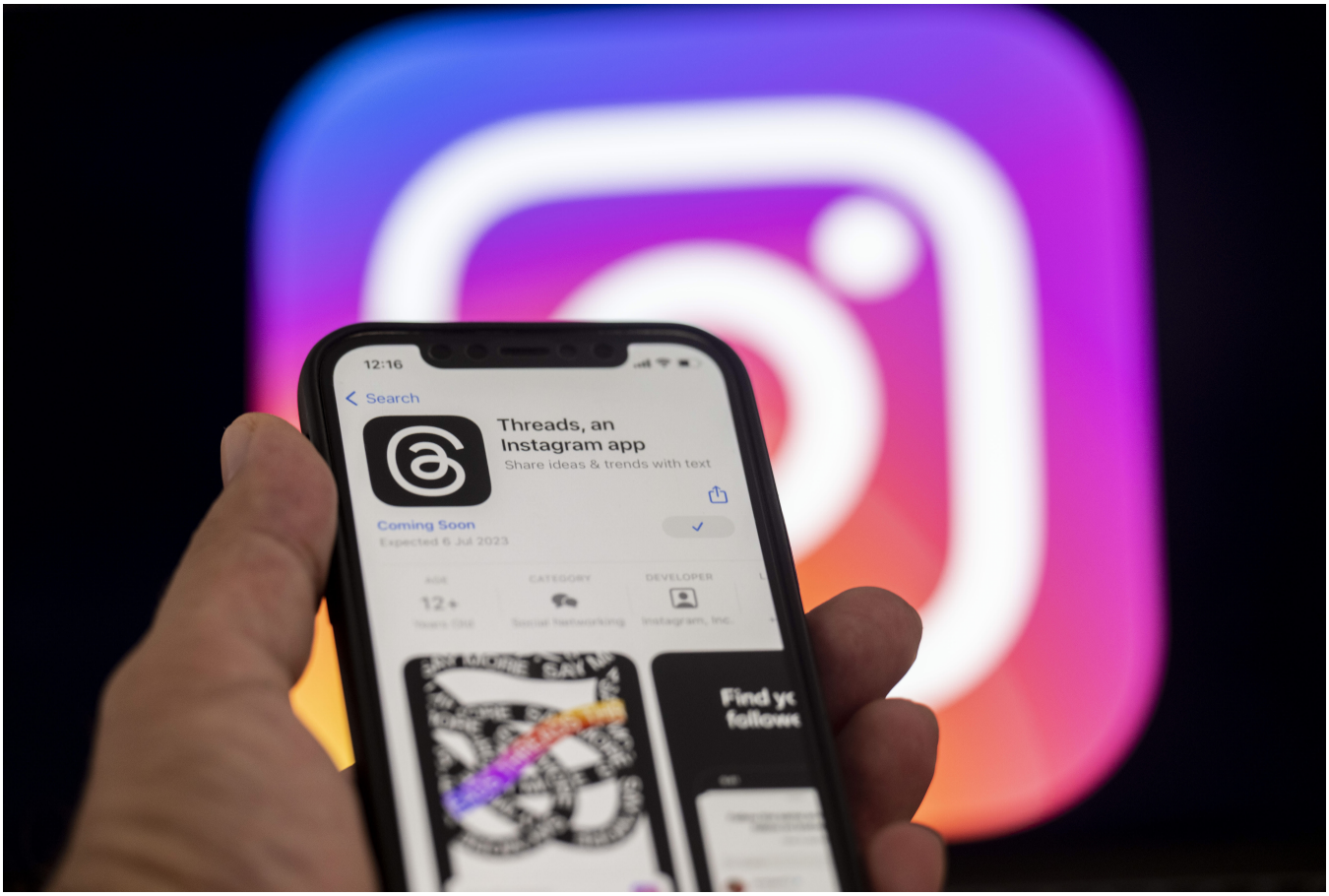
Analyzing User Sentiments and Performance of Threads: An Instagram App Reviews Study



In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
df = pd.read_csv("threads_reviews.csv")
```

In [4]:

```
df.head()
```

Out[4]:

	source	review_description	rating	review_date
0	Google Play	Meh. Not the greatest experience on a Chromebo...	2	2023-07-08 14:18:24
1	Google Play	Pretty good for a first launch!! Its easy to u...	3	2023-07-19 20:52:48
2	Google Play	For a brand new app, it's very well optimized....	3	2023-07-06 23:03:11
3	Google Play	Great app with a lot of potential! However, th...	3	2023-07-10 00:53:25
4	Google Play	The app is good, but it needs a lot of functio...	3	2023-07-06 16:57:43

In [5]:

```
df.tail()
```

Out[5]:

	source	review_description	rating	review_date
32905	App Store	This killed my dog. Mark zuckerburg strangled ...	1	2023-07-06 01:23:55
32906	App Store	Add Search and hashtag like Twitter !	1	2023-07-19 08:01:06
32907	App Store	bad twister	1	2023-07-17 06:39:13
32908	App Store	Yet another trash from Meta.	1	2023-07-07 17:47:16
32909	App Store	Nothing special this app is just a copy of twi...	1	2023-07-07 07:01:43

In [6]:

```
df.shape
```

Out[6]:

```
(32910, 4)
```

In [7]:

```
df.columns
```

Out[7]:

```
Index(['source', 'review_description', 'rating', 'review_date'], dtype='ob
ject')
```

In [8]:

```
df.duplicated().sum()
```

Out[8]:

```
1
```

In [9]:

```
df = df.drop_duplicates()
```

In [10]:

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

Out[10]:

```
source                0
review_description    0
rating                0
review_date           0
dtype: int64
```

In [11]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32909 entries, 0 to 32909
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   source              32909 non-null  object
 1   review_description  32909 non-null  object
 2   rating              32909 non-null  int64 
 3   review_date         32909 non-null  object
dtypes: int64(1), object(3)
memory usage: 1.3+ MB
```

In [12]:

```
df.describe()
```

Out[12]:

	rating
count	32909.000000
mean	3.398432
std	1.751484
min	1.000000
25%	1.000000
50%	4.000000
75%	5.000000
max	5.000000

In [13]:

```
df.nunique()
```

Out[13]:

```
source                   2
review_description    26706
rating                    5
review_date           31667
dtype: int64
```

In [14]:

```
df['source'].unique()
```

Out[14]:

```
array(['Google Play', 'App Store'], dtype=object)
```

In [15]:

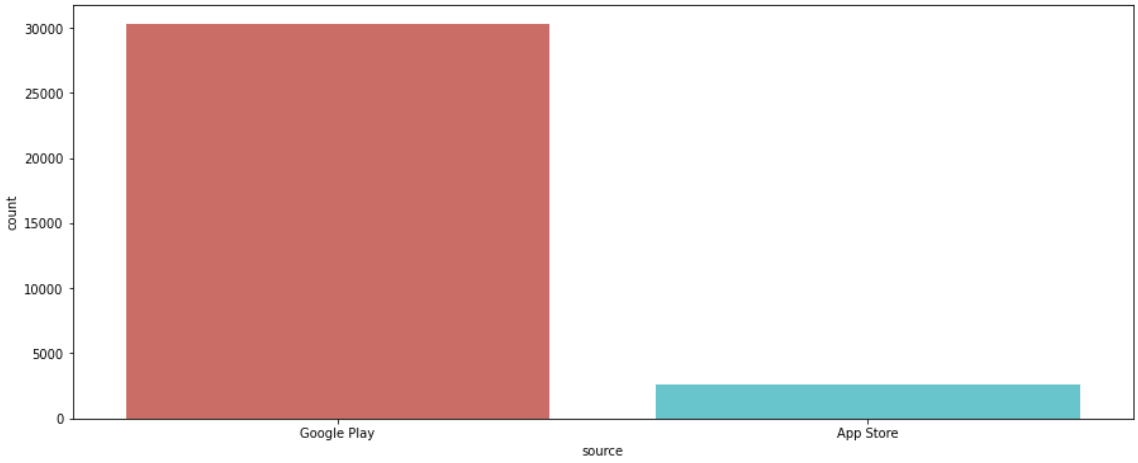
```
df['source'].value_counts()
```

Out[15]:

```
Google Play    30270
App Store       2639
Name: source, dtype: int64
```

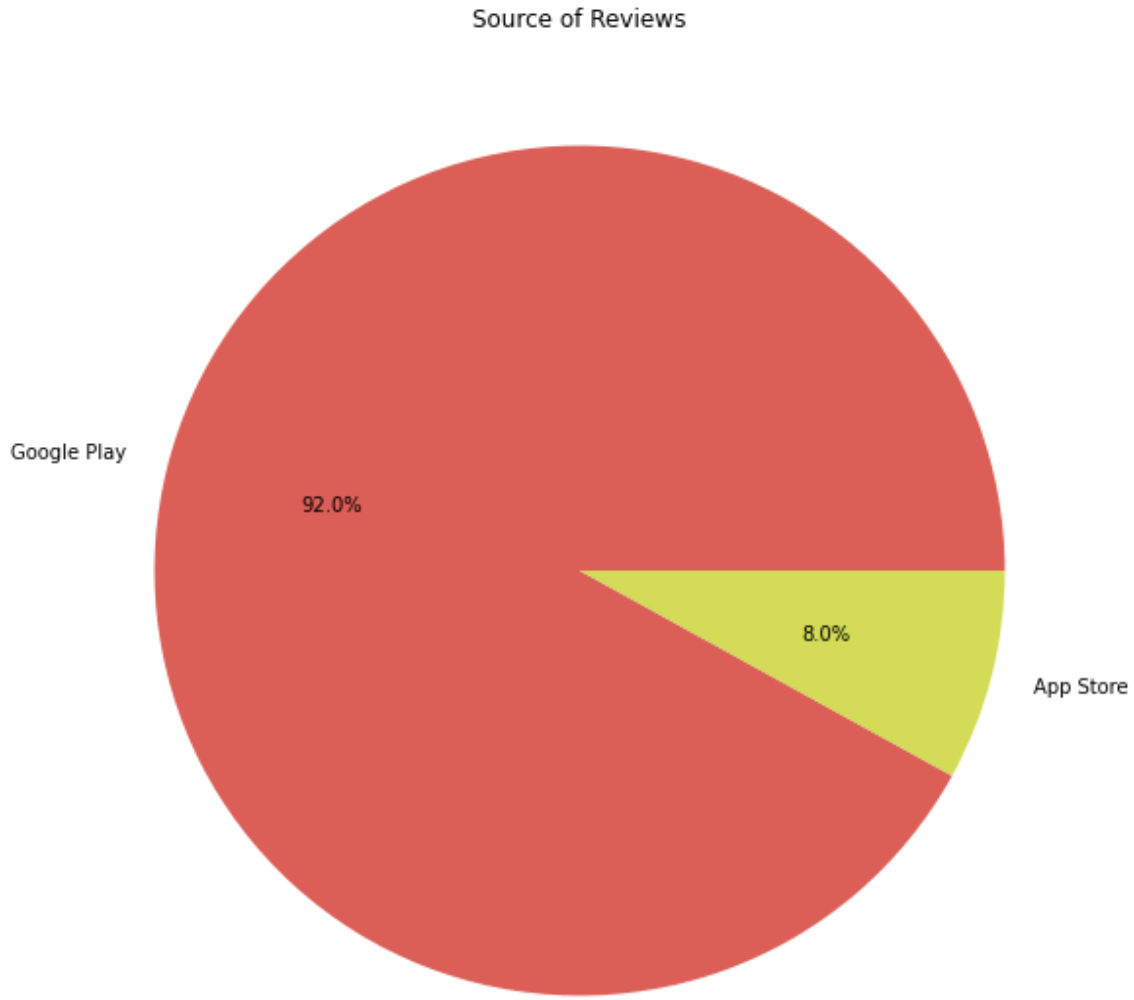
In [16]:

```
plt.figure(figsize=(15,6))
sns.countplot(df['source'], data = df, palette = 'hls')
plt.show()
```



In [17]:

```
plt.figure(figsize=(10, 10))
counts = df['source'].value_counts()
plt.pie(counts, labels=counts.index, autopct='%1.1f%%', colors=sns.color_palette('hls'))
plt.title('Source of Reviews')
plt.show()
```

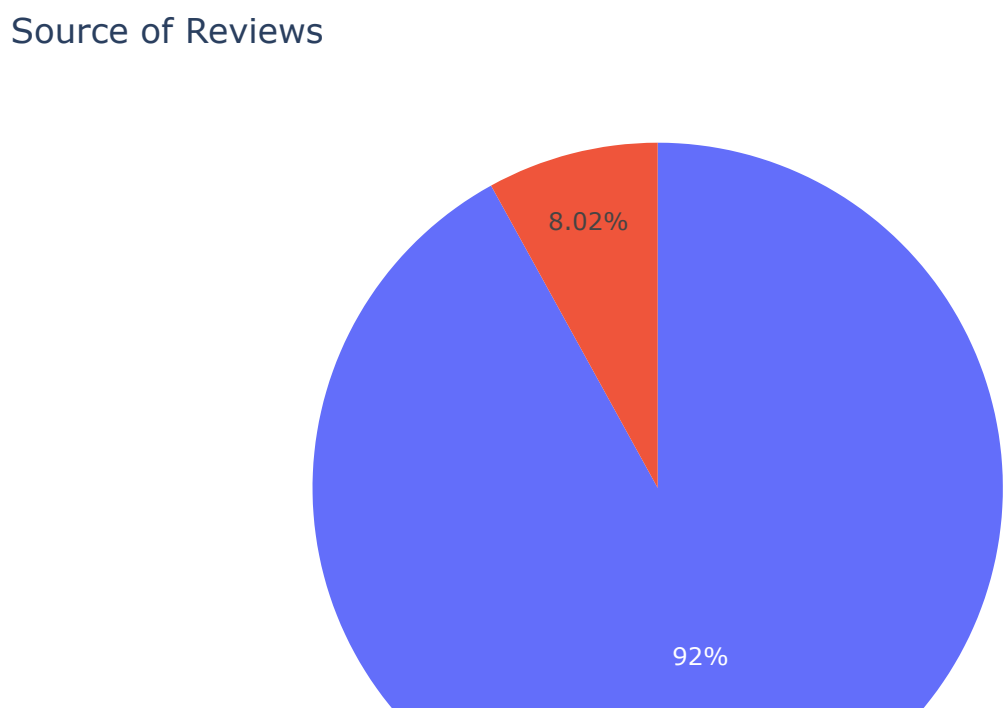


In [18]:

```
fig = go.Figure(data=[go.Bar(x=df['source'].value_counts().index, y=df['source'].value_c
fig.update_layout(title='Source of Reviews',xaxis_title="Source",yaxis_title="Count")
fig.show()
```

In [19]:

```
counts = df['source'].value_counts()
fig = go.Figure(data=[go.Pie(labels=counts.index, values=counts)])
fig.update_layout(title='Source of Reviews')
fig.show()
```



In [20]:

```
df['rating'].unique()
```

Out[20]:

```
array([2, 3, 1, 5, 4], dtype=int64)
```

In [21]:

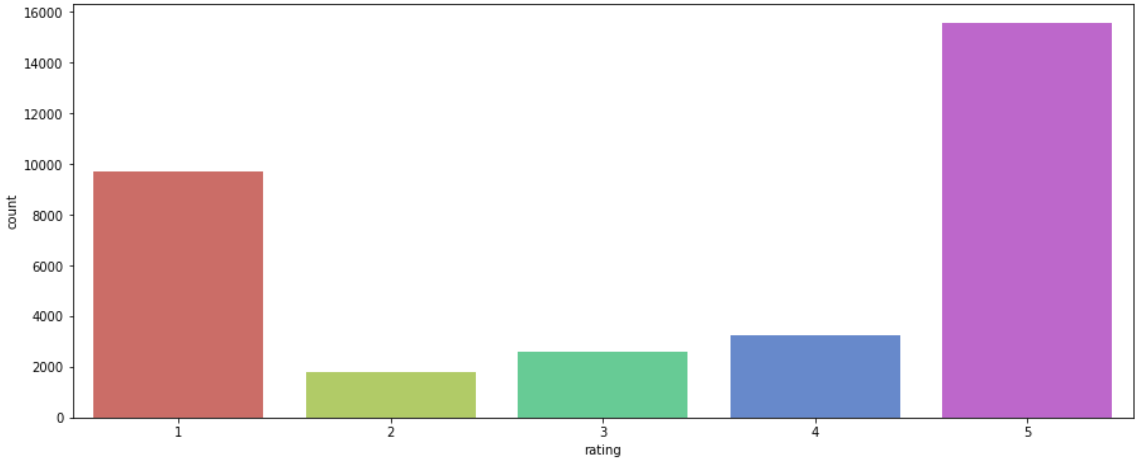
```
df['rating'].value_counts()
```

Out[21]:

```
5    15558
1     9726
4     3244
3     2585
2     1796
Name: rating, dtype: int64
```

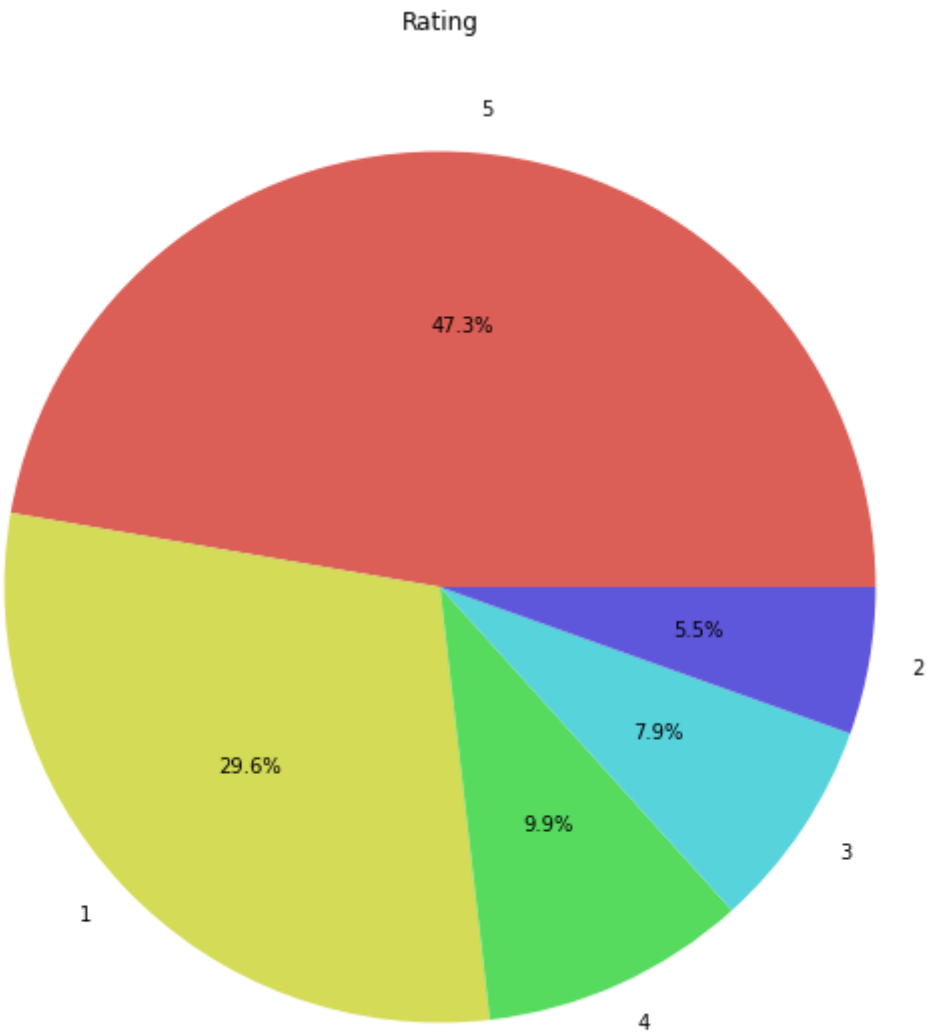
In [22]:

```
plt.figure(figsize=(15,6))
sns.countplot(df['rating'], data = df, palette = 'hls')
plt.show()
```



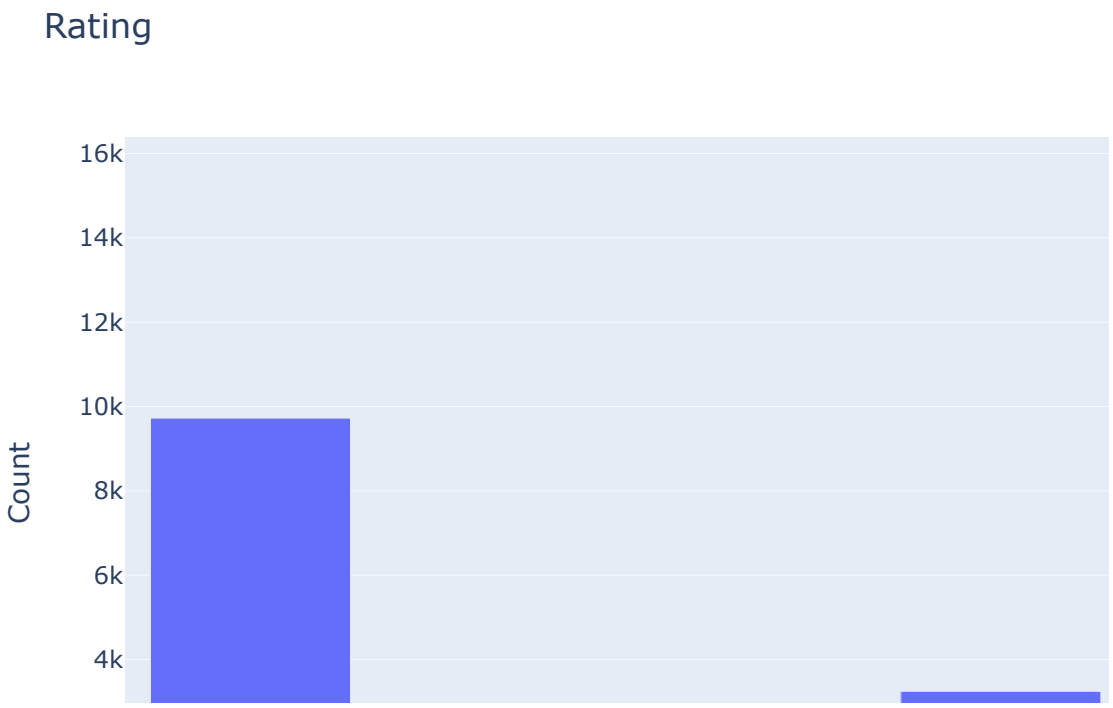
In [23]:

```
plt.figure(figsize=(10, 10))
counts = df['rating'].value_counts()
plt.pie(counts, labels=counts.index, autopct='%1.1f%%', colors=sns.color_palette('hls'))
plt.title('Rating')
plt.show()
```



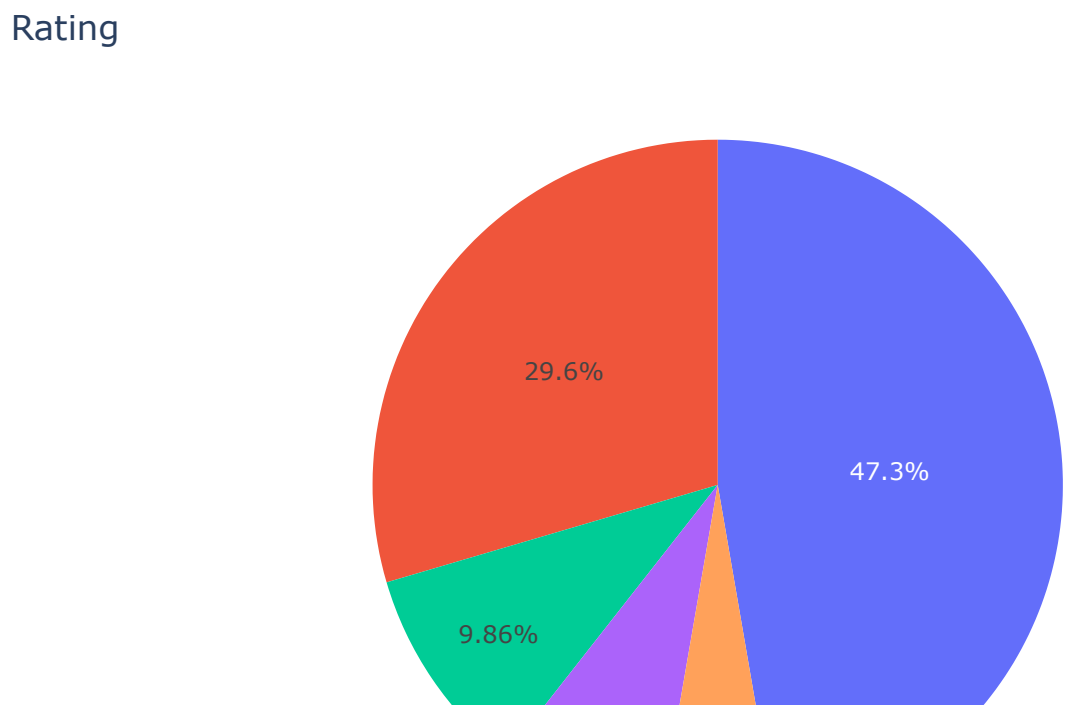
In [24]:

```
fig = go.Figure(data=[go.Bar(x=df['rating'].value_counts().index, y=df['rating'].value_c
fig.update_layout(title='Rating',xaxis_title="Rating",yaxis_title="Count")
fig.show()
```



In [25]:

```
counts = df['rating'].value_counts()
fig = go.Figure(data=[go.Pie(labels=counts.index, values=counts)])
fig.update_layout(title='Rating')
fig.show()
```



In [26]:

```
df['review_date'] = pd.to_datetime(df['review_date'])
```

In [27]:

```
df1 = df.copy()
```

In [28]:

```
df1.set_index('review_date', inplace=True)
```

In [29]:

```
start_date = df1.index.min()
end_date = df1.index.max()
```

In [30]:

```
print("Start Date of Reviews:", start_date)
print("End Date of Reviews:", end_date)
```

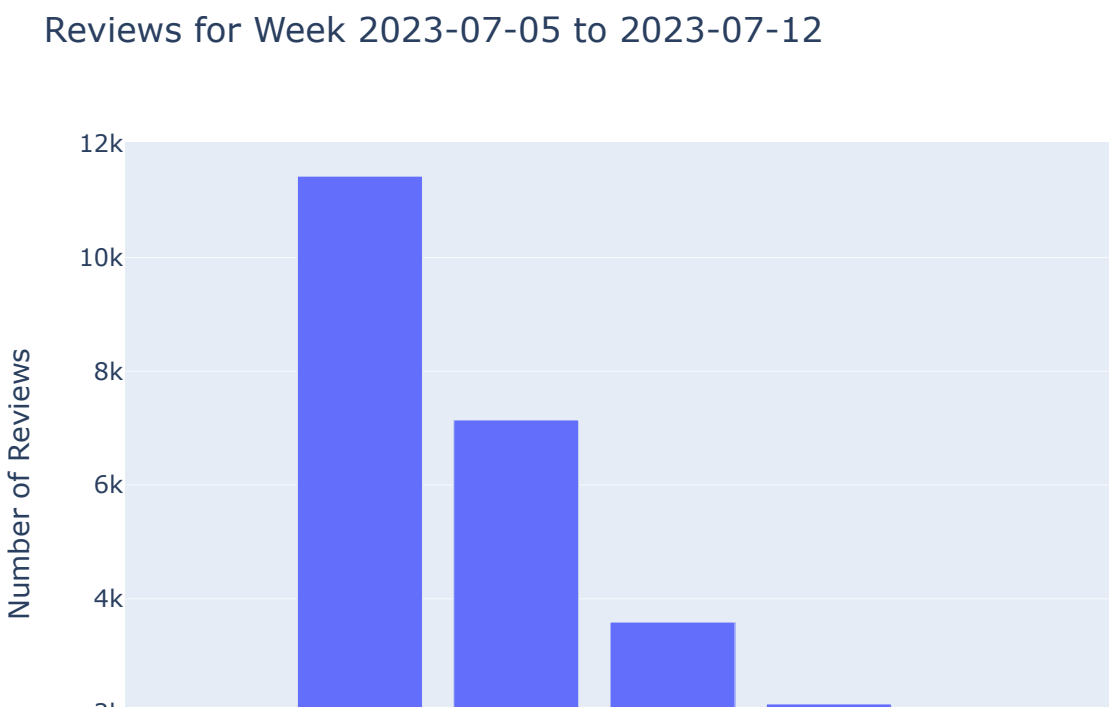
```
Start Date of Reviews: 2023-07-05 22:53:12
End Date of Reviews: 2023-07-25 09:42:20
```

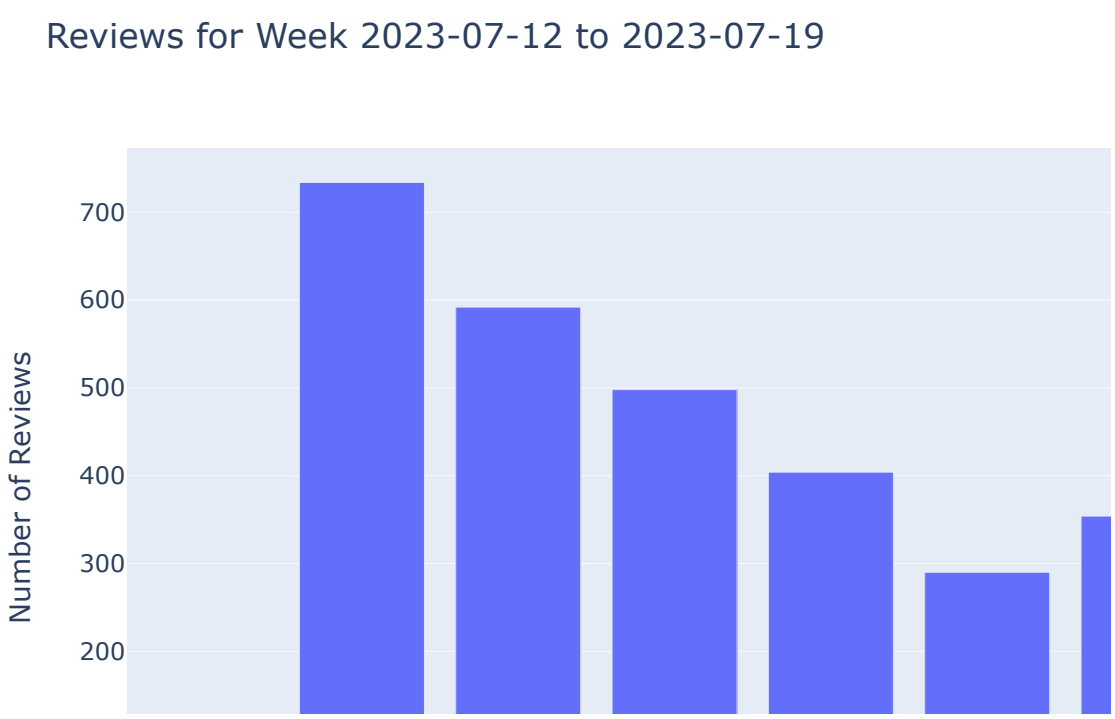
In [31]:

```
week_delta = pd.Timedelta(days=7)
current_date = start_date
```

In [32]:

```
while current_date <= end_date:
    start_week = current_date
    end_week = current_date + week_delta
    current_week_data = df1[(df1.index >= start_week) & (df1.index < end_week)]
    weekly_counts = current_week_data.resample('D').size()
    fig = px.bar(weekly_counts, x=weekly_counts.index, y=weekly_counts.values,
                 labels={'x': 'Date', 'y': 'Number of Reviews'},
                 title=f'Reviews for Week {start_week.strftime("%Y-%m-%d")} to {end_week
    fig.update_layout(xaxis_tickangle=-45)
    fig.show()
    current_date += week_delta
```



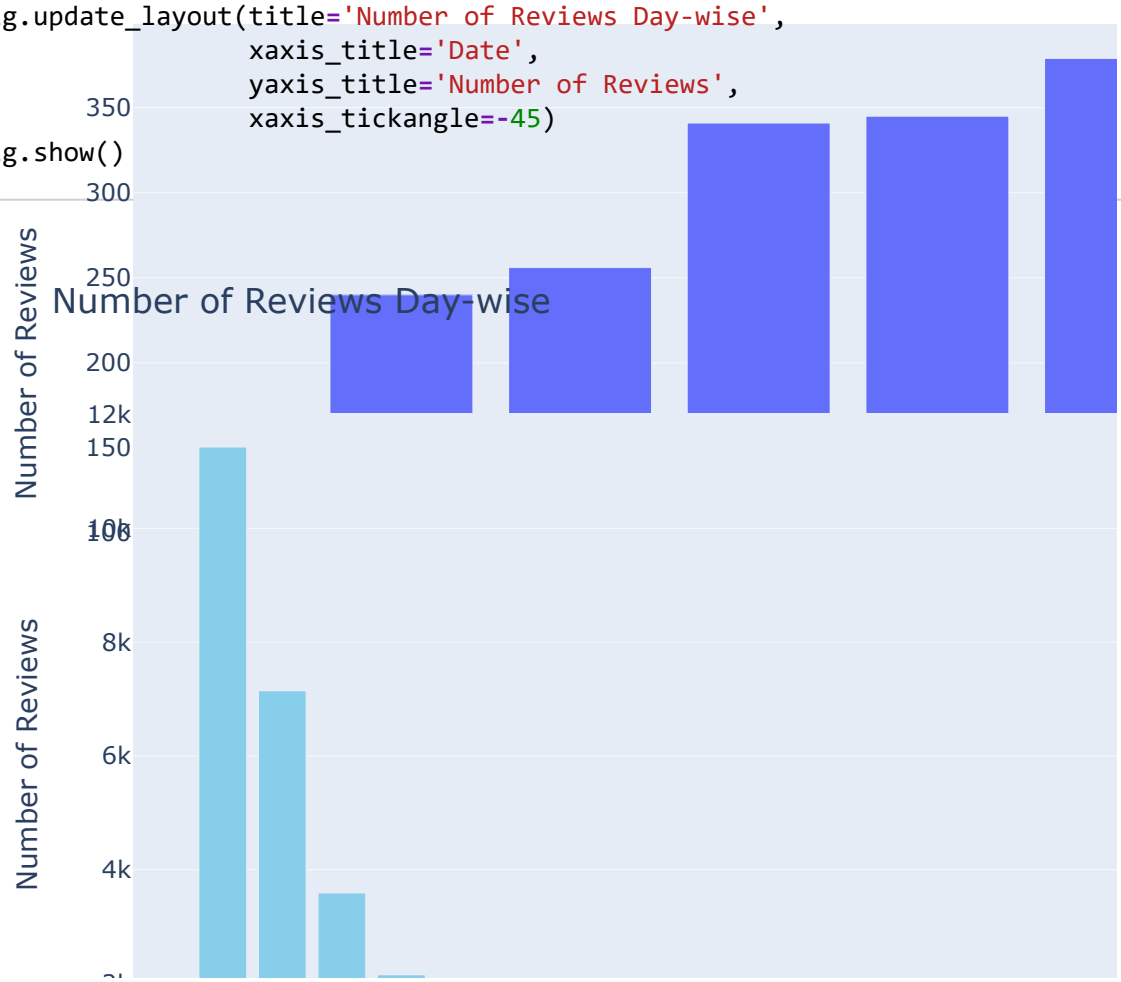


In [33]:

```
daily_counts = df1.resample('D').size()
```

In [34]:

```
fig = go.Figure()
fig.add_trace(go.Bar(x=daily_counts.index, y=daily_counts.values,
                     marker_color='skyblue'))
fig.update_layout(title='Number of Reviews Day-wise',
                  xaxis_title='Date',
                  yaxis_title='Number of Reviews',
                  xaxis_tickangle=-45)
fig.show()
```



In [35]:

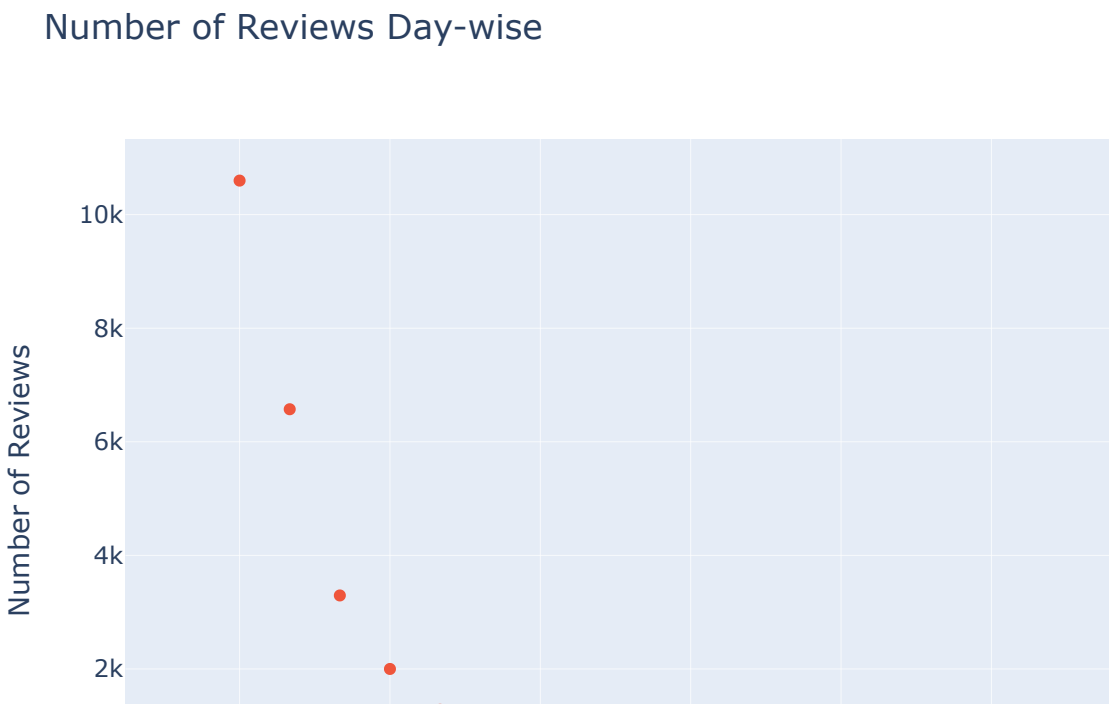
```
fig = go.Figure()
for source_name, source_data in df1.groupby('source'):
    fig.add_trace(go.Scatter(x=source_data.resample('D').size().index, y=source_data.res
                             mode='lines', name=source_name))
fig.update_layout(title='Number of Reviews Day-wise',
                  xaxis_title='Date',
                  yaxis_title='Number of Reviews',
                  xaxis_tickangle=-45)
fig.show()
```

In [36]:

```
fig = go.Figure()
for source_name, source_data in df1.groupby('source'):
    fig.add_trace(go.Scatter(x=source_data.resample('D').size().index, y=source_data.res
                             mode='lines', name=source_name))
fig.update_layout(title='Number of Reviews Day-wise',
                  xaxis_title='Date',
                  yaxis_title='Number of Reviews',
                  xaxis_tickangle=-45)
fig.show()
```

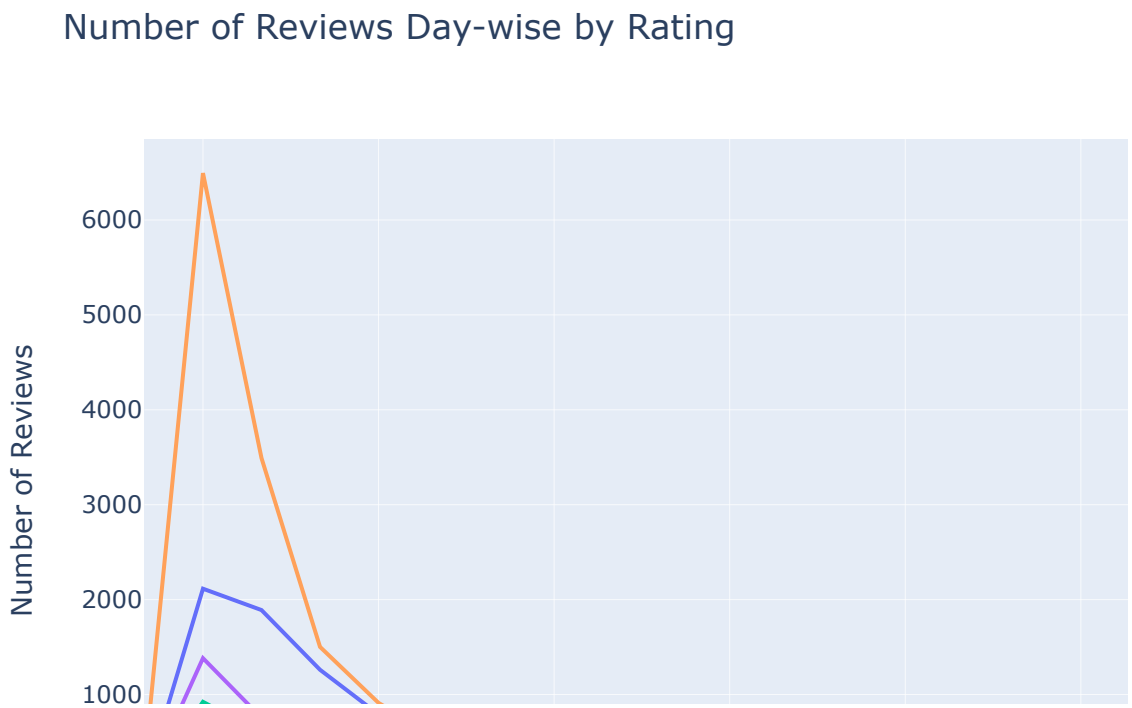
In [37]:

```
fig = go.Figure()
for source_name, source_data in df1.groupby('source'):
    fig.add_trace(go.Scatter(x=source_data.resample('D').size().index, y=source_data.res
                             mode='markers', name=source_name))
fig.update_layout(title='Number of Reviews Day-wise',
                  xaxis_title='Date',
                  yaxis_title='Number of Reviews',
                  xaxis_tickangle=-45)
fig.show()
```



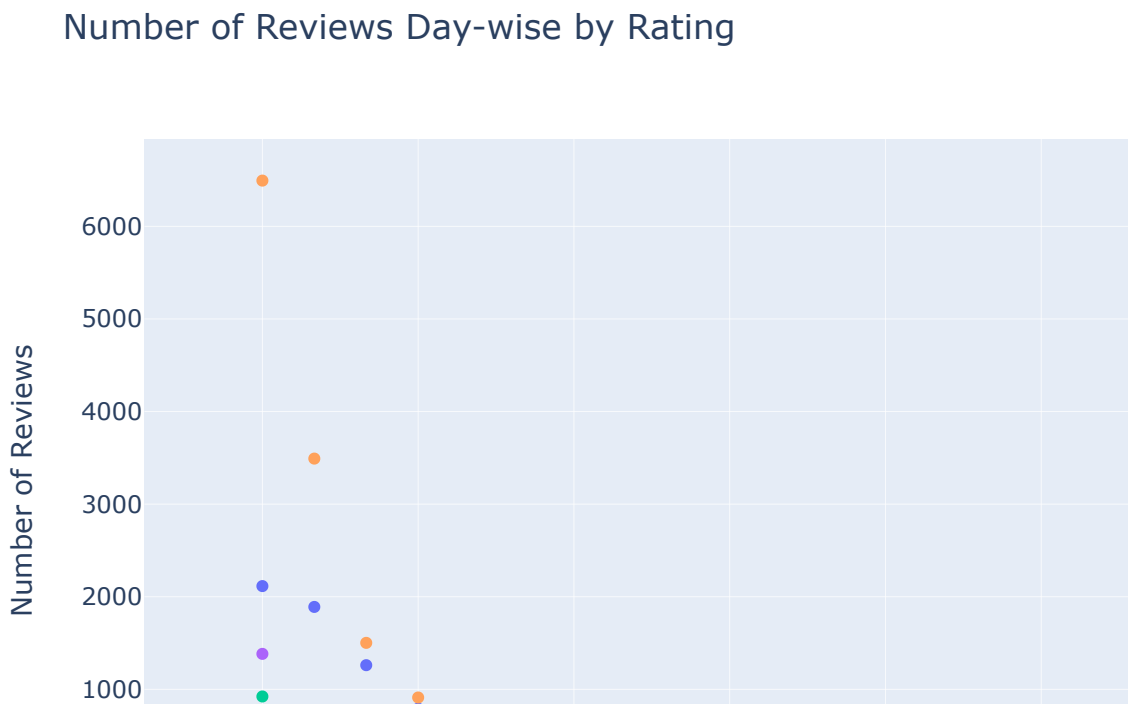
In [38]:

```
fig = go.Figure()
for rating_val, rating_data in df1.groupby('rating'):
    fig.add_trace(go.Scatter(x=rating_data.resample('D').size().index, y=rating_data.res
                             mode='lines', name=f'Rating {rating_val}'))
fig.update_layout(title='Number of Reviews Day-wise by Rating',
                  xaxis_title='Date',
                  yaxis_title='Number of Reviews',
                  xaxis_tickangle=-45)
fig.show()
```



In [39]:

```
fig = go.Figure()
for rating_val, rating_data in df1.groupby('rating'):
    fig.add_trace(go.Scatter(x=rating_data.resample('D').size().index, y=rating_data.res
                             mode='markers', name=f'Rating {rating_val}'))
fig.update_layout(title='Number of Reviews Day-wise by Rating',
                  xaxis_title='Date',
                  yaxis_title='Number of Reviews',
                  xaxis_tickangle=-45)
fig.show()
```



In [40]:

```
df['rating']=df['rating'].map({1:-1,2:-1,3:0,4:1,5:1})
```

In [41]:

```
df
```

Out[41]:

	source	review_description	rating	review_date
0	Google Play	Meh. Not the greatest experience on a Chromebo...	-1	2023-07-08 14:18:24
1	Google Play	Pretty good for a first launch!! Its easy to u...	0	2023-07-19 20:52:48
2	Google Play	For a brand new app, it's very well optimized....	0	2023-07-06 23:03:11
3	Google Play	Great app with a lot of potential! However, th...	0	2023-07-10 00:53:25
4	Google Play	The app is good, but it needs a lot of functio...	0	2023-07-06 16:57:43
...	...	...	...	...
32905	App Store	This killed my dog. Mark zuckerburg strangled ...	-1	2023-07-06 01:23:55
32906	App Store	Add Search and hashtag like Twitter !	-1	2023-07-19 08:01:06
32907	App Store	bad twister	-1	2023-07-17 06:39:13
32908	App Store	Yet another trash from Meta.	-1	2023-07-07 17:47:16
32909	App Store	Nothing special this app is just a copy of twi...	-1	2023-07-07 07:01:43

32909 rows × 4 columns

In [42]:

```
df_new = df[['review_description', 'rating']]
```

In [43]:

```
df_new
```

Out[43]:

	review_description	rating
0	Meh. Not the greatest experience on a Chromebo...	-1
1	Pretty good for a first launch!! Its easy to u...	0
2	For a brand new app, it's very well optimized....	0
3	Great app with a lot of potential! However, th...	0
4	The app is good, but it needs a lot of functio...	0
...	...	...
32905	This killed my dog. Mark zuckerburg strangled ...	-1
32906	Add Search and hashtag like Twitter !	-1
32907	bad twister	-1
32908	Yet another trash from Meta.	-1
32909	Nothing special this app is just a copy of twi...	-1

32909 rows × 2 columns

In [44]:

```
df_new['rating'].unique()
```

Out[44]:

```
array([-1,  0,  1], dtype=int64)
```

In [45]:

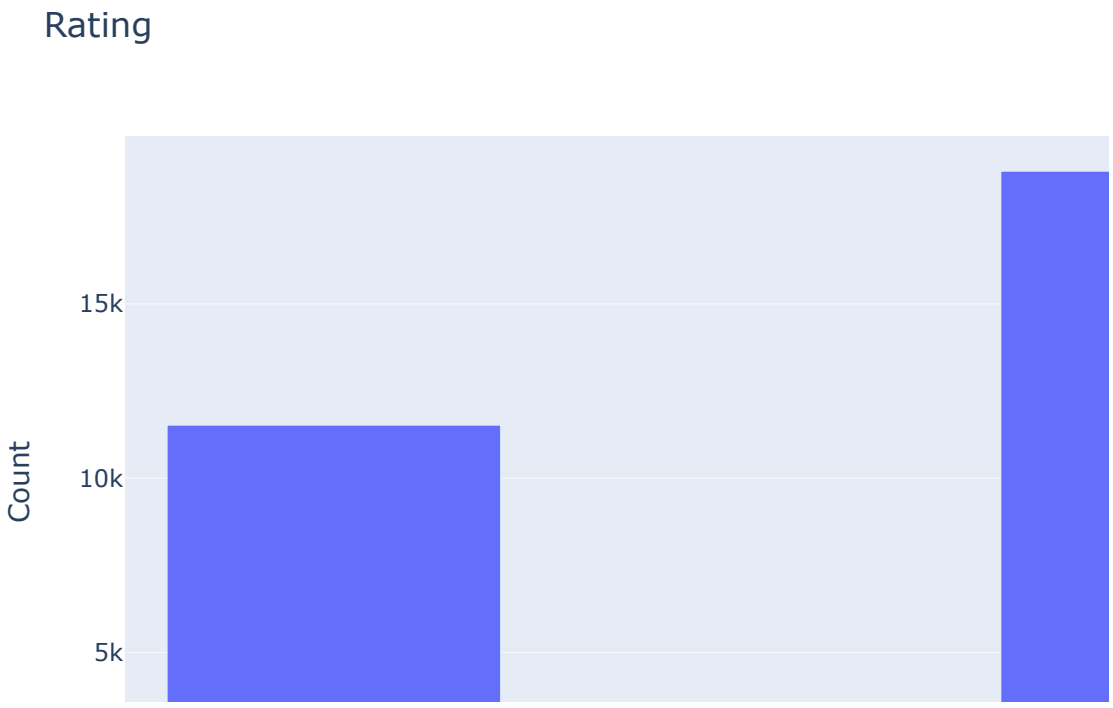
```
df_new['rating'].value_counts()
```

Out[45]:

```
 1    18802
-1    11522
 0     2585
Name: rating, dtype: int64
```

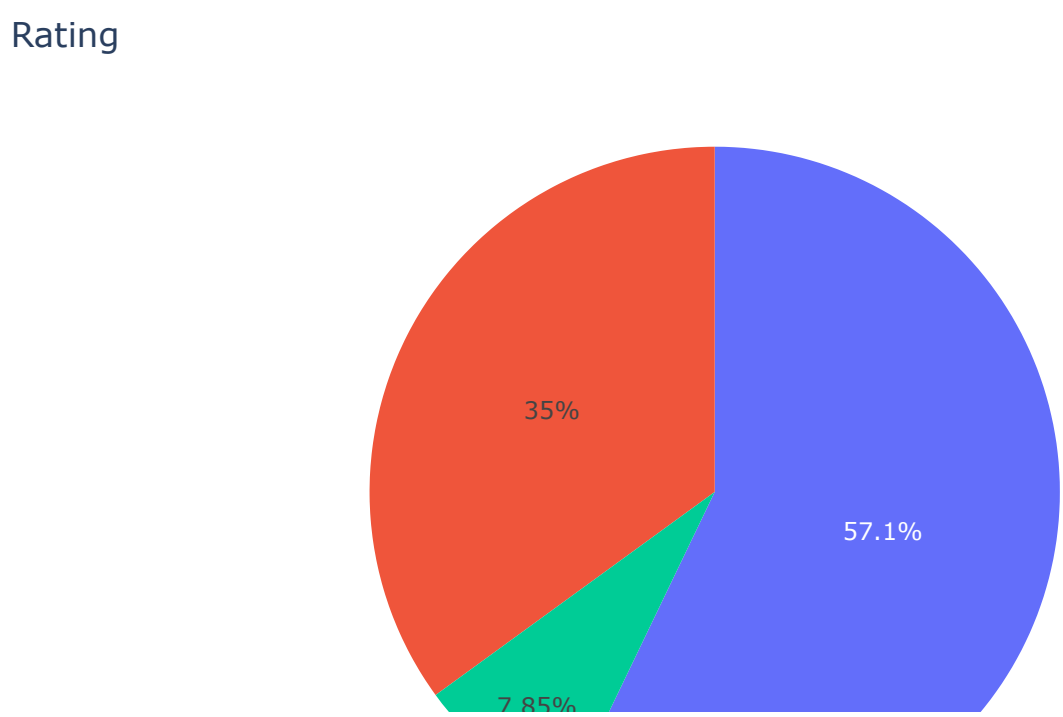
In [46]:

```
fig = go.Figure(data=[go.Bar(x=df_new['rating'].value_counts().index, y=df_new['rating']
fig.update_layout(title='Rating',xaxis_title="Rating",yaxis_title="Count")
fig.show()
```



In [47]:

```
counts = df_new['rating'].value_counts()
fig = go.Figure(data=[go.Pie(labels=counts.index, values=counts)])
fig.update_layout(title='Rating')
fig.show()
```



In [48]:

```
def clean_text(text):
    text = text.lower()
    return text.strip()
```

In [49]:

```
df_new.review_description = df_new.review_description.apply(lambda x: clean_text(x))
```

In [50]:

```
import string
string.punctuation
```

Out[50]:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [51]:

```
def remove_punctuation(text):
    punctuationfree="".join([i for i in text if i not in string.punctuation])
    return punctuationfree
```

In [52]:

```
df_new.review_description = df_new.review_description.apply(lambda x:remove_punctuation(
```

In [53]:

```
df_new.review_description = df_new.review_description.apply(lambda x: x.lower())
```

In [54]:

```
import re
```

In [55]:

```
def tokenization(text):
    tokens = re.split('W+',text)
    return tokens
```

In [56]:

```
df_new.review_description = df_new.review_description.apply(lambda x: tokenization(x))
```

In [57]:

```
import nltk
nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words('english')
stopwords[0:10]
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\hp5cd\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[57]:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "yo
u're"]
```

In [58]:

```
def remove_stopwords(text):
    output= " ".join(i for i in text if i not in stopwords)
    return output
```

In [59]:

```
df_new.review_description = df_new.review_description.apply(lambda x:remove_stopwords(x)
```

In [63]:

```
def clean_text(text):
    text = re.sub('\[.*\]','', text).strip()
    text = re.sub('\S*\d\S*\s*','', text).strip()
    return text.strip()
```

In [64]:

```
df_new.review_description = df_new.review_description.apply(lambda x: clean_text(x))
```

In [65]:

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

In [66]:

```
stopwords = nlp.Defaults.stop_words
def lemmatizer(text):
    doc = nlp(text)
    sent = [token.lemma_ for token in doc if not token.text in set(stopwords)]
    return ' '.join(sent)
```

In [67]:

```
df_new.review_description =  df_new.review_description.apply(lambda x: lemmatizer(x))
```

In [68]:

```
def remove_urls(vTEXT):
    vTEXT = re.sub(r'(https|http)?:\/\/(\w|\.|\/|\?|\=|\&|\%)*\b', '', vTEXT, flags=re.M
    return(vTEXT)
```

In [69]:

```
df_new.review_description = df_new.review_description.apply(lambda x: remove_urls(x))
```

In [70]:

```
def remove_digits(text):
    clean_text = re.sub(r"\b[0-9]+\b\s*", "", text)
    return(text)
```

In [71]:

```
df_new.review_description = df_new.review_description.apply(lambda x: remove_digits(x))
```

In [72]:

```
def remove_digits1(sample_text):
    clean_text = " ".join([w for w in sample_text.split() if not w.isdigit()])
    return(clean_text)
```

In [73]:

```
df_new.review_description = df_new.review_description.apply(lambda x: remove_digits1(x))
```

In [74]:

```
def remove_emojis(data):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"
                               u"\U0001F300-\U0001F5FF"
                               u"\U0001F680-\U0001F6FF"
                               u"\U0001F1E0-\U0001F1FF"
                               "]+", flags=re.UNICODE)
    return re.sub(emoji_pattern, '', data)
```

In [75]:

```
df_new.review_description = df_new.review_description.apply(lambda x: remove_emojis(x))
```

In [76]:

```
df_new
```

Out[76]:

	review_description	rating
0	meh great experience chromebook customize phon...	-1
1	pretty good launch easy use selfexplanatory d ...	0
2	brand new app optimize miss feature app like t...	0
3	great app lot potential lot need fix example o...	0
4	app good need lot functionality example search...	0
...	...	...
32905	kill dog mark zuckerburg strangle dog go	-1
32906	add search hashtag like twitter	-1
32907	bad twister	-1
32908	trash meta	-1
32909	special app copy twitter	-1

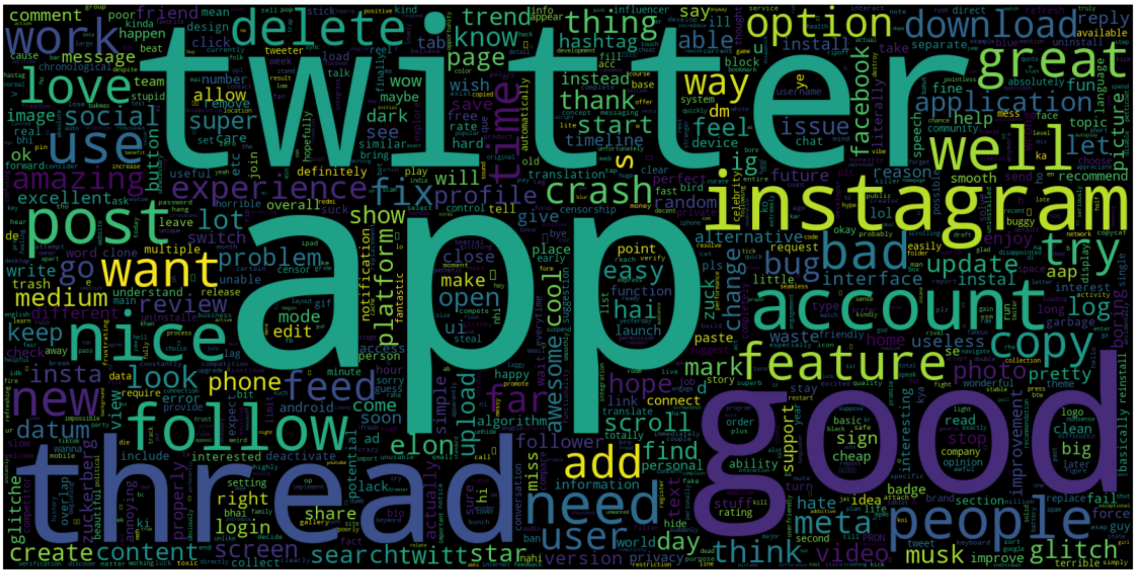
32909 rows × 2 columns

In [77]:

```python
import wordcloud
```

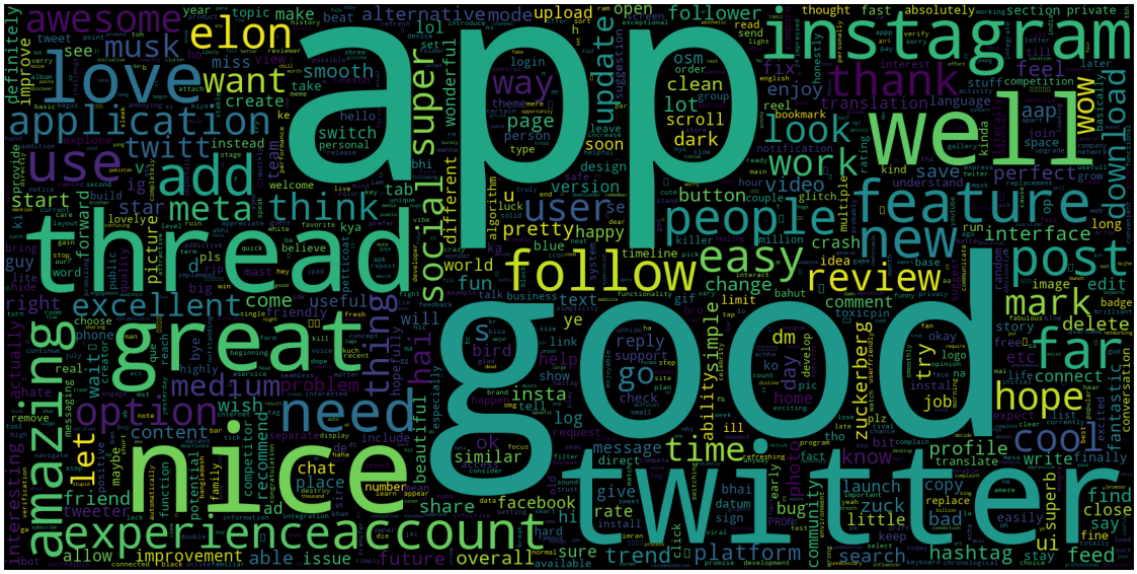
In [78]:

```python
from wordcloud import WordCloud
data = df_new.review_description
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data))
plt.imshow(wc)
plt.axis('off')
plt.show()
```



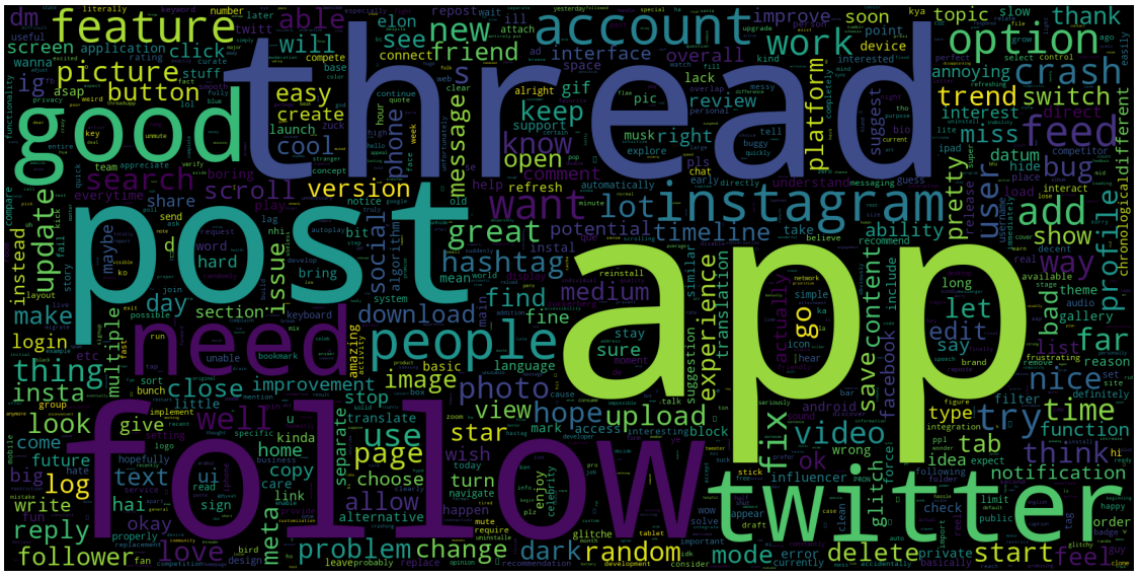
In [79]:

```
from wordcloud import WordCloud
data = df_new[df_new['rating'] == 1]['review_description']
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data))
plt.imshow(wc)
plt.axis('off')
plt.show()
```



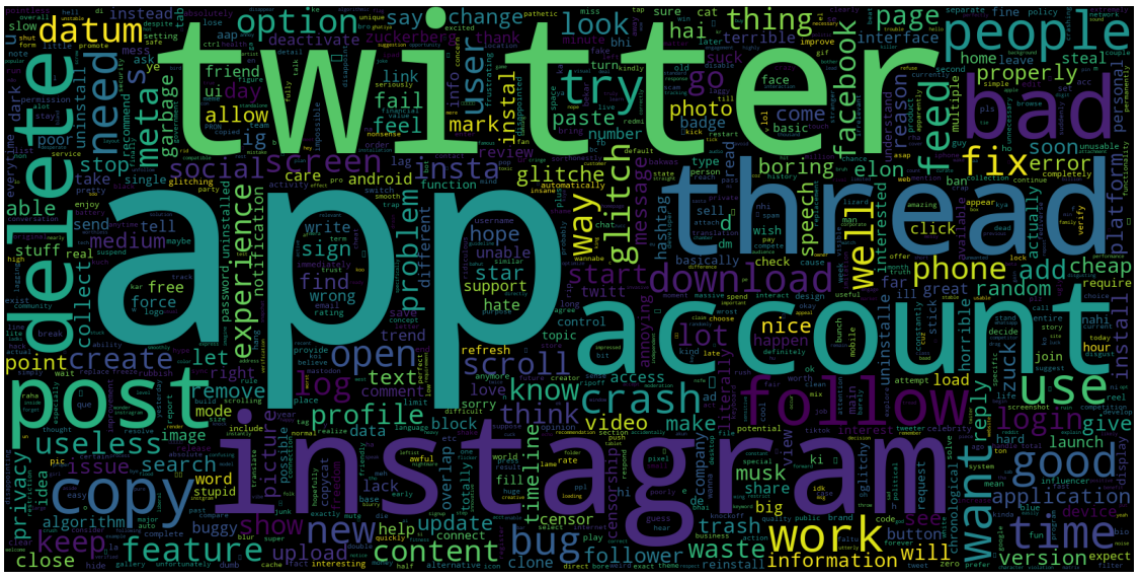
In [80]:

```
data = df_new[df_new['rating'] == 0]['review_description']
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data))
plt.imshow(wc)
plt.axis('off')
plt.show()
```



In [81]:

```
data = df_new[df_new['rating'] == -1]['review_description']
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data))
plt.imshow(wc)
plt.axis('off')
plt.show()
```



In [82]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [84]:

```
tf1=TfidfVectorizer()
data_vec=tf1.fit_transform(df_new['review_description'])
```

In [85]:

```
y=df_new['rating'].values
```

In [86]:

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
```

In [89]:

```
X_train,X_test,y_train,y_test=train_test_split(data_vec,y,test_size=0.2,stratify = y, ra
```

In [90]:

```
from imblearn.over_sampling import SMOTE
```

In [91]:

```
smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X_train, y_train)
```

In [93]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [95]:

```
sv = SVC()
dt = DecisionTreeClassifier()
rf = RandomForestClassifier()
ad = AdaBoostClassifier()

models = [sv, dt, rf, ad]

accuracies = []

for model in models:
    print('Results for the model:', model.__class__.__name__)
    model.fit(X_balanced, y_balanced)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print('Accuracy:', accuracy)

    cm = confusion_matrix(y_test, y_pred)
    print('Confusion Matrix:\n', cm)

    report = classification_report(y_test, y_pred)
    print('Classification Report:\n', report)

    print('\n')

    accuracies.append(accuracy)

print('List of Accuracies:', accuracies)
```

```
Results for the model: SVC
Accuracy: 0.7572166514737162
Confusion Matrix:
 [[1703  103  498]
 [ 207   63  247]
 [ 256  287 3218]]
Classification Report:
              precision    recall  f1-score   support

          -1       0.79      0.74      0.76      2304
           0       0.14      0.12      0.13       517
           1       0.81      0.86      0.83      3761

    accuracy                           0.76      6582
   macro avg       0.58      0.57      0.58      6582
weighted avg       0.75      0.76      0.75      6582



Results for the model: DecisionTreeClassifier
Accuracy: 0.7046490428441203
Confusion Matrix:
 [[1470  251  583]
 [ 190  122  205]
 [ 442  273 3046]]
Classification Report:
              precision    recall  f1-score   support

          -1       0.70      0.64      0.67      2304
           0       0.19      0.24      0.21       517
           1       0.79      0.81      0.80      3761

    accuracy                           0.70      6582
   macro avg       0.56      0.56      0.56      6582
weighted avg       0.71      0.70      0.71      6582



Results for the model: RandomForestClassifier
Accuracy: 0.742631419021574
Confusion Matrix:
 [[1702  143  459]
 [ 237  104  176]
 [ 438  241 3082]]
Classification Report:
              precision    recall  f1-score   support

          -1       0.72      0.74      0.73      2304
           0       0.21      0.20      0.21       517
           1       0.83      0.82      0.82      3761

    accuracy                           0.74      6582
   macro avg       0.59      0.59      0.59      6582
weighted avg       0.74      0.74      0.74      6582



Results for the model: AdaBoostClassifier
Accuracy: 0.54922515952598
Confusion Matrix:
 [[1201  870  233]
```

```
 [ 145  251  121]
 [ 384 1214 2163]]
Classification Report:
              precision    recall  f1-score   support

          -1       0.69      0.52      0.60      2304
           0       0.11      0.49      0.18       517
           1       0.86      0.58      0.69      3761

    accuracy                           0.55      6582
   macro avg       0.55      0.53      0.49      6582
weighted avg       0.74      0.55      0.62      6582



List of Accuracies: [0.7572166514737162, 0.7046490428441203, 0.74263141902
1574, 0.54922515952598]
```

In [97]:

```
model_names = ['SVC', 'DecisionTree', 'RandomForest', 'AdaBoost']
fig = go.Figure(data=go.Bar(x=model_names, y=accuracies))
fig.update_layout(title='Model Accuracies',
                  xaxis_title='Model',
                  yaxis_title='Accuracy',
                  yaxis_tickformat='.2%',
                  yaxis_range=[0, 1],
                  xaxis_tickangle=0)
fig.show()
```