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

In [2]:
```
import zipfile

zip_file_path = 'C:/Users/stask/Analitics_Karpov/Module6/Project/ads.zip'
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    file_names = zip_ref.namelist()

dataframes = []
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    for file_name in file_names:
        with zip_ref.open(file_name) as file:
            df = pd.read_csv(file)
            dataframes.append(df)

ads_clients_data = dataframes[0]  # Assuming 'ads_clients_data.csv' is the first file in the zip
ads_data = dataframes[1]  # Assuming 'ads_data.csv' is the second file in the zip

ads_clients_data['create_date'] = pd.to_datetime(ads_clients_data['create_date'])
ads_clients_data['date'] = pd.to_datetime(ads_clients_data['date'])

ads_data['time'] = pd.to_datetime(ads_data['time'])
ads_data['date'] = pd.to_datetime(ads_data['date'])
```

Let's look at the distribution of impressions and clicks. Calculate the average number of impressions and the average number of clicks per ad for the whole period (round to integers).

In [3]:
```
ads_data.groupby(['ad_id','event','date'], as_index=False) \
    .agg({'platform':'count'}) \
    .rename(columns={'platform' :'total'}) \
    .groupby(['ad_id','event'], as_index=False) \
    .agg({'total':'mean'}) \
    .rename(columns={'total':'avg_per_day'}) \
    .sort_values(['ad_id', 'avg_per_day'], ascending=[True, False]).head()

# .tail(5)['ad_id'].astype(str).str.cat(sep=', ')
```

Out[3]:

	ad_id	event	avg_per_day
1	2	view	50.5
0	2	click	1.0
3	3	view	92.0
2	3	click	1.5
5	1902	view	20.5

In [4]:
```
# there are missing 0 values for click
number_per_ad_per_event = ads_data.groupby(['ad_id','event'], as_index=False) \
    .agg({'platform':'count'}) \
    .rename(columns={'platform':'event_number'})

pivot_per_ads = number_per_ad_per_event.pivot(index='ad_id',
                                              columns='event',
                                              values='event_number') \
                                       .fillna(0)
pivot_per_ads.head()
```

Out[4]:

event	click	view
ad_id		
2	1.0	101.0
3	3.0	184.0
1902	1.0	41.0
2064	0.0	35.0
2132	1.0	58.0

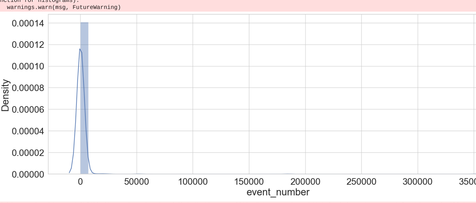
In [5]:
```
pivot_per_ads.mean()
```

Out[5]:
```
event
click    113.137824
view     923.131606
dtype: float64
```

Draw a graph of the distribution of impressions per ad for the whole period.

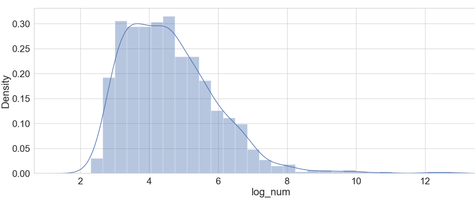
In [6]:
```
number_per_ad_per_event['log_num'] = np.log(number_per_ad_per_event.event_number)
filtered_df = number_per_ad_per_event.query('event == "view"')

sns.set(
font_scale=2,
    style='whitegrid',
    rc={'figure.figsize':(20,7)}
)

sns.distplot(filtered_df.event_number)
plt.show()
sns.distplot(filtered_df.log_num)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[6]:
```
<AxesSubplot:xlabel='log_num', ylabel='Density'>
```



Now let's calculate the moving average of impressions with window 2. What is the moving average for April 6, 2019 (round up to integers)?

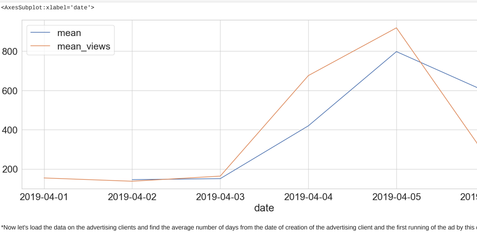
In [7]:
```
ads_view_per_date = ads_data.query('event == "view"') \
                            .pivot_table(index='ad_id',
                                         columns='date',
                                         values='platform',
                                         aggfunc='count')

mean_views = ads_view_per_date.mean()
rolling_views = mean_views.rolling(2).mean().round(0)
rolling_views
```

Out[7]:
```
date
2019-04-01      NaN
2019-04-02    146.0
2019-04-03    151.0
2019-04-04    420.0
2019-04-05    798.0
2019-04-06    598.0
dtype: float64
```

The moving average is often used to find anomalies in the data. Let's try to plot the values of the arithmetic average by day and the moving average of the number of impressions on the same graph. On which day is there the greatest difference modulo between the arithmetic average and the moving average? The days in which the moving average is equal to NaN are ignored.

In [8]:
```
sns.lineplot(data=rolling_views, label='mean')
sns.lineplot(data=mean_views, label='mean_views')
```

Out[8]:
```
<AxesSubplot:xlabel='date'>
```



*Now let's load the data on the advertising clients and find the average number of days from the date of creation of the advertising client and the first running of the ad by this client.

In [9]:
```
full_data = ads_data.merge(ads_clients_data.drop(columns=['date','community_id']))
full_data.head(1)
```

Out[9]:

	date	time	event	platform	ad_id	client_union_id	campaign_union_id	ad_cost_type	ad_cost	has_video	target_audience_count	create_date
0	2019-04-01	2019-04-01 00:00:48	view	android	45061	34734	45061	CPM	200.6	0	1955269	2018-12-04

In [10]:
```
full_data\
    .groupby('client_union_id') \
    .apply(lambda group: (group.date - group.create_date).min()).mean().days
```

Out[10]:
```
124
```

*Calculate the conversion from the creation of the advertising client to the launch of the first ad for a maximum of 365 days. Give the answer as a percentage and round to the hundredths. (Filter by the value in pd.Timedelta(365, unit='d'))

In [11]:
```
creation_diff = full_data \
    .groupby('client_union_id', as_index=False) \
    .apply(lambda group: (group.date - group.create_date).min().days)\
    .rename(columns={None:'days_diff'})
# threshold = pd.to_timedelta(365, unit='d')
# creation_diff.query('days_diff < @threshold')
creation_diff['year'] = creation_diff.query('days_diff < 365')
```

In [12]:
```
round(creation_diff_year.shape[0] / ads_clients_data.client_union_id.nunique() * 100, 2)
```

Out[12]:
```
0.69
```

*Let's break down our clients by the interval from creation to launch of the ad, equal to 30. Determine how many unique clients ran their first ad in their first month (0 to 30 days). The list of gaps for the pd.cut method is [0, 30, 90, 180, 365].

In [13]:
```
interval = pd.cut(creation_diff.days_diff,
         bins=[0, 30, 90, 180, 365],
         labels=['month', 'season', 'half a year', 'year'])
creation_diff['date_range'] = interval
creation_diff.head(3)
```

Out[13]:

	client_union_id	days_diff	date_range
0	1	98	half a year
1	9	114	half a year
2	13	66	season

In [14]:
```
creation_diff.query('date_range == "month"').client_union_id.nunique()
```

Out[14]:
```
11
```

*Now let's display these categories on an interactive graph with the number of unique customers in them.

In [15]:
```
sns.countplot(creation_diff['date_range'])

# Create the countplot using Plotly
fig = px.histogram(creation_diff, x='date_range', category_orders={'date_range': ['month', 'season', 'half a year', 'year']})

# Display the countplot
fig.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```