.rcParams['legend.facecolor'] = 'FFFFFF'
```

Data.csv insights

In [4]:

```
data =  pd.read_csv(DATA_PATH, sep=',', parse_dates=['doc_date', 'created_at',
'product_since'], low_memory=False)
data.head()
```

Out[4]:

	order_price_without_vat	order_price_with_vat	bill_country	setting_currency_id	create
0	562.29	674.75	BG	1	2020-(19:25:20.84
1	562.29	674.75	BG	1	2020-(19:25:20.84
2	562.29	674.75	BG	1	2020-(19:25:20.84
3	562.29	674.75	BG	1	2020-(19:25:20.84
4	562.29	674.75	BG	1	2020-(19:25:20.84

In [5]:

```
# Look onto numeric data information
data.describe()
```

Out[5]:

	order_price_without_vat	order_price_with_vat	setting_currency_id	shop_basket_id	ex
count	3.680646e+06	3.680646e+06	3.680646e+06	3.680646e+06	
mean	1.156494e+04	1.394551e+04	6.574431e+00	2.172649e+06	
std	4.545011e+05	5.374165e+05	3.087940e+00	1.399298e+06	
min	-5.154205e+04	-7.030000e+00	1.000000e+00	3.000000e+00	
25%	1.330000e+02	1.596000e+02	4.000000e+00	8.762170e+05	
50%	6.455050e+02	7.750000e+02	6.000000e+00	2.084738e+06	
75%	3.464360e+03	4.170000e+03	1.000000e+01	3.395335e+06	
max	6.645066e+08	7.508925e+08	1.500000e+01	4.713444e+06	

We got types of columns via dtypes. Used data types are following:\ int64 = integer\ float64 = float\ object = string (or mixed)\ datetime64[ns] = datetime \ bool = boolean

In [6]:

```
print(data.dtypes)
```

```
order_price_without_vat                          float64
order_price_with_vat                             float64
bill_country                                      object
setting_currency_id                                int64
created_at                                datetime64[ns]
shop_basket_id                                     int64
doc_date                                  datetime64[ns]
exchange_currency_rate                           float64
source                                            object
canceled_date                                     object
currency_code                                     object
currency_symbol                                   object
price_round_system                                 int64
basket_total_price_before_discount_with_vat      float64
basket_total_price_with_vat                      float64
count_basket_items                                 int64
basket_count_products                              int64
basket_type                                       object
item_quantity                                      int64
item_type                                         object
item_unit_price_with_vat                         float64
item_unit_price_without_vat                      float64
item_total_discount_with_vat                     float64
product_id                                         int64
product_code                                       int64
catalog_category_id                              float64
catalog_brand_id                                   int64
product_name                                      object
product_status                                    object
reviews_count                                      int64
reviews_average_score_price                      float64
reviews_average_score_quality                    float64
reviews_average_score_properties                 float64
reviews_average_score_overall                    float64
reviews_average_score                            float64
is_in_stock                                       object
is_ended                                          object
is_new                                            object
is_boosted                                        object
product_purchase_price                           float64
eshop_stock_count                                float64
is_fifo                                           object
product_name_parameterize                         object
product_since                             datetime64[ns]
category                                          object
tree_path                                         object
category_name_parameterized                       object
category_status                                   object
catalog_segment_id                               float64
categories_ancestor_ids                           object
categories_descendant_ids                         object
category_full_name_path                           object
default_warranty_period                          float64
brand_name                                        object
brand_parameterized                               object
segment_name                                      object
segment_parameterized                             object
segment_status                                    object
dtype: object
```

As we can see, most of the variables are objects - we will change them later to numeric representation suitable for machine learning.

In [7]:

```
print(data.dtypes.value_counts())
```

```
object            27
float64           18
int64             10
datetime64[ns]     3
dtype: int64
```

In [8]:

```
print(f'There is {data.shape[0]} rows over {data.shape[1]} columns in datafram
e.')
```

```
There is 3680646 rows over 58 columns in dataframe.
```

We are looking into data shown on graphs below. \ Value distribution in columns is usually normalized, which means every value is in the range of <0,1> and sum of those values equals 1 or alternatively are shown in percentage.

Clearing data

In [9]:

```
def create_others_value_counts(threshold : float, df : pd.DataFrame, column_name
: str) -> pd.Series:
    """
    Function which groups all of the values under threshold into 'others' with p
reserving of original categories
    Args
        threshold - float in range <0,1> as a threshold for key to be marked as
 others (desired percentage / 100)
        df - desired DataFrame containing given column
        column_name - string with the exact name of a desired Series
    Returns
        pd.Series - returns Series with the new 'others' group
    """

    over_thr = df[column_name].value_counts(normalize=True).loc[lambda x : x > t
hreshold]
    under_thr = df[column_name].value_counts(normalize=True).loc[lambda x : x <=
threshold]
    merged_series =  pd.concat([over_thr, pd.Series(under_thr.sum(), index=['oth
ers'])])
    return merged_series
```

In [10]:

```
print(data.isnull().sum())
```

```
order_price_without_vat                          0
order_price_with_vat                             0
bill_country                                     0
setting_currency_id                              0
created_at                                       0
shop_basket_id                                   0
doc_date                                        93
exchange_currency_rate                           0
source                                           0
canceled_date                              3507088
currency_code                                    0
currency_symbol                                  0
price_round_system                               0
basket_total_price_before_discount_with_vat      0
basket_total_price_with_vat                      0
count_basket_items                               0
basket_count_products                            0
basket_type                                      0
item_quantity                                    0
item_type                                        0
item_unit_price_with_vat                         0
item_unit_price_without_vat                      0
item_total_discount_with_vat                     0
product_id                                       0
product_code                                     0
catalog_category_id                          24875
catalog_brand_id                                 0
product_name                                     0
product_status                                   0
reviews_count                                    0
reviews_average_score_price                      0
reviews_average_score_quality                    0
reviews_average_score_properties                 0
reviews_average_score_overall                    0
reviews_average_score                            0
is_in_stock                                      0
is_ended                                        28
is_new                                           0
is_boosted                                       0
product_purchase_price                       26474
eshop_stock_count                                0
is_fifo                                          0
product_name_parameterize                        0
product_since                                    0
category                                     24875
tree_path                                    24875
category_name_parameterized                  24875
category_status                              24875
catalog_segment_id                           24875
categories_ancestor_ids                      24875
categories_descendant_ids                    24875
category_full_name_path                      24875
default_warranty_period                      24875
brand_name                                       0
brand_parameterized                              0
segment_name                                 24875
segment_parameterized                        24875
segment_status                               24875
dtype: int64
```

If canceled_date is NOT na then the order was canceled and we don't want to have it in data. \ We are only keeping NaN canceled dates and then deleting this column (full of nans now).

In [11]:

```
data = data[data.canceled_date.isna()]
data.drop('canceled_date', inplace=True, axis=1)
```

Dropping all other NaN rows from the data dataframe, since they are in important columns.

In [12]:

```
data.dropna(inplace=True)
```

There is only one value in source column, so that coulumn might be dropped as well.

In [13]:

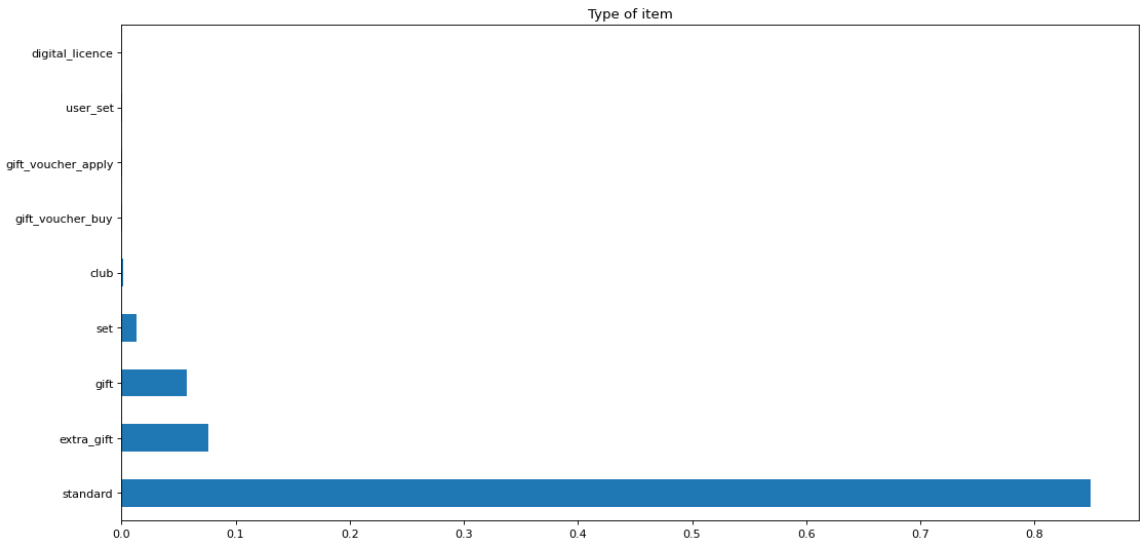
```
print(data.source.unique())
data.drop('source', inplace=True, axis=1)
```

```
['eshop']
```

Item types

In [14]:

```
figure().patch.set_facecolor('white')
data.item_type.value_counts(normalize=True).plot(kind='barh', title='Type of ite
m')
plt.show()
```



In [15]:

```
# We want to drop gifts as well as extra_gifts from orders, as they would mess u
p with machine learning (price of gifted product is usually 0).
# We are dropping club purchases for the same reason - those are gifts in a way,
not necessarily purchases for money, so price is 0 again.

print(data[data.item_type == 'gift'].sort_values(by='item_unit_price_with_vat',
ascending=False).item_unit_price_with_vat)

data = data[data.item_type != 'gift']
data = data[data.item_type != 'extra_gift']
data = data[data.item_type != 'club']

# Voucher buys and applies are dropped as well, since those are only other ways
 of payment method, not products.
data = data[data.item_type != 'gift_voucher_buy']
data = data[data.item_type != 'gift_voucher_apply']

# Merge set and user set, they work on the same principle.
data.item_type.replace('user_set', 'set', inplace=True)
```

```
2949       0.0
2284144    0.0
2284098    0.0
2284099    0.0
2284100    0.0
          ...
983600     0.0
983601     0.0
983602     0.0
983603     0.0
3671371    0.0
Name: item_unit_price_with_vat, Length: 200043, dtype: float64
```

In [16]:

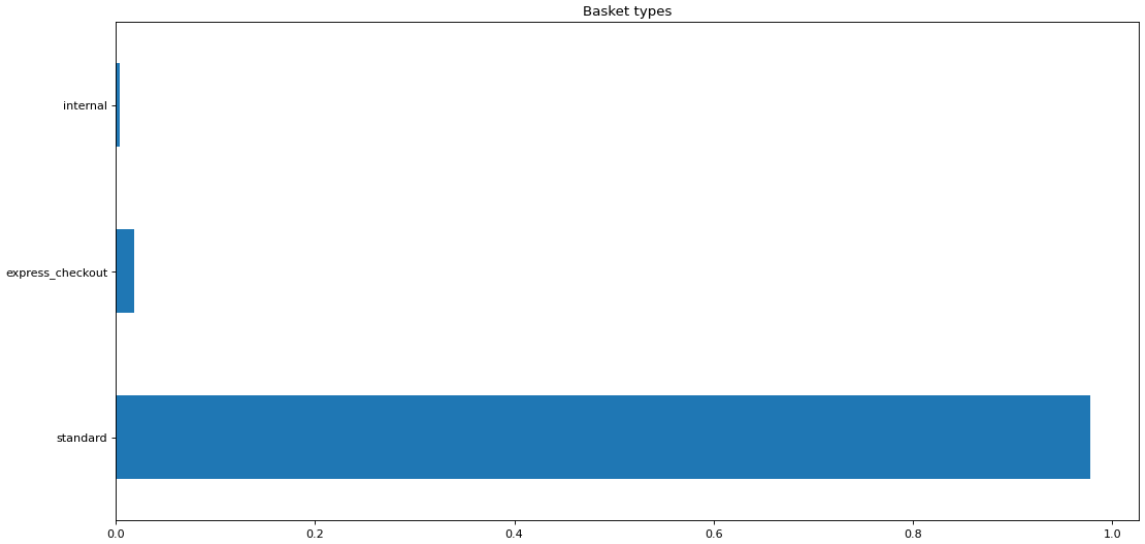
```
# We have only three types of items remaining now - standard, set and digital li
cence. All of those are classic sale types.
print(data.item_type.value_counts(normalize=True))
```

```
standard           0.984501
set                0.015455
digital_licence    0.000043
Name: item_type, dtype: float64
```

Basket types

In [17]:

```
# Three different basket types. All of them are valid, made by customer - intern
al means some internal change (after the order was accepted) made thanks to use
r's request.
figure().patch.set_facecolor('white')
data.basket_type.value_counts(normalize=True).plot(kind='barh', title='Basket ty
pes')
plt.show()
```

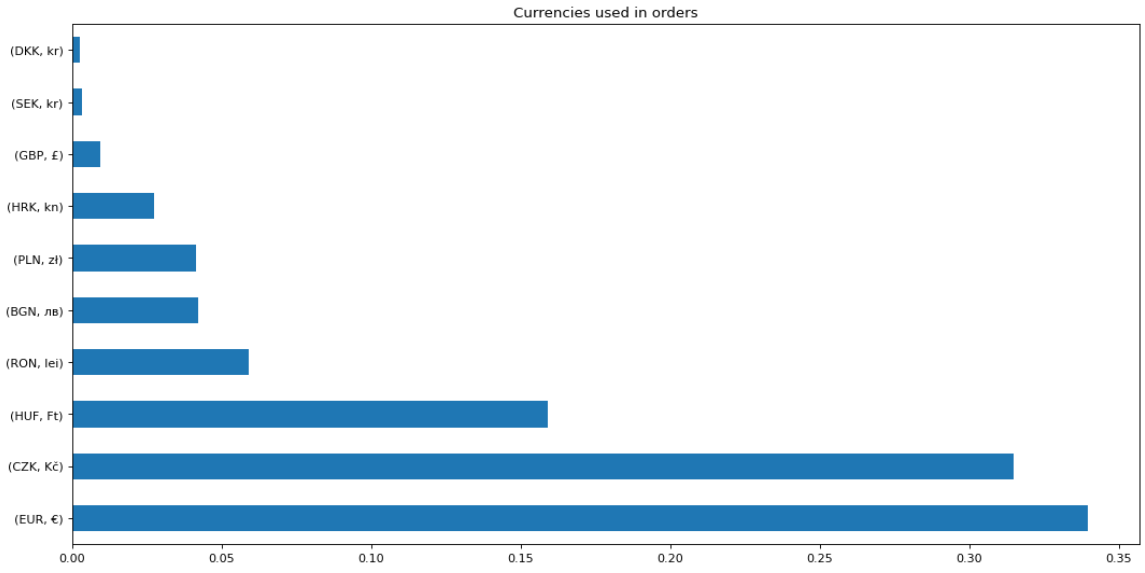


Currency

We would like to see what currency was the most used. All values will be convertet to EURos, but original, used, currencies will be still saved in the DataFrame.

In [18]:

```
data[['currency_code', 'currency_symbol']].value_counts(normalize=True).plot(kin
d='barh', title='Currencies used in orders', xlabel='')
plt.show()
```



Change all prices to EUR so we can easily compare them.

In [19]:

```
non_eur_columns = ['order_price_without_vat', 'order_price_with_vat', 'basket_to
tal_price_before_discount_with_vat',
                   'basket_total_price_with_vat', 'item_unit_price_with_vat', 'i
tem_unit_price_without_vat', 'item_total_discount_with_vat']

for column_name in non_eur_columns:
    data[column_name] =  round(data[column_name] / data.exchange_currency_rate,
2)
```

In [20]:

```
# Rename currency code column so it's clear they (original currencies) are not i
n use anymore
data.rename(columns={'currency_code': 'original_currency_code'}, inplace=True)

# Drop column that is not useful anymore
data.drop('currency_symbol', inplace=True, axis=1)
data.drop('price_round_system', inplace=True, axis=1)
```

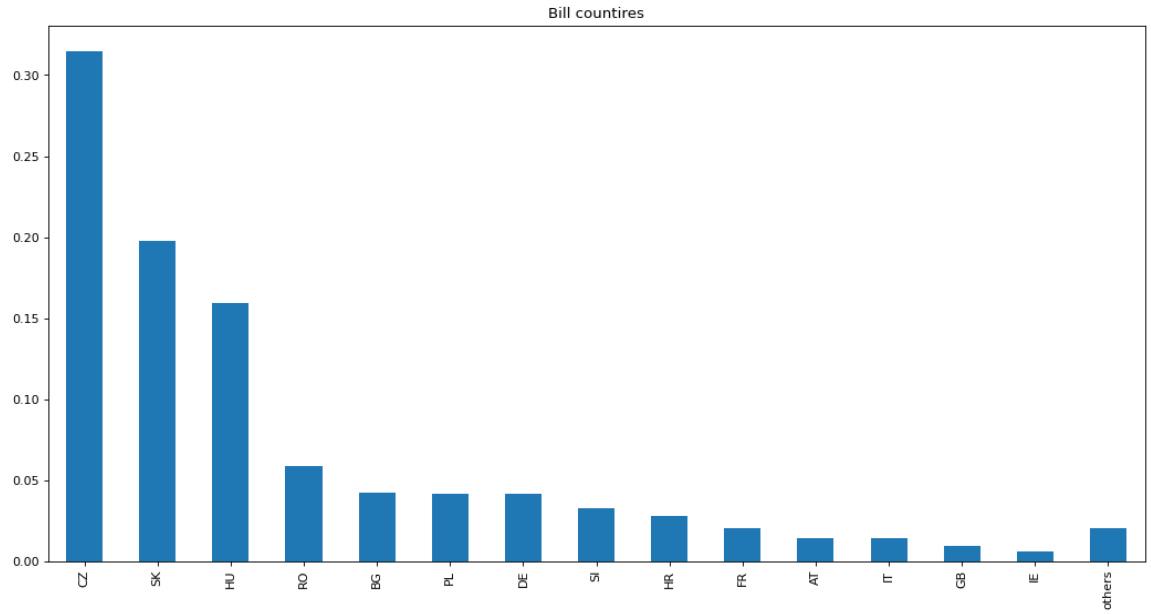
Countries

Percentage of bill countries, with the threshold of 0.5% to be shown as bill country of an order.

In [21]:

```
print(f'Bill countries in data: {data.bill_country.unique()}.\n That means {len
(data.bill_country.unique())} countries in total used as bill countries.')

figure().patch.set_facecolor('white')
counted_countries = create_others_value_counts(0.005, data, 'bill_country')
counted_countries.plot(kind='bar', title='Bill countires')
plt.show()
```

```
Bill countries in data: ['BG' 'CZ' 'SK' 'PL' 'DE' 'SI' 'RO' 'FR' 'A
T' 'HU' 'HR' 'IT' 'NL' 'DK'
 'SE' 'BE' 'LT' 'GB' 'LV' 'IE' 'CH' 'PT' 'ES' 'LU' 'FI' 'EE' 'GR' 'E
L'
 'TR' 'UA' 'RS' 'BA'].
 That means 32 countries in total used as bill countries.
```



More info about products

In [22]:

```
true_false_columns = ['is_in_stock', 'is_ended', 'is_new', 'is_boosted', 'is_fif
o']
data.replace(['t', 'f'], [True, False], inplace=True)   # replace T/F with boole
an values
data[true_false_columns]
```

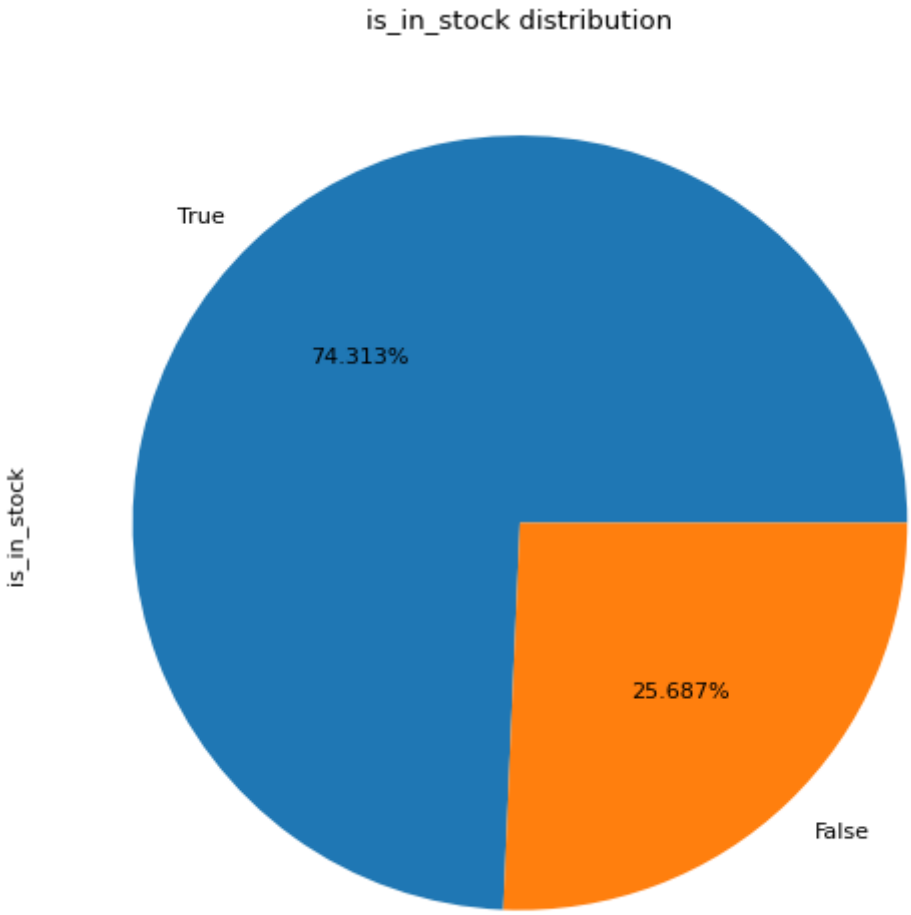
Out[22]:

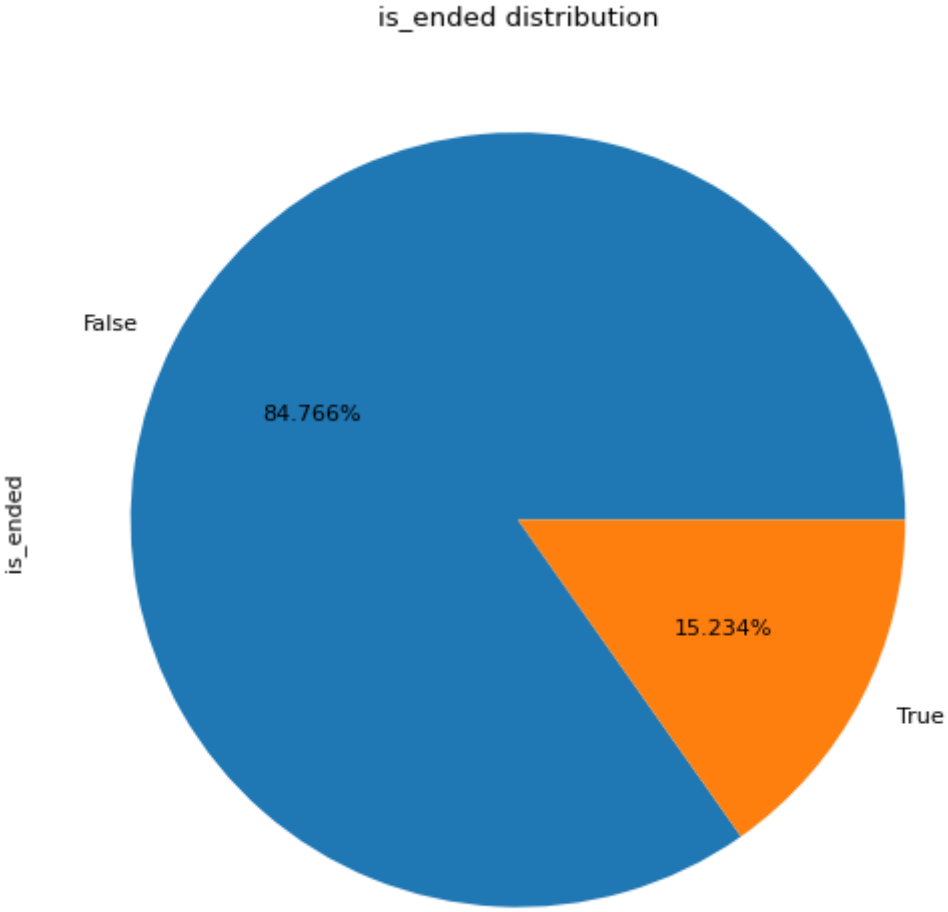
	is_in_stock	is_ended	is_new	is_boosted	is_fifo
0	True	False	False	False	False
1	False	False	False	False	False
2	True	False	False	False	False
3	True	False	False	False	False
4	True	True	False	False	False
...	...	...	...	...	...
3680641	True	False	False	False	False
3680642	False	True	False	False	False
3680643	True	False	False	False	False
3680644	False	True	False	False	False
3680645	True	False	False	False	False

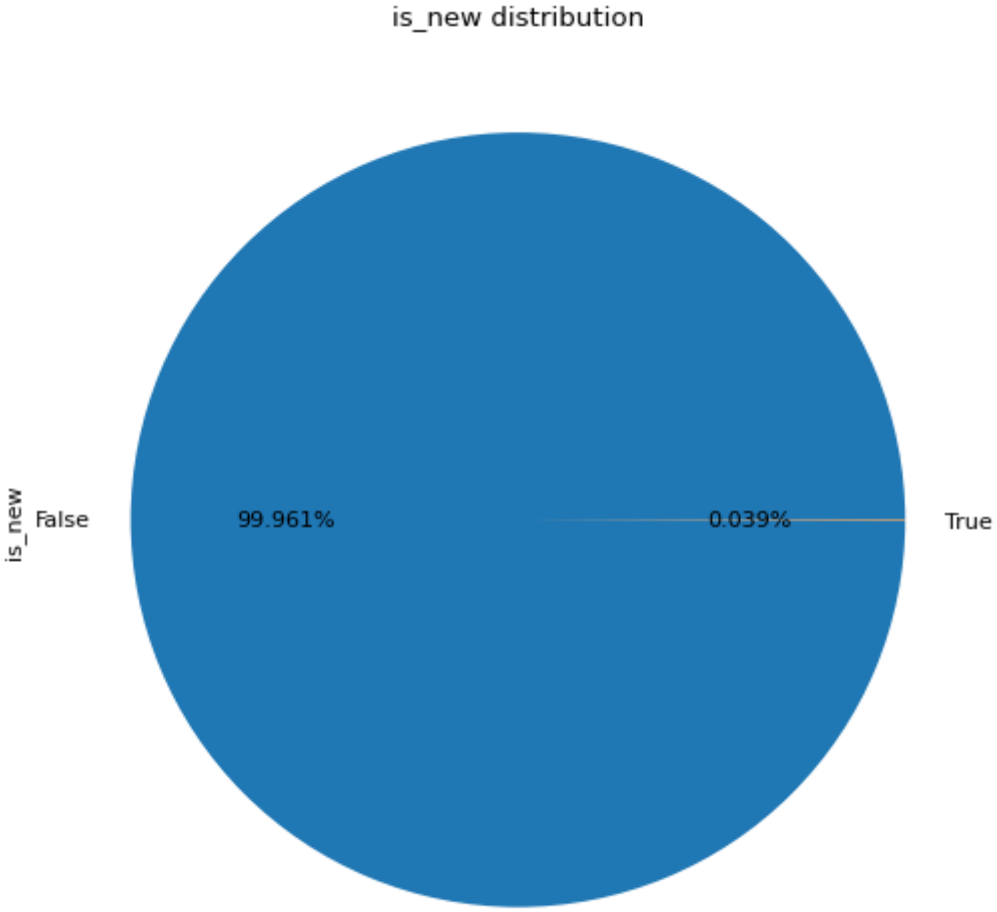
3003435 rows × 5 columns

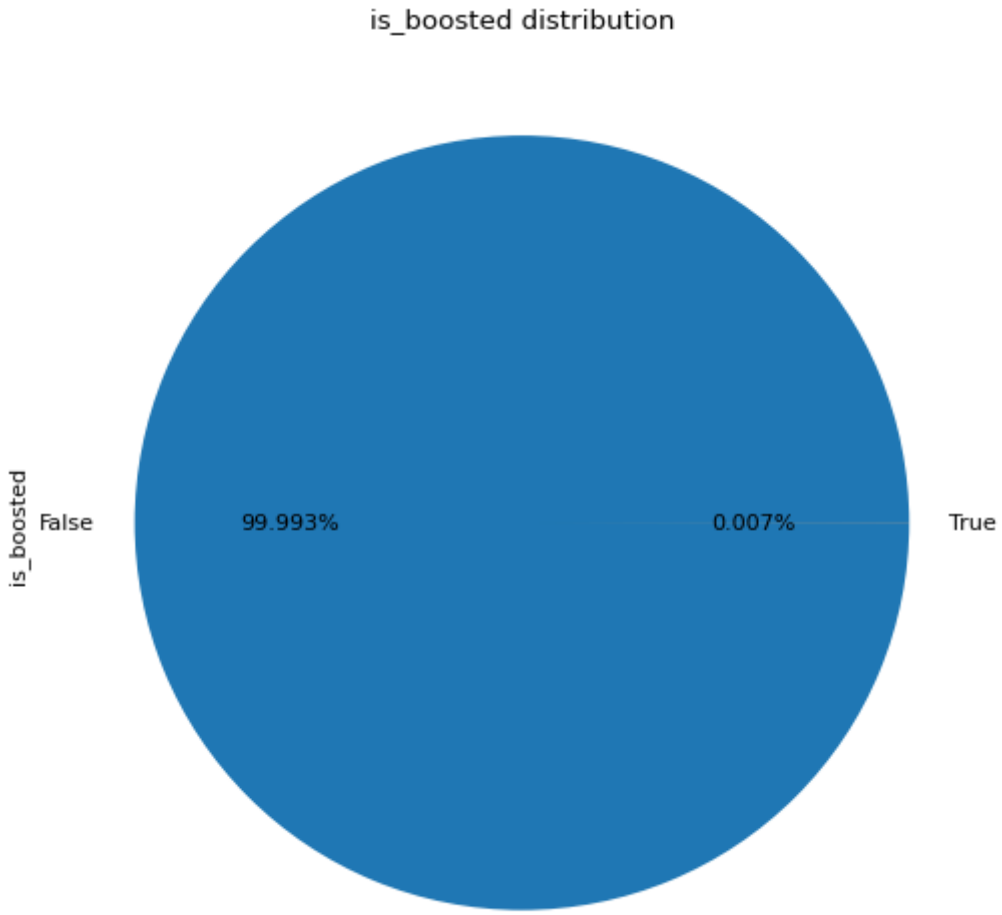
In [23]:

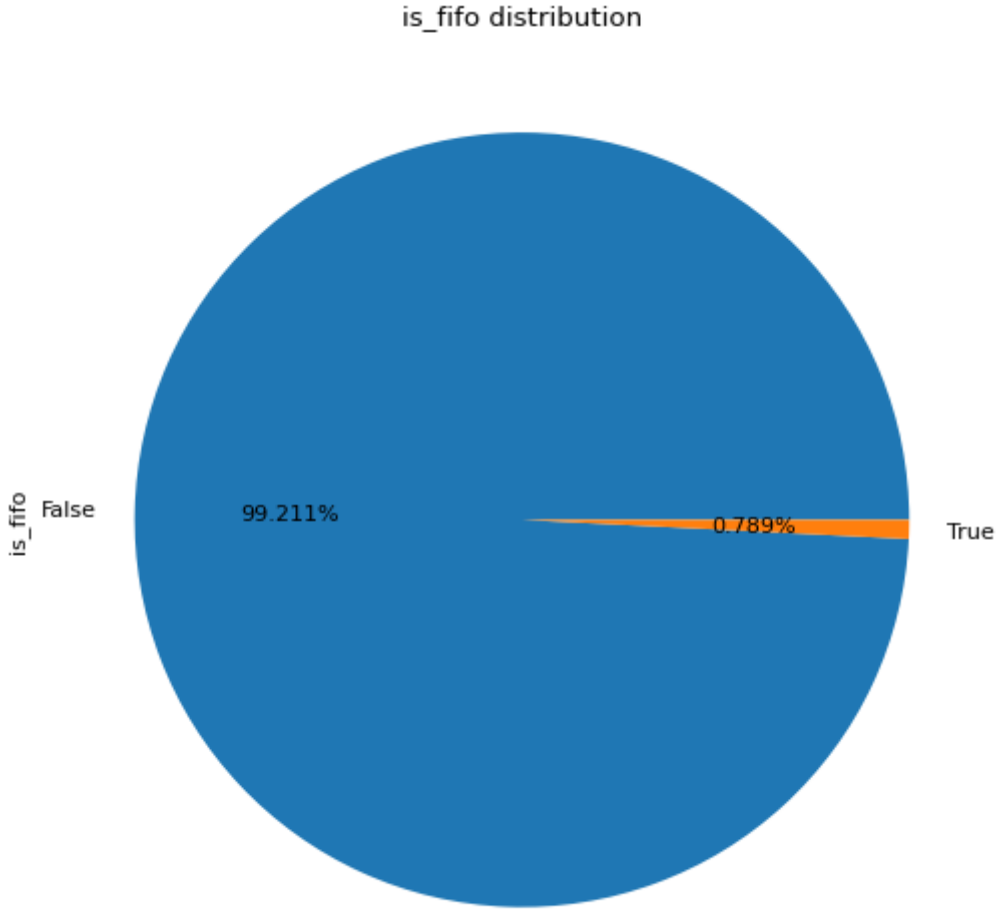
```
for col in true_false_columns:
    figure().patch.set_facecolor('white')
    data[col].value_counts().plot(kind='pie', title=f'{col} distribution', autop
ct='%1.3f%%')
    plt.show()
```


is_in_stock distribution
True
74.313%
is_in_stock
25.687%
False


is_ended distribution
is_ended
False
84.766%
15.234%
True


is_new distribution
is_new
False
99.961%
0.039%
True


is_boosted distribution
is_boosted
False
99.993%
0.007%
True


is_fifo distribution
is_fifo
False
True
99.211%
0.789%

We can see several True/False information about products, which might be usefull in the future.

For example we want to use ended product sales for model training, but not for evaluating the prediciton (as there are not new sales of ended products). \ Also FIFO products are important to note, as those are usually the products with shorter best before date. \ Boost and new are interesting as well, but there is probably too little amount of them to make significant difference (we will see later).

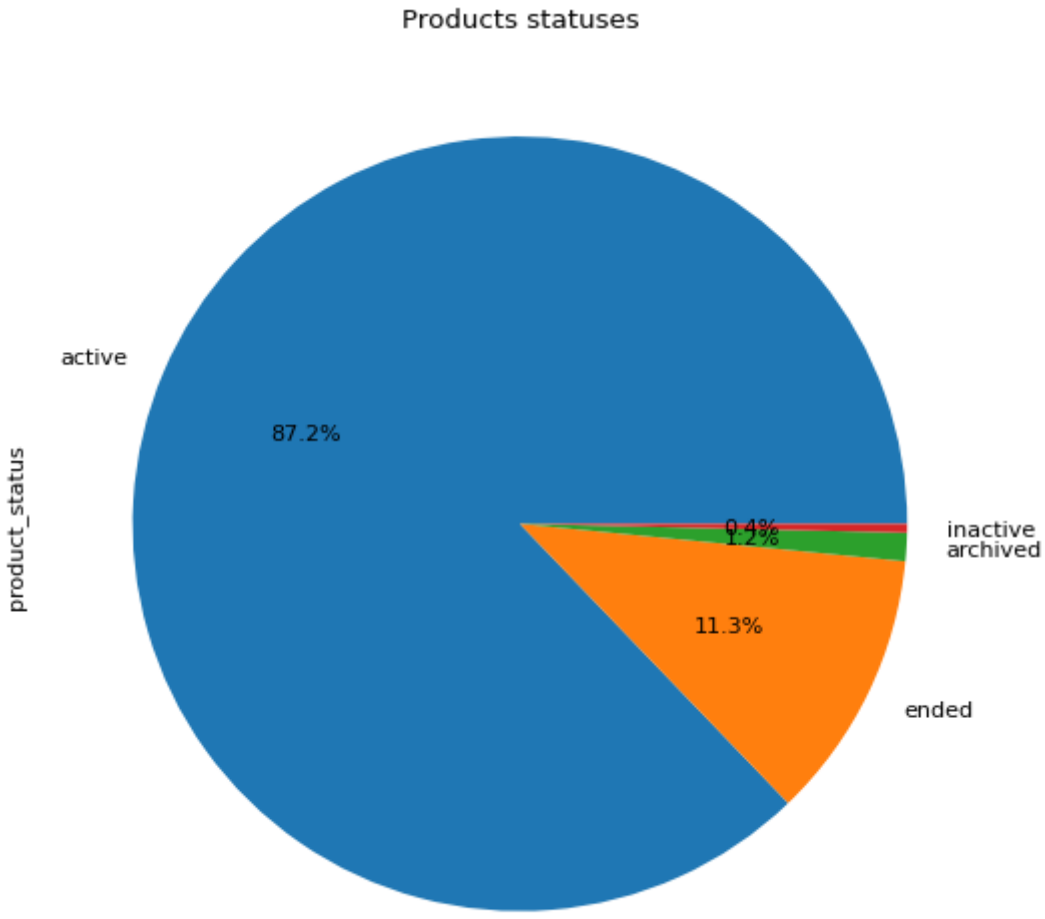
In [24]:

```
print(data[data.product_status == 'ended'].is_ended.unique())
```

```
[ True]
```

In [25]:

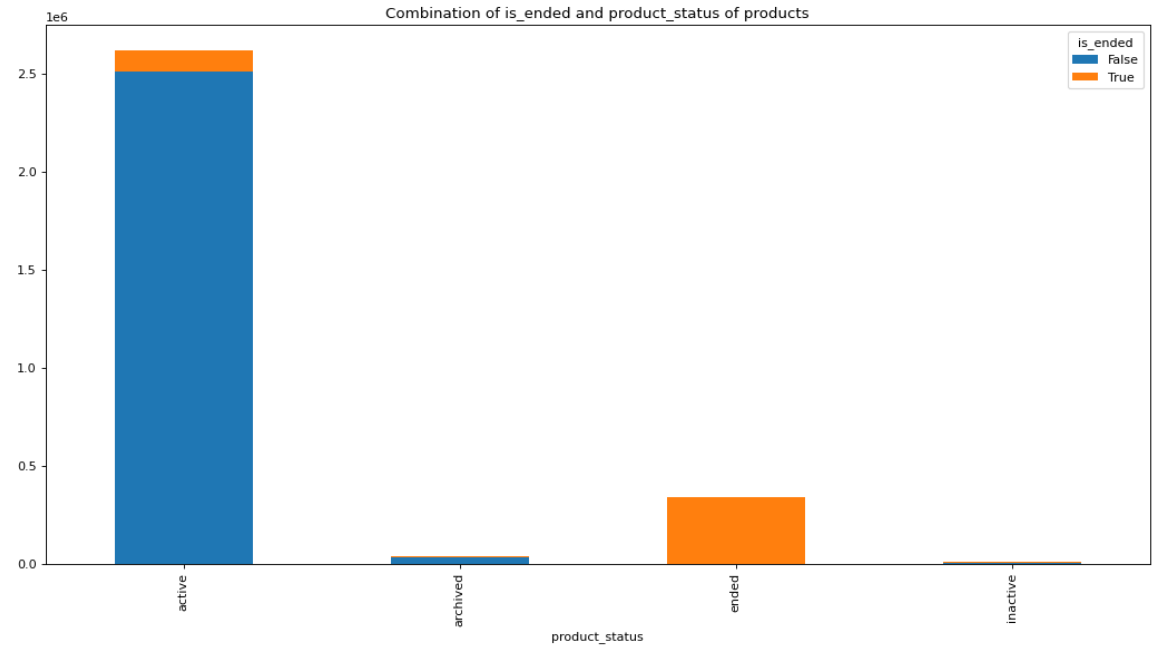
```
figure().patch.set_facecolor('white')
data.product_status.value_counts().plot(kind='pie', autopct='%1.1f%%', title='Pr
oducts statuses')
plt.show()
```



If product status is ended, then is_ended is always True. \ Other combinations also occure, as shown belov - for example active but is_ended product mean that it is still available, but new pieces are not pruchased.

In [26]:

```
pd.crosstab(columns=data['is_ended'], index=data['product_status']).plot.bar(sta
cked=True, title='Combination of is_ended and product_status of products')
plt.show()
```



Normalized data just so we can see that there is multiple combinations of status, ended and stock.

In [27]:

```
print(data[['product_status','is_ended', 'is_in_stock']].value_counts(normalize=
True))
```

```
product_status  is_ended  is_in_stock
active          False     True          0.708066
                          False         0.129193
ended           True      False         0.112778
active          True      True          0.034537
archived        False     False         0.010055
inactive        True      False         0.002901
archived        True      False         0.001634
inactive        True      True          0.000413
                False     False         0.000305
archived        True      True          0.000078
inactive        False     True          0.000035
archived        False     True          0.000006
dtype: float64
```

Below are top sold products alongside with their sale share. The share is, of course, very low as our e-shop is not based on certain product.

In [28]:

```
print(data.product_name.value_counts(normalize=True).head(10))
```

```
2221 Regular Slinky                                    0.001851
351 Shape Premium Pick Medium Red Moto                 0.001773
RCL 30203 D6                                           0.001704
Scarlett Solo 3rd Generation                           0.001652
Pick 'N' Strap                                         0.001373
C-1U USB Studio Condenser Microphone                   0.001281
PA-PMP-5 50x50x5 Dark Gray                             0.001279
UM2 U-Phoria                                           0.001218
PSA1                                                   0.001169
Professional Series Instrument Cable S/A 3 m Black     0.001162
Name: product_name, dtype: float64
```

In [29]:

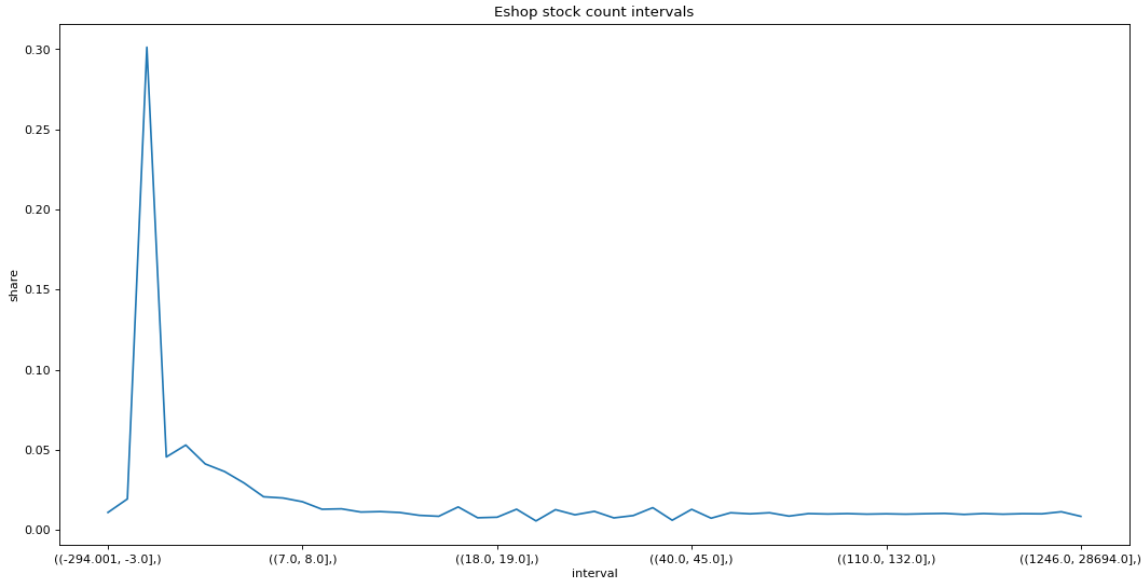
```
print(f'Amount of unique products sold {len(data.product_name.unique())}.')
```

```
Amount of unique products sold 163510.
```

In [30]:

```
# Let's lok how the most common eshop stock count looks like - normalized numbers in value_counts.\
# Numbers lower than 0 mean that item was ordere even though it is not in stock currently.
pd.DataFrame(pd.qcut(np.array(data.eshop_stock_count), q=100, duplicates='drop')).value_counts(sort=False, normalize=True).plot(title='Eshop stock count intervals', xlabel='interval', ylabel='share')
plt.plot()

print(data.eshop_stock_count.value_counts(normalize=True).head(10))
```

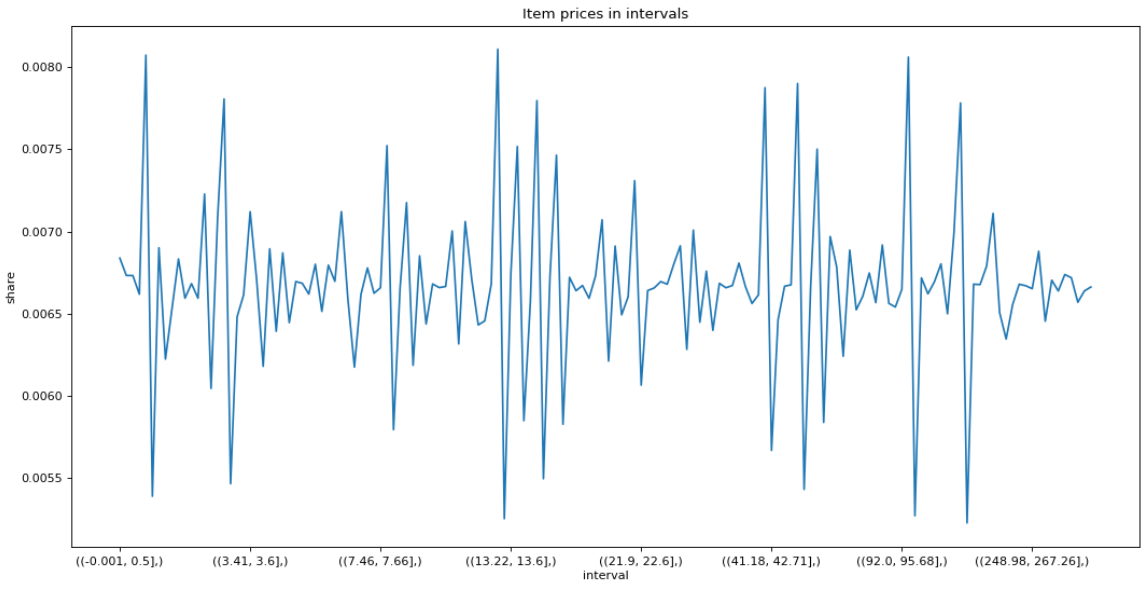
```
 0.0    0.301271
 2.0    0.052945
 1.0    0.045528
 3.0    0.041167
 4.0    0.036382
 5.0    0.029240
 6.0    0.020676
 7.0    0.019896
 8.0    0.017561
-1.0    0.014563
Name: eshop_stock_count, dtype: float64
```



In [31]:

```
pd.DataFrame(pd.qcut(np.array(data.item_unit_price_with_vat), q=150, duplicates=
'drop')).value_counts(normalize=True, sort=False).plot(title='Item prices in int
ervals', xlabel='interval', ylabel='share')
plt.plot()

print('The most popular selling prices: ')
print(data.item_unit_price_with_vat.value_counts(normalize=True).head(10))

print(f'\nPrice distribution of products varies, from very cheap products to ver
y expensive ones - {max(data.item_unit_price_with_vat)}€ is the most expensive p
roduct.')
```

```
The most popular selling prices:
14.90    0.004689
11.90    0.004593
19.90    0.003762
13.90    0.003669
15.90    0.003629
0.69     0.003525
12.90    0.003488
17.90    0.003423
10.90    0.003362
16.90    0.003031
Name: item_unit_price_with_vat, dtype: float64

Price distribution of products varies, from very cheap products to v
ery expensive ones - 22799.0€ is the most expensive product.
```



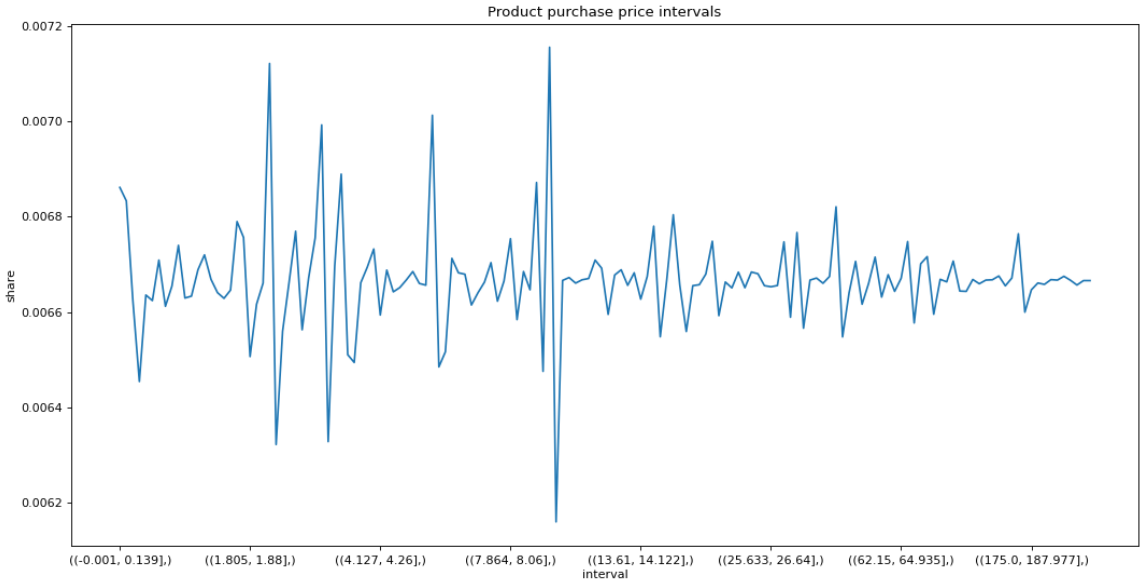
What is also important for us, not so much for the machine learning part, is the price we've purchased the product for. Let's have a look at it.

In [32]:

```
pd.DataFrame(pd.qcut(np.array(data.product_purchase_price), q=150, duplicates='d
rop')).value_counts(normalize=True, sort=False).plot(title='Product purchase pri
ce intervals', xlabel='interval', ylabel='share')
plt.plot()
```

Out[32]:

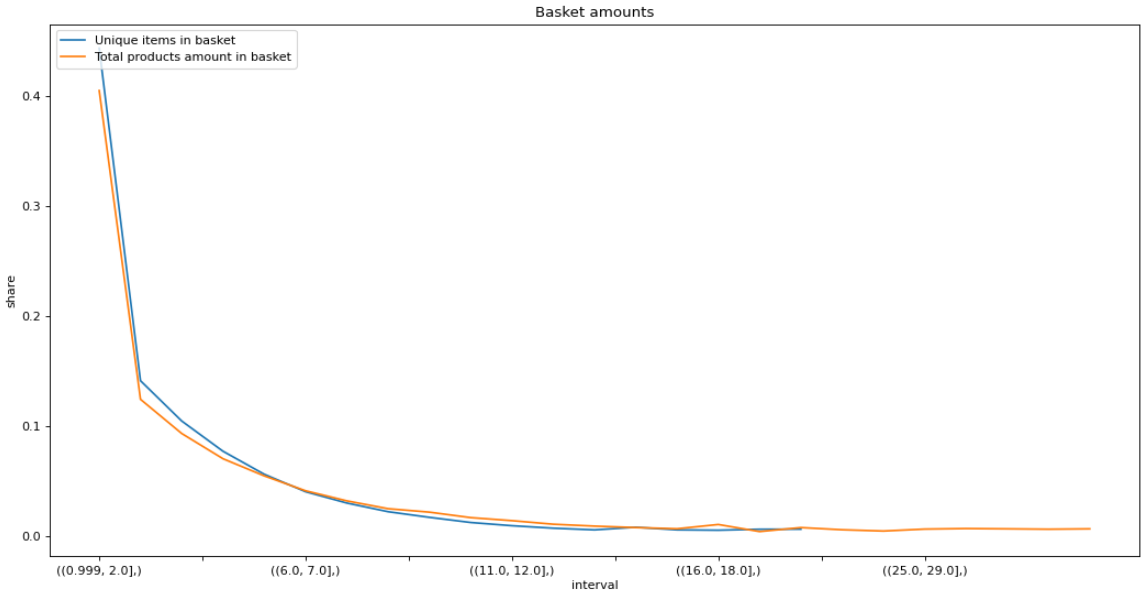
```
[]
```



In [33]:

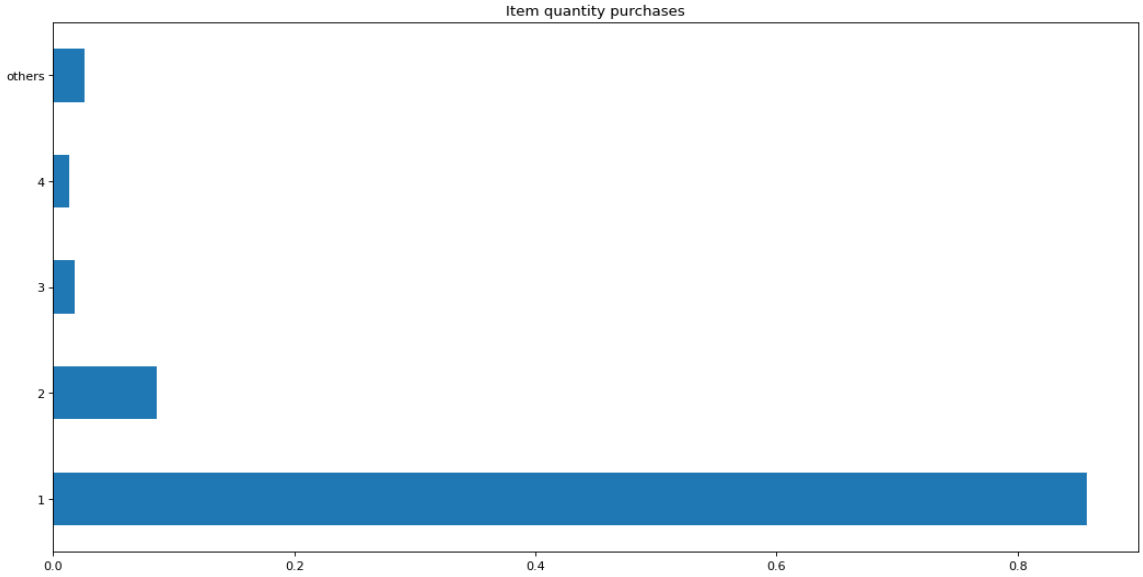
```
pd.DataFrame(pd.qcut(np.array(data.count_basket_items), q=150, duplicates='drop'
)).value_counts(normalize=True, sort=False).plot(title='Basket amounts', label=
'Unique items in basket', xlabel='interval', ylabel='share')
pd.DataFrame(pd.qcut(np.array(data.basket_count_products), q=150, duplicates='dr
op')).value_counts(normalize=True, sort=False).plot(label='Total products amount
in basket', xlabel='interval', ylabel='share')

plt.legend(loc="upper left")
plt.plot()

print(data.product_purchase_price.value_counts(normalize=True).head(10))
```

```
4.018252     0.001851
0.161053     0.001773
2.844984     0.001704
81.544000    0.001652
14.211471    0.001574
5.750000     0.001480
0.520000     0.001462
2.004322     0.001373
1.879729     0.001279
22.554966    0.001218
Name: product_purchase_price, dtype: float64
```



In [34]:

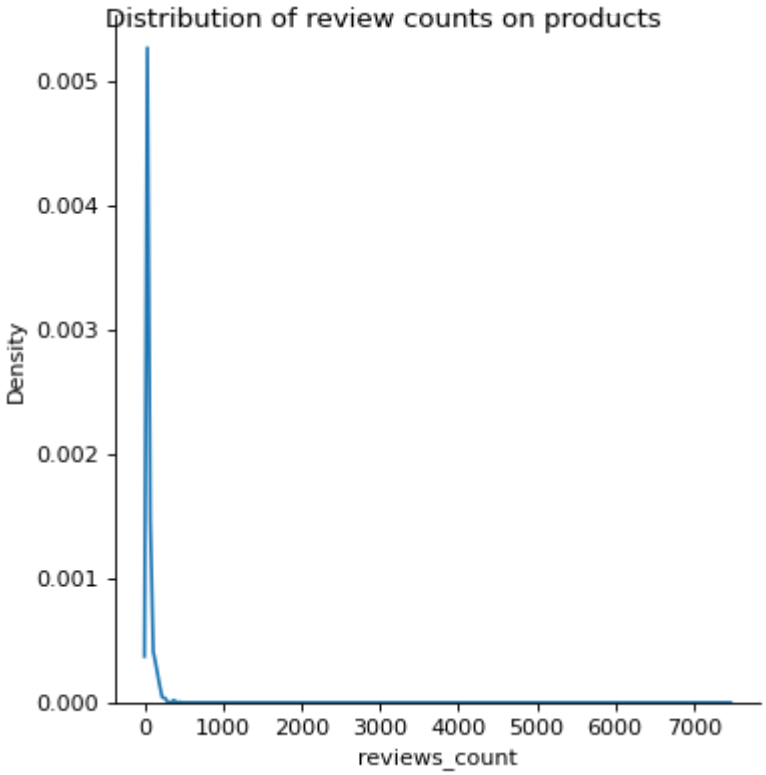
```
# We can see that item is only bought as 1 piece and with more pieces bought at
 once the occurence goes down too.
figure().patch.set_facecolor('white')
create_others_value_counts(0.01, data, 'item_quantity').plot(kind='barh', title=
'Item quantity purchases', ylabel='')
plt.show()
```



Reviews

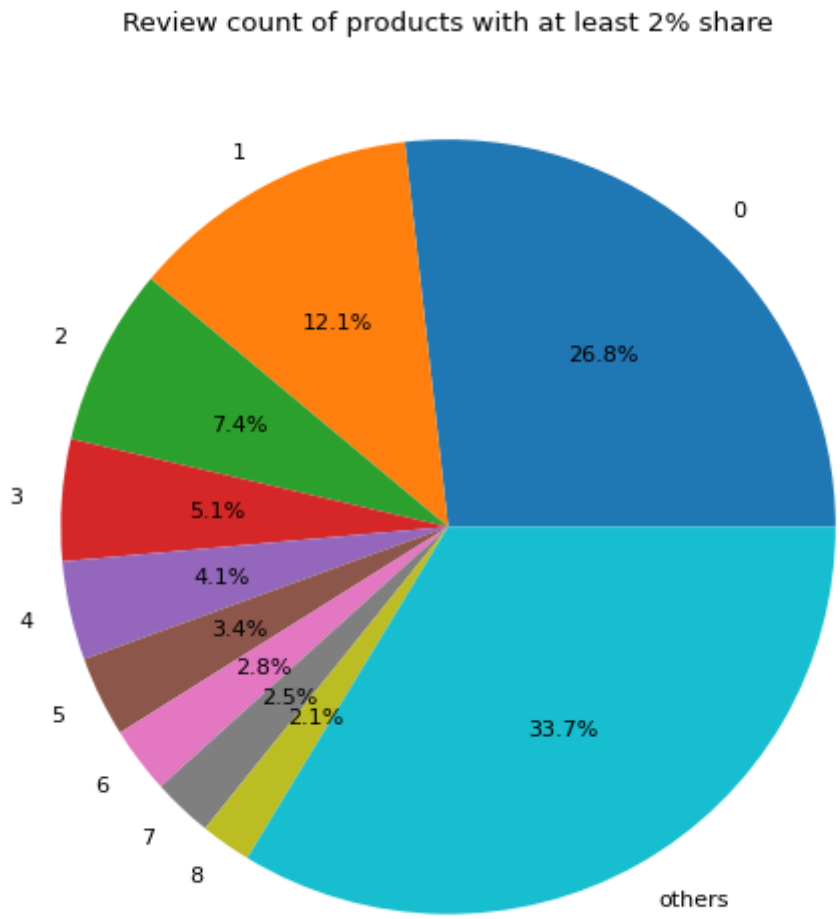
In [35]:

```
sns.displot(data.reviews_count, kind='kde').fig.suptitle('Distribution of review
counts on products')
plt.show()
```



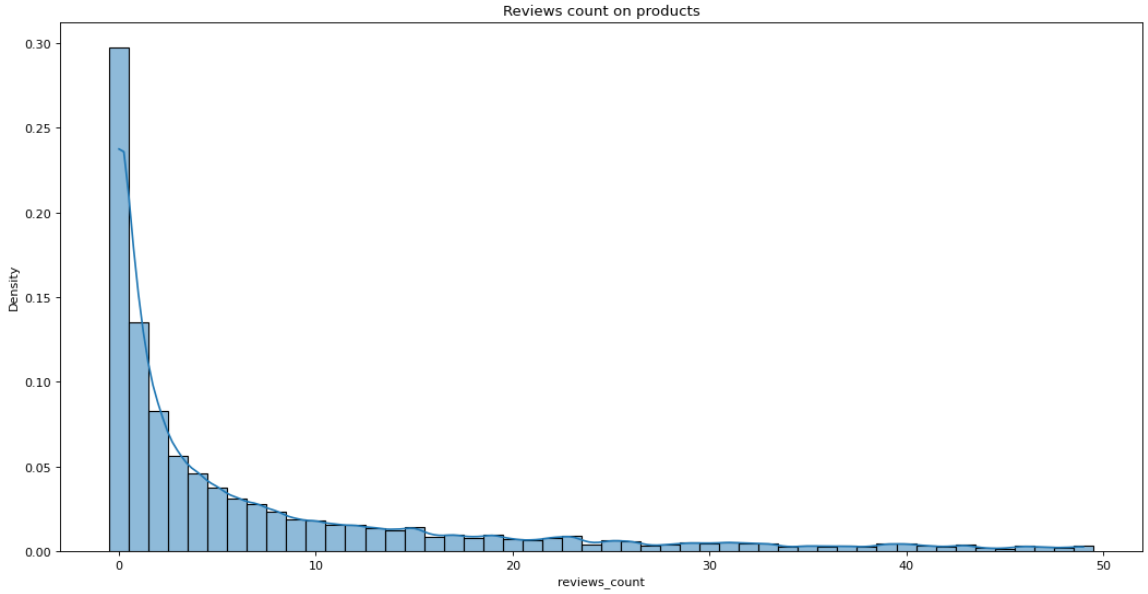
In [36]:

```
figure().patch.set_facecolor('white')
create_others_value_counts(0.02, data, 'reviews_count').plot(kind='pie', autopct
='%1.1f%%', title='Review count of products with at least 2% share', label='')
plt.show()
```


Review count of products with at least 2% share
0
1
2
3
4
5
6
7
8
others
26.8%
12.1%
7.4%
5.1%
4.1%
3.4%
2.8%
2.5%
2.1%
33.7%

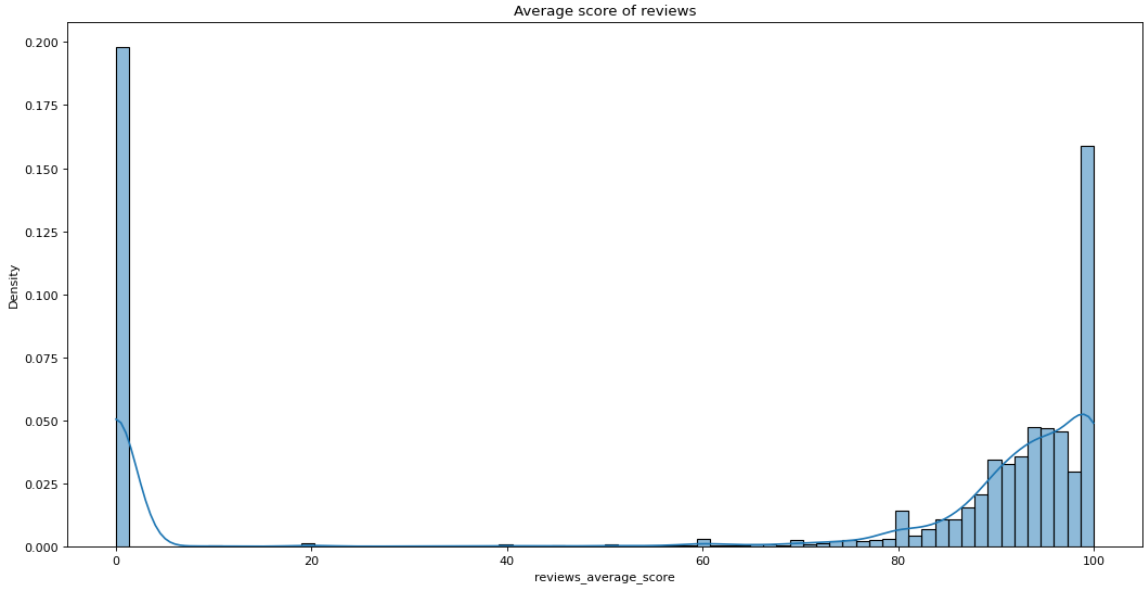
In [37]:

```
# Almost 30% of products got 0 reviews
sns.histplot(data[data.reviews_count.__lt__(50)].reviews_count, kde=True, discrete=True, stat='density').set(title='Reviews count on products')
plt.show()
```



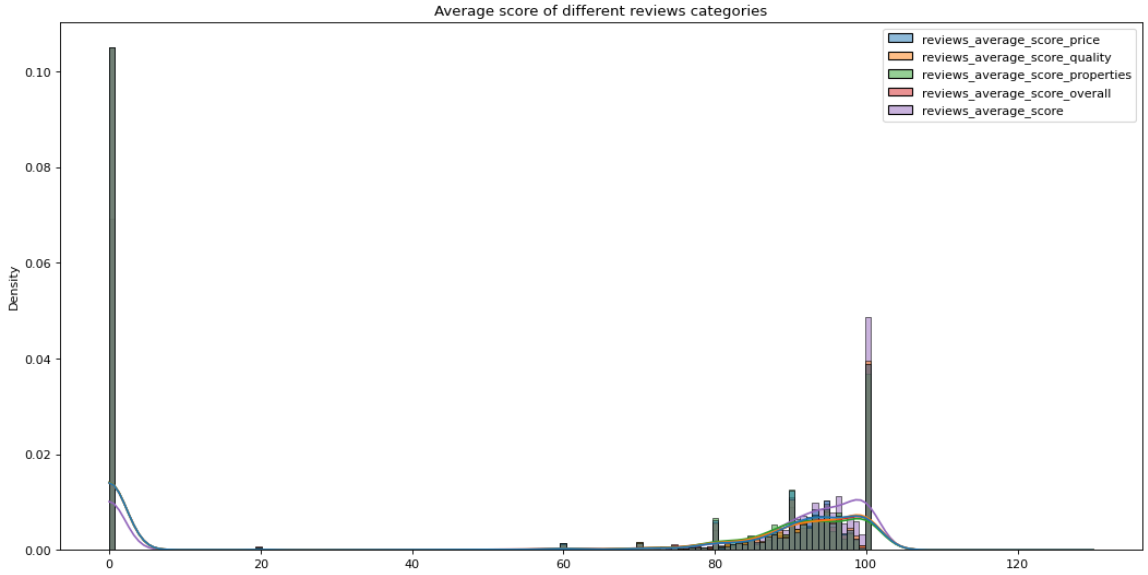
In [38]:

```
# Some of the reviews are from an old eshop, that's why we can have higher than 0 score on product with 0 reviews
sns.histplot(data.reviews_average_score, kde=True, stat='density').set(title='Average score of reviews')
plt.show()
```



In [39]:

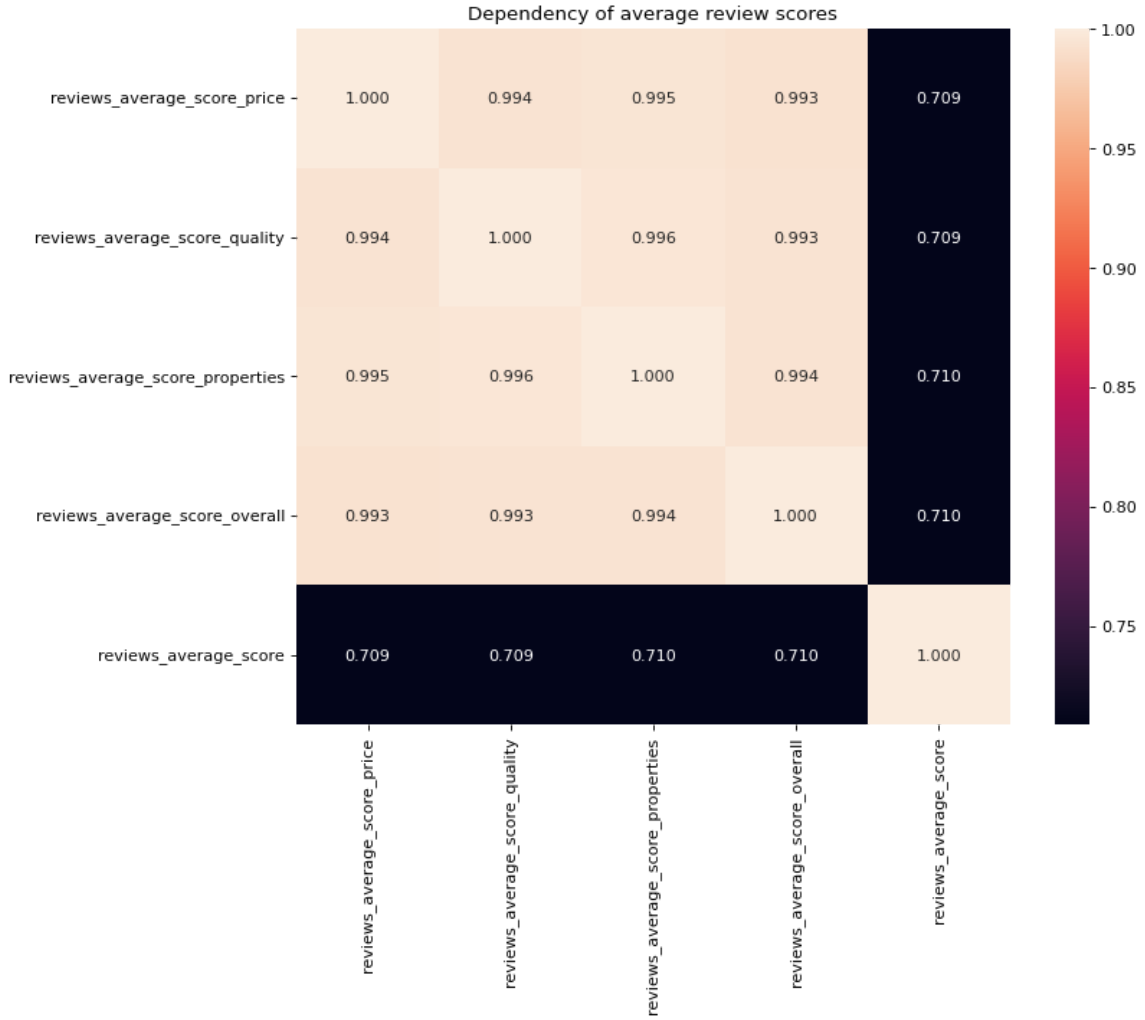
```
sns.histplot(data[['reviews_average_score_price', 'reviews_average_score_qualit
y', 'reviews_average_score_properties', 'reviews_average_score_overall', 'review
s_average_score']], kde=True, stat='density').set(title='Average score of differ
ent reviews categories')
plt.show()
```



There is many products without reviews, but if they get reviewed, it might indicate user's satisfaction with given product

In [40]:

```
review_columns = [
 'reviews_average_score_price',
 'reviews_average_score_quality',
 'reviews_average_score_properties',
 'reviews_average_score_overall',
 'reviews_average_score']

fig, ax = plt.subplots(figsize=(10,8))
sns.heatmap(data[review_columns].corr(), ax=ax, annot=True, fmt=".3f").set(title
='Dependency of average review scores')
plt.show()

review_columns.append('reviews_count')
```



Dates of orders

We have two columns to determine order date - doc_cate and created_at. We don't need precise hour, day of purchase is enough

In [41]:

```
for row in range (len(data.index)):
    if data.doc_date[0].date() != data.created_at[0].date():
        print(row)
```

No output from cycle above shows that doc_date and created_at are on the same day, we can drop created_at as it also contains hours

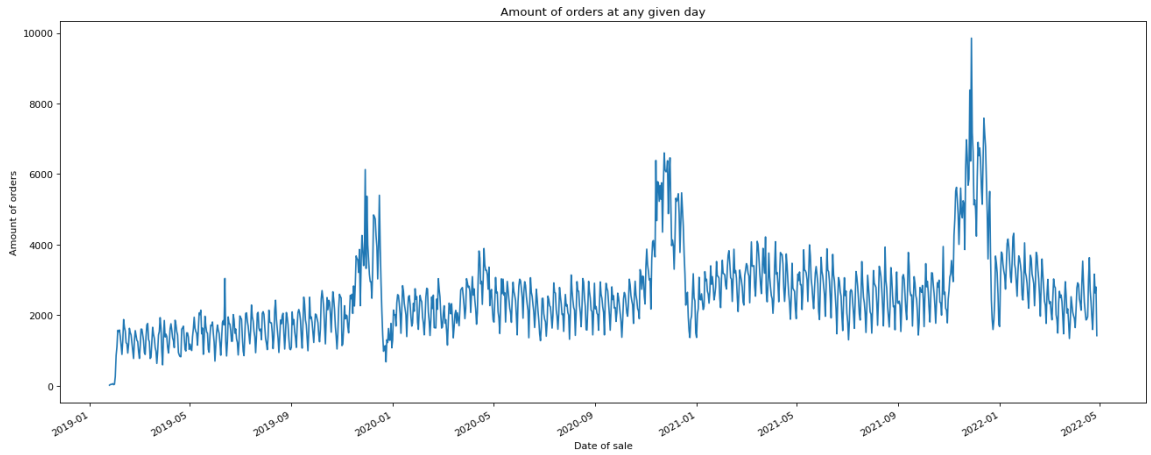
In [42]:

```
data.drop('created_at', inplace=True, axis=1)
```

We have more than 3 years of daily data of sales.

In [43]:

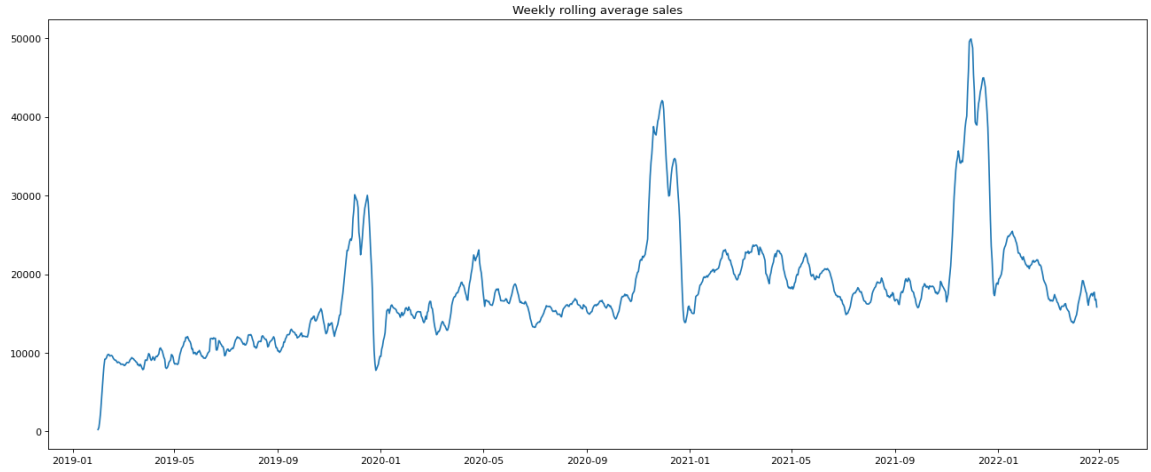
```
figure(figsize=(20, 8))
data.doc_date.value_counts().plot(xlabel='Date of sale', ylabel='Amount of order
s', title = 'Amount of orders at any given day')
plt.show()
```



We will also look at rolling average sales over 7 days. since, as we can see, days vary greatly. This can help better understanding on patterns and trend in orders.

In [44]:

```
orders_over_time = pd.DataFrame(index=data.doc_date.value_counts().index, data={
'orders' : data.doc_date.value_counts().values})
orders_over_time = orders_over_time.sort_index()
orders_over_time['orders'] = orders_over_time['orders'].rolling(7).sum()


figure(figsize=(20, 8))
#data.doc_date.value_counts().plot(xlabel='Date of sale', ylabel='Amount of orde
rs', title = 'Amount of orders at any given day')
plt.plot(orders_over_time)
plt.title('Weekly rolling average sales')
plt.show()
```

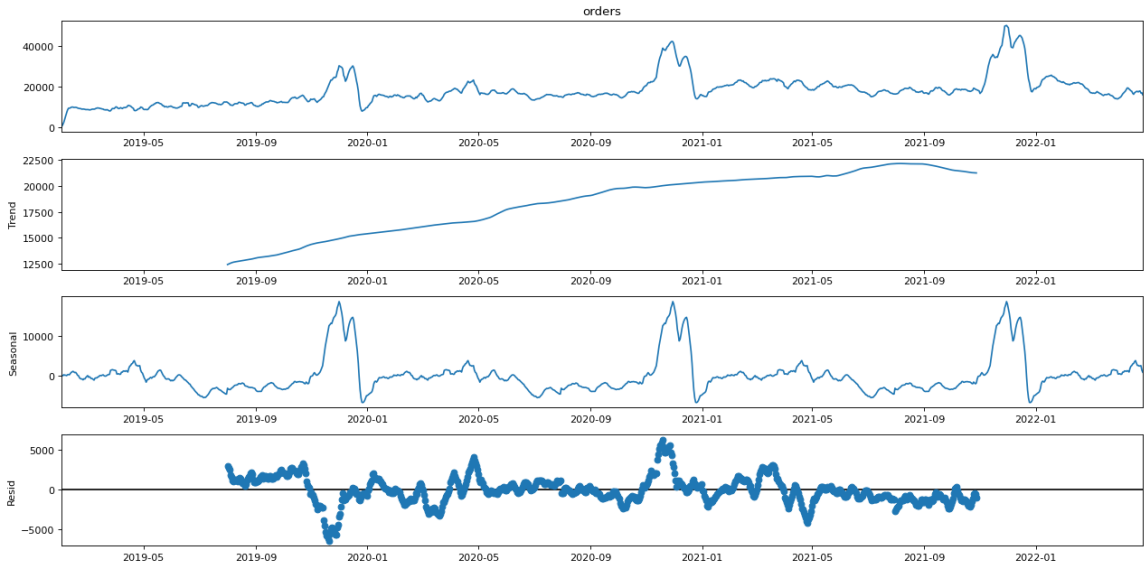


Decomposing data to Trend, Seasonal and Resid can help us understand what's happening in the data. \
What we can see is that Christmast are always the strongest season and we can also see rising trend with slight decrease in the past few months.

In [45]:

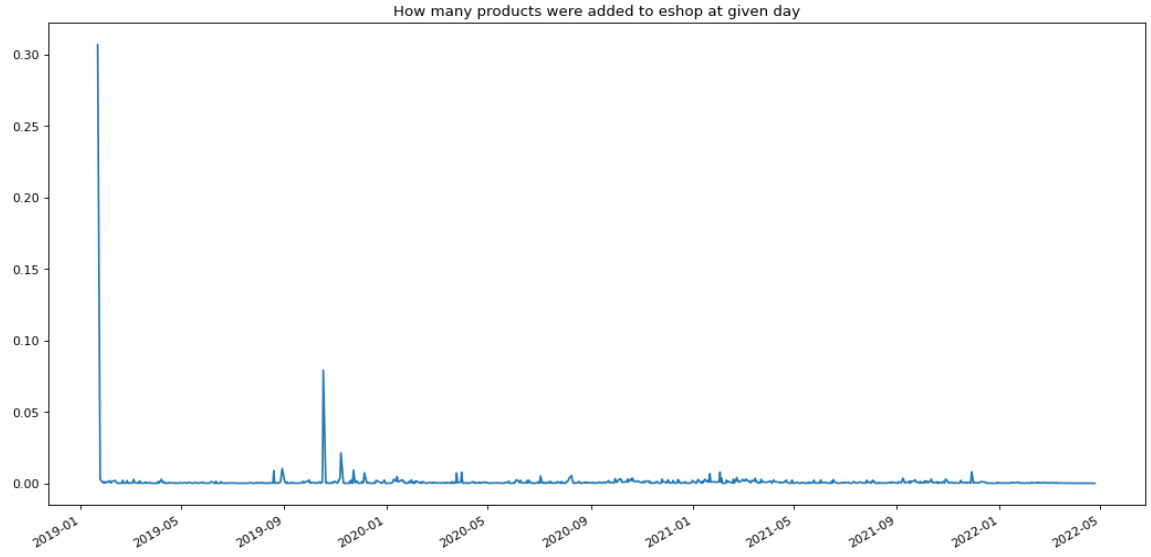
```
figure(figsize=(20, 8))
figure().patch.set_facecolor('white')
plt.figure(num=None, figsize=(50, 20), dpi=80, facecolor='w', edgecolor='k')
result = seasonal_decompose(orders_over_time.dropna().orders, model='additive',
period=365)  # Alternatively period = N days
result.plot()
pass
```

```
<Figure size 1600x640 with 0 Axes>

<Figure size 1280x640 with 0 Axes>

<Figure size 4000x1600 with 0 Axes>
```



In [46]:

```
# Take only date from timestamp in since_date - we don't neet to know exact hour
# Then look at days of data adding to the eshop.
product_since_to_date = []

for i in data.product_since:
    product_since_to_date.append(pd.Timestamp(i.date()))

data.product_since = product_since_to_date
data[['product_id', 'product_since']].drop_duplicates()['product_since'].value_c
ounts(normalize=True).plot(title='How many products were added to eshop at given
day')

plt.show()
```

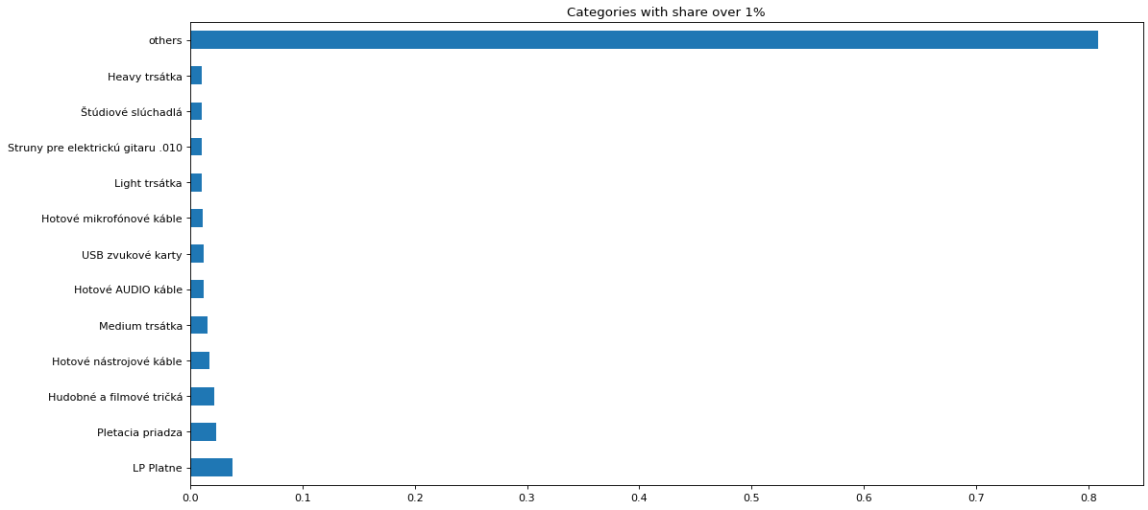


Categories

In [47]:

```
print(f'There is {len(data.category.unique())} categories used.')
create_others_value_counts(0.01, data, 'category').plot(kind='barh', title='Cate
gories with share over 1%')
plt.plot()
print(create_others_value_counts(0.01, data, 'category'))
```

```
There is 2463 categories used.
LP Platne                              0.038201
Pletacia priadza                       0.023490
Hudobné a filmové tričká               0.021730
Hotové nástrojové káble                0.017079
Medium trsátka                         0.015646
Hotové AUDIO káble                     0.012101
USB zvukové karty                      0.011836
Hotové mikrofónové káble               0.011012
Light trsátka                          0.010487
Struny pre elektrickú gitaru .010      0.010245
Štúdiové slúchadlá                     0.010132
Heavy trsátka                          0.010039
others                                 0.808001
dtype: float64
```

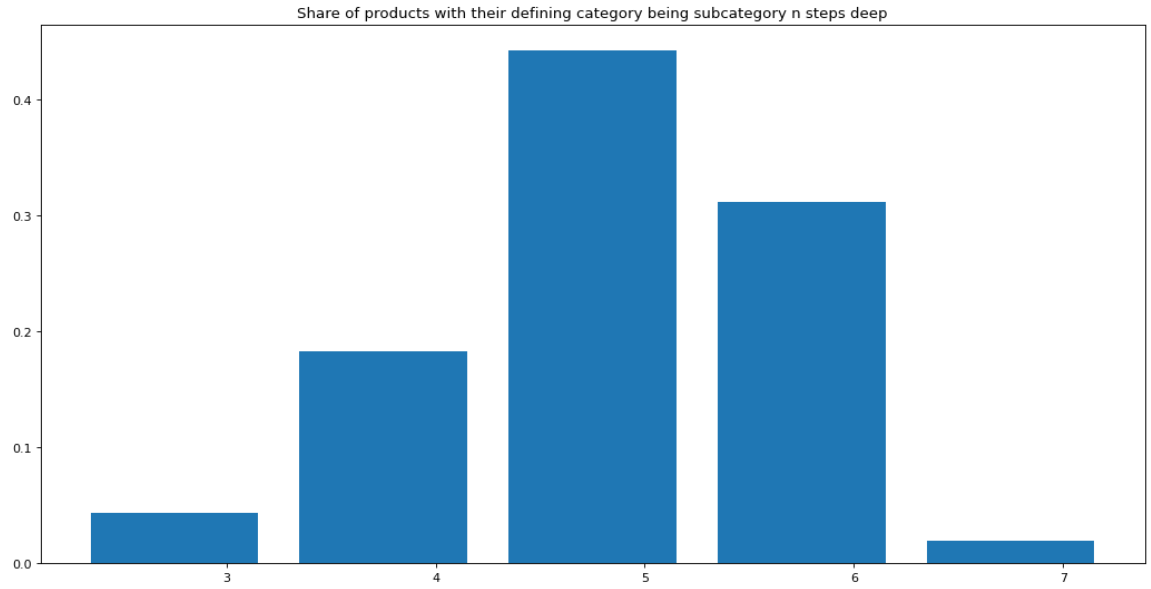


There is many categories used, but, as expected, products are distributed in between them, so even the biggest category (LP platne) doesn't reach even 4% share. \ I expect categories to be useful in product distribution and ML.

Categories are also stored in the category tree, so let's have a look how deep in the tree they are.

In [48]:

```
distribution = [len(i) for i in list(data.tree_path)]
occurrence = []
for value in list(set(distribution)):
    occurrence.append(distribution.count(value))

plt.bar([i/4 +1 for i in list(set(distribution))], [float(i)/sum(occurrence) for
i in occurrence])
plt.title('Share of products with their defining category being subcategory n st
eps deep')
plt.show()
```



In [49]:

```
print(f'There was {len(data.shop_basket_id.unique())} unique baskets ordered - t
otal amount of orders.')
```

```
There was 1211479 unique baskets ordered - total amount of orders.
```

In [50]:

```
print('Standard default warranty is 24 months, which is also true in almost all
 categories but one')
print(data[['catalog_category_id', 'default_warranty_period']].drop_duplicates()
.default_warranty_period.value_counts())

print('\n\nAlso used categories are almost all active. Inactive category doesn\'
t mean inactive products, but rather ending usage of the category.')
print(data[['catalog_category_id', 'category_status']].drop_duplicates().categor
y_status.value_counts())
```

```
Standard default warranty is 24 months, which is also true in almost
all categories but one
24.0    2464
1.0        1
Name: default_warranty_period, dtype: int64


Also used categories are almost all active. Inactive category does
n't mean inactive products, but rather ending usage of the category.
active      2460
inactive       5
Name: category_status, dtype: int64
```

Columns default_warranty_period and category_status are not very wide, they offer very little differences, so we will probably drop them later.

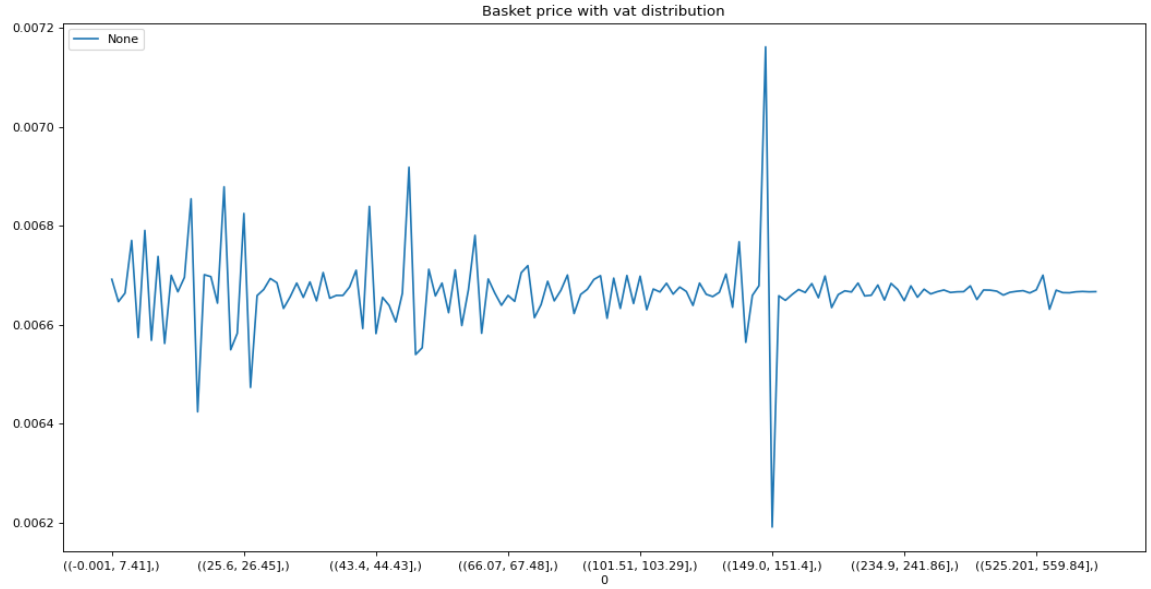
Money spent on orders

There are 4 columns totaling the price of shopping - order_price_without_vat, order_price_with_vat, basket_total_price_before_discount_with_vat, basket_total_price_with_vat. \ For the purpose of summing the purchase, one of them is enough - the best will be basket_total_price_with_vat. \ We are working with vat prices of products too and we are not going to include shipping costs (which are included in order price), but rather only total price of products in baskets. \ Customer has to pay vat, so it is note very relevant for them how 'cheap' the item is without vat, that's why we don't need prices without vat.

In [51]:

```
# We can drop all other summing columns
data.drop('order_price_without_vat', axis=1, inplace=True)
data.drop('order_price_with_vat', axis=1, inplace=True)
data.drop('basket_total_price_before_discount_with_vat', axis=1, inplace=True)

data.drop('item_unit_price_without_vat', axis=1, inplace=True)

pd.DataFrame(pd.qcut(np.array(data.basket_total_price_with_vat), q=150, duplicat
es='drop')).value_counts(normalize=True, sort=False).plot(title='Basket price wi
th vat distribution')
plt.legend(loc="upper left")
plt.plot()

print(f'Median basket price with vat {round(data.basket_total_price_with_vat.med
ian(), 2)} €')
print(f'Mean basket price with vat {round(data.basket_total_price_with_vat.mean
(), 2)} €')
```

```
Median basket price with vat 91.9 €
Mean basket price with vat 256.66 €
```



Brands

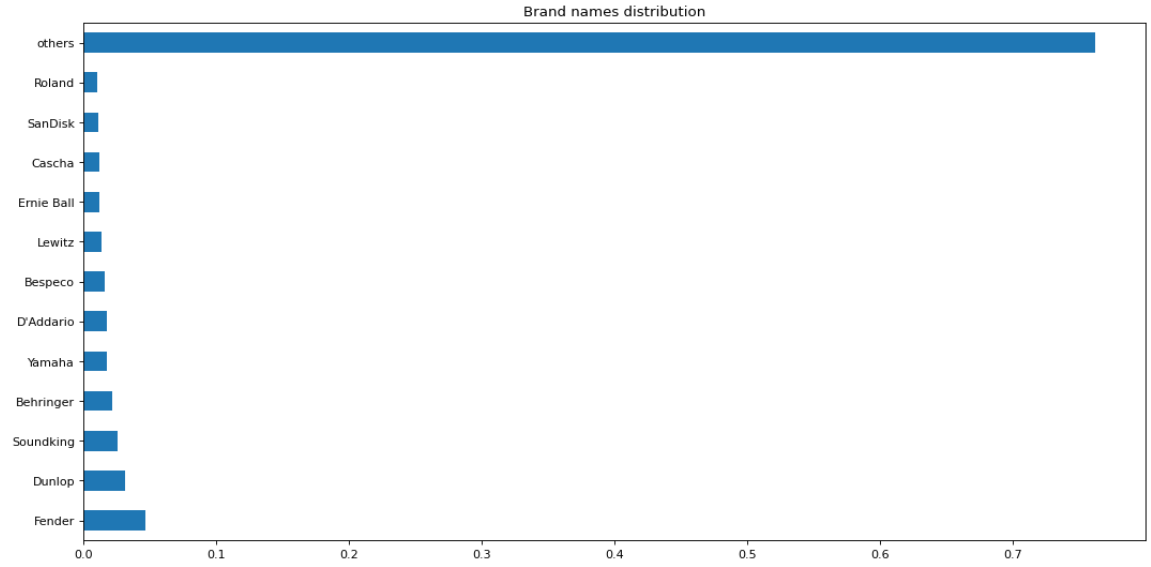
In [52]:

```
# calculate how many brands there are as well as the most common
print(f'There is {len(data.brand_name)} brands.')
print(data.brand_name.value_counts())
```

```
There is 3003435 brands.
Fender                      141077
Dunlop                       94411
Soundking                    78583
Behringer                    65795
Yamaha                       53069
                             ...
Anneke Van Giersbergen           1
Diplo                            1
Falk Neca                        1
Društvo Mrtvih Pesnikov          1
Ecstatic Vision                  1
Name: brand_name, Length: 5326, dtype: int64
```

In [53]:

```
create_others_value_counts(0.01, data, 'brand_name').plot(kind='barh', title='Br
and names distribution')
plt.show()
```

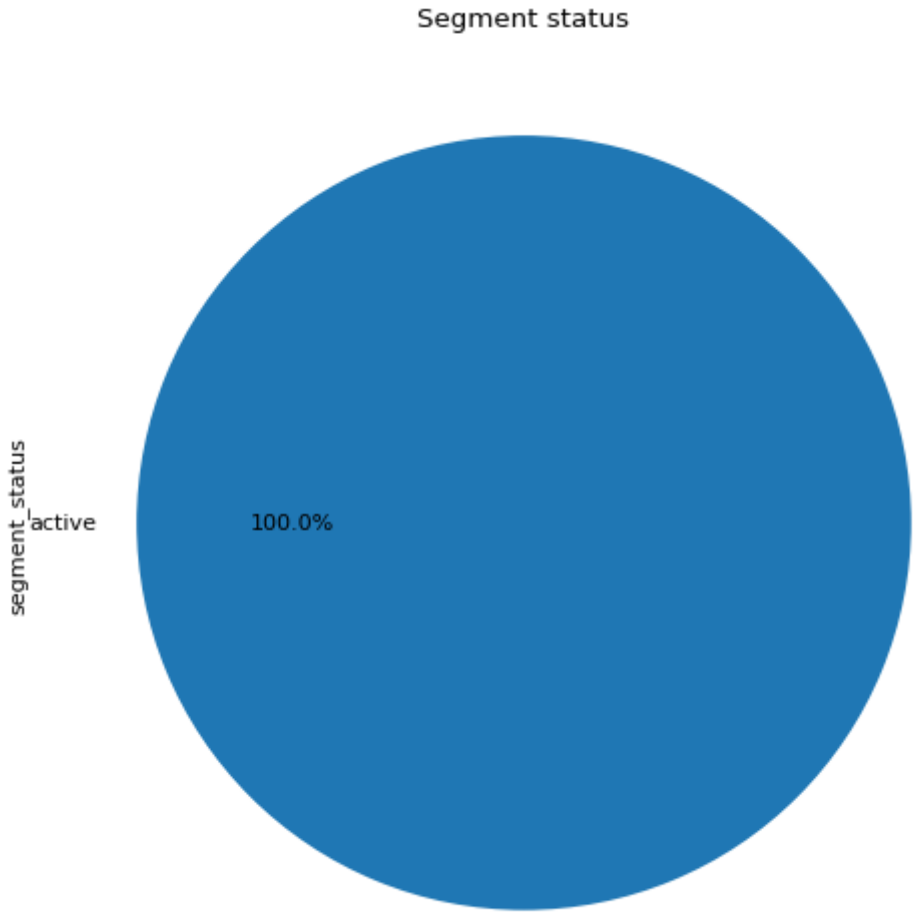


As we can see above, there is HUGE variety of brands in products with only few of them having more than 1% of products.

Segments

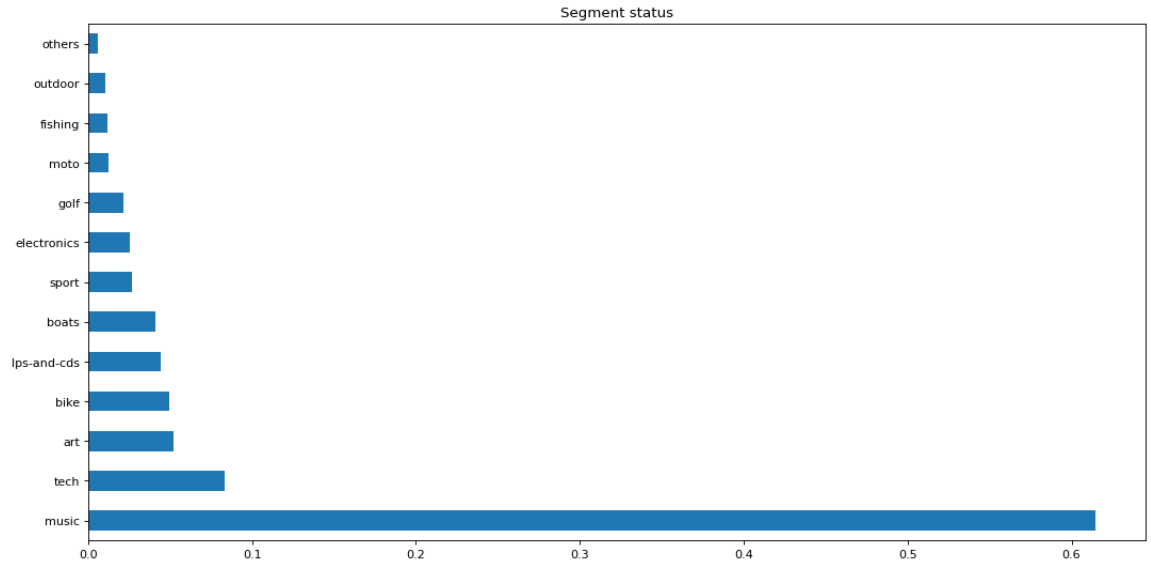
In [54]:

```
figure().patch.set_facecolor('white')
data.segment_status.value_counts().plot(kind='pie', autopct='%1.1f%%', title='Se
gment status')
plt.show()
```



In [55]:

```
figure().patch.set_facecolor('white')
create_others_value_counts(0.01, data, 'segment_parameterized').plot(kind='barh'
, title='Segment status', label='segment_parameterized')
plt.show()
```



In [56]:

```
print('Active and inactive segments from the data.')
data[['segment_parameterized', 'segment_status']].drop_duplicates().sort_values(
by='segment_status')
```

```
Active and inactive segments from the data.
```

Out[56]:

	segment_parameterized	segment_status
0	music	active
8	general	active
11	tech	active
21	sport	active
24	electronics	active
26	lps-and-cds	active
43	boats	active
64	bike	active
80	outdoor	active
81	kids	active
82	golf	active
89	fishing	active
118	moto	active
869	zoo	active
6351	art	active

As of now, segment status is active for all of the segments, so we can drop this column as well, it will not be useful as a feature.

In [57]:

```
data.drop('segment_status', inplace=True, axis=1)
```

In [58]:

```
data.to_csv('data/data_after_EDA.csv', index=False)
```