Introduction

Build a model that can predict customers' Long Term Value (LTV).

Import Necessary Libraries and Packages

In [ ]:
```
pip install klib -q
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.1/133.1 KB
3.9 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that a
re installed. This behaviour is the source of the following dependency conflicts.
notebook 5.7.16 requires jinja2<=3.0.0, but you have jinja2 3.1.2 which is incompatible.
flask 1.1.4 requires Jinja2<3.0,>=2.10.1, but you have jinja2 3.1.2 which is incompatible.
```

In [ ]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # visualization
from matplotlib import pyplot as plt # visualization
import klib
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import re
from google.cloud import bigquery
```

Setup project id for to connect with google bigquery

In [ ]:
```
PROJECT_ID = "my-project-1480770657959"
if PROJECT_ID == "" or PROJECT_ID is None:
  PROJECT_ID = "PUT_YOUR_PROJECT_ID_HERE"

PROJECT_ID
```

Out[ ]:
```
'my-project-1480770657959'
```

In [ ]:
```
import os
import sys

# If you are running this notebook in Colab, run this cell and follow the
# instructions to authenticate your GCP account. This provides access to your
# Cloud Storage bucket and lets you submit training jobs and prediction
# requests.

# The Google Cloud Notebook product has specific requirements
IS_GOOGLE_CLOUD_NOTEBOOK = os.path.exists("/opt/deeplearning/metadata/env_version")

# If on Google Cloud Notebooks, then don't execute this code
if not IS_GOOGLE_CLOUD_NOTEBOOK:
    if "google.colab" in sys.modules:
        from google.colab import auth as google_auth

        google_auth.authenticate_user()

    # If you are running this notebook locally, replace the string below with the
    # path to your service account key and run this cell to authenticate your GCP
    # account.
    elif not os.getenv("IS_TESTING"):
        %env GOOGLE_APPLICATION_CREDENTIALS ''
```

In [ ]:
```
#Client manages connections to the BigQuery API and helps
#bundle configuration (project, credentials) needed for API requests.
client = bigquery.Client(PROJECT_ID)

# to make sure all columns are displayed while working with dataframe
pd.set_option('display.max_columns', None)
```

Dataset and table name

In [ ]:
```
PROJECT_ID_DATA = "mh-hackathon"
DATASET_ID_DATA = "ga4_data"
TABLE_ID_TRAIN  = "ga4_train"
TABLE_ID_TEST  = "ga4_test"
START_DATE = "20201101"
END_DATE = "20210131"
```

Function for boxplot

In [ ]:
```
def box_plot(df,col,rot=None):
  _=plt.figure(figsize=(8,6))
  _=sns.boxplot(y=df[col])
  _=plt.title(col.capitalize()+" Distribution",fontsize=25)
  _=plt.ylabel(col,fontsize=20,rotation=rot)
  _=plt.yticks(fontsize=14)
```

Function to show group by numberical values distribution in density and box plot.

In [ ]:
```
def side_by_side_plot(df,grp,valcol,rot=None,title=""):
    clr="Paired"
    fig,(ax1,ax2) = plt.subplots(1,2,figsize=(18,6))
    fig.tight_layout()
    sns.kdeplot(x=df[valcol], hue=df[grp],ax=ax1,palette=clr)
    ax1.set_title(grp.capitalize()+" Wise "+title+" Distribution",size=15)
    ax1.set_xlabel(valcol,fontsize=20)
    sns.boxplot(x=df[grp],y=df[valcol],ax=ax2,palette=clr)
    ax2.set_title(grp.capitalize()+" Wise "+title+" Distribution",size=15)
    ax2.set_xlabel(grp,fontsize=20)
    ax2.tick_params(rotation=rot)
```

Let's see the table structure

In [ ]:
```
query = f"""
SELECT
  DISTINCT(column_name),
  data_type
FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.INFORMATION_SCHEMA.COLUMNS`
"""

query_job = client.query(query)
predict_data = query_job.to_dataframe()
predict_data
```

Out[ ]:

	column_name	data_type
0	user_pseudo_id	STRING
1	event_date	STRING
2	event_timestamp	INT64
3	event_name	STRING
4	event_params	ARRAY<STRUCT<key STRING, value STRUCT<string_v...
5	event_previous_timestamp	INT64
6	event_value_in_usd	FLOAT64
7	event_bundle_sequence_id	INT64
8	event_server_timestamp_offset	INT64
9	user_id	STRING
10	privacy_info	STRUCT<analytics_storage INT64, ads_storage IN...
11	user_properties	ARRAY<STRUCT<key INT64, value STRUCT<string_va...
12	user_first_touch_timestamp	INT64
13	device	STRUCT<category STRING, mobile_brand_name STRI...
14	geo	STRUCT<continent STRING, sub_continent STRING,...
15	app_info	STRUCT<id STRING, version STRING, install_stor...
16	traffic_source	STRUCT<medium STRING, name STRING, source STRING>
17	stream_id	INT64
18	platform	STRING
19	event_dimensions	STRUCT<hostname STRING>
20	ecommerce	STRUCT<total_item_quantity INT64, purchase_rev...
21	items	ARRAY<STRUCT<item_id STRING, item_name STRING,...
22	ltv	FLOAT64

Let's see the shape of the data

In [ ]:

```
query=f"""
SELECT
  count(*)
FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}`
"""
query_job = client.query(query)
query_job.to_dataframe()
```

Out[ ]:

	f0_
0	3859763

The train dataset has 3 million rows

Let's see how many unique psuedo user id in train dataset

In [ ]:

```
query=f"""
SELECT
```

```
  count(distinct user_pseudo_id) as unique_user_pseudo_id
FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}`
"""
query_job = client.query(query)
query_job.to_dataframe()
```

Out[ ]:

	unique_user_pseudo_id
0	243394

The train dataset has 243394 unique user pseudo id.

Let's see the total null values in by string, int64, float64 data types columns.

In [ ]:

```
def query_to_df(field_list):
  null_count=[]
  for param in field_list:
    query=f"""SELECT
    sum(CASE WHEN {param} IS NULL THEN 1 ELSE 0 END) as {re.sub(".","_",param)}
    FROM
      `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}`
    """
    query_job = client.query(query)
    df=query_job.to_dataframe()
    null_count.append(df.iloc[:,[0]].values[0][0])
  return pd.DataFrame({'column_name':field_list,'null_value_count':null_count})
```

In [ ]:

```
def null_count_plot(df,x,y):
  _=plt.figure(figsize=(12,6))
  sns.barplot(x=df[x],y=df[y]);
```

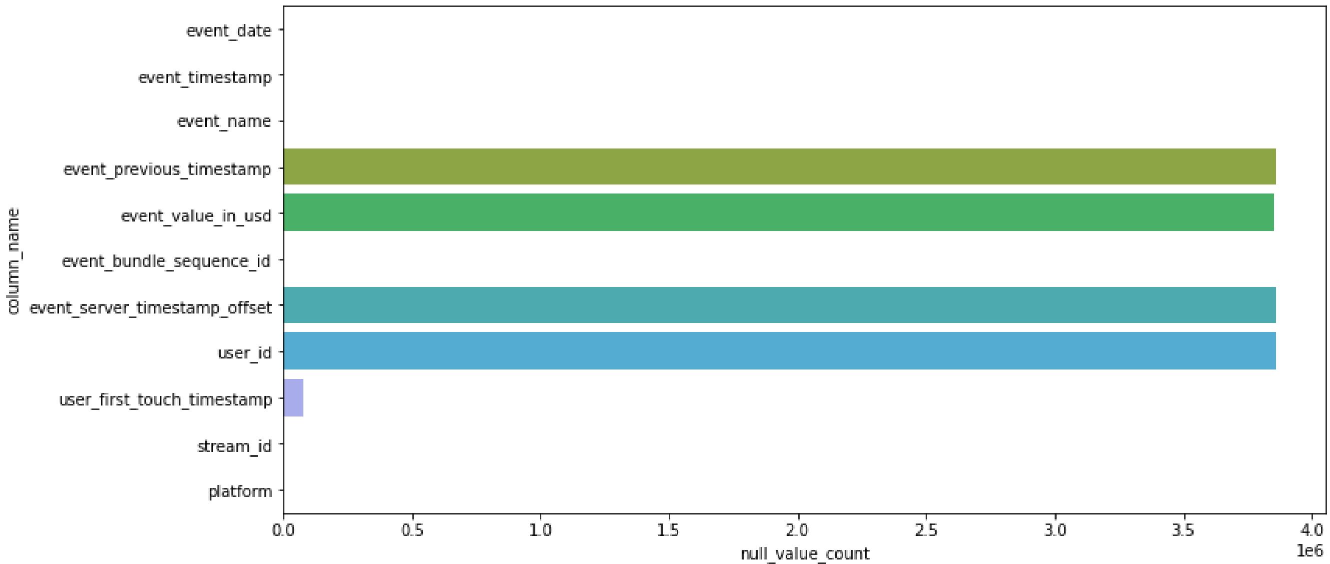
In [ ]:

```
col_list=['event_date', 'event_timestamp', 'event_name', 'event_previous_timestamp',
            'event_value_in_usd',
            'event_bundle_sequence_id',
            'event_server_timestamp_offset',
            'user_id',
            'user_first_touch_timestamp',
            'stream_id', 'platform']

null_count_by_col=query_to_df(col_list)
```

In [ ]:

```
null_count_plot(null_count_by_col,'null_value_count','column_name')
```



In [ ]:
```
null_count_by_col
```

Out[ ]:

	column_name	null_value_count
0	event_date	0
1	event_timestamp	0
2	event_name	0
3	event_previous_timestamp	3859763
4	event_value_in_usd	3855007
5	event_bundle_sequence_id	0
6	event_server_timestamp_offset	3859763
7	user_id	3859763
8	user_first_touch_timestamp	78423
9	stream_id	0
10	platform	0

The above information explain that the columns event_previous_timestamp, event_value_in_usd, event_server_timestamp_offset,user_id have almost null values for all rows.

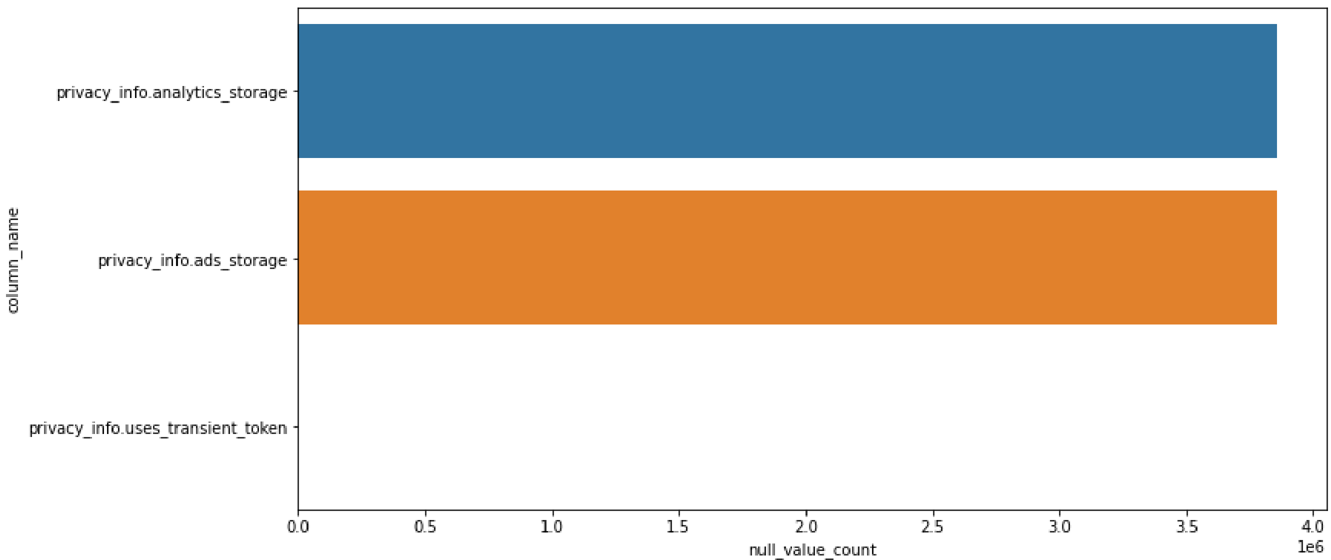
The user_first_touch_timestamp column has 78423 null values.

Let's see the structure datatype columns and null value count of each field.

Let's see the privacy info column and its fields' null value count.

In [ ]:
```
field_list=['privacy_info.analytics_storage','privacy_info.ads_storage','privacy_info.uses_tr
privacy_info_null_count=query_to_df(field_list)
```

In [ ]:
```
null_count_plot(privacy_info_null_count,'null_value_count','column_name')
```



In [ ]:
```
privacy_info_null_count
```

Out[ ]:

	column_name	null_value_count
0	privacy_info.analytics_storage	3859763
1	privacy_info.ads_storage	3859763
2	privacy_info.uses_transient_token	0

The above information shows that the privacy_info.analytics_storage, privacy_info.ads_storage columns non null values is zero.

Let's see the device column and its field null value count.

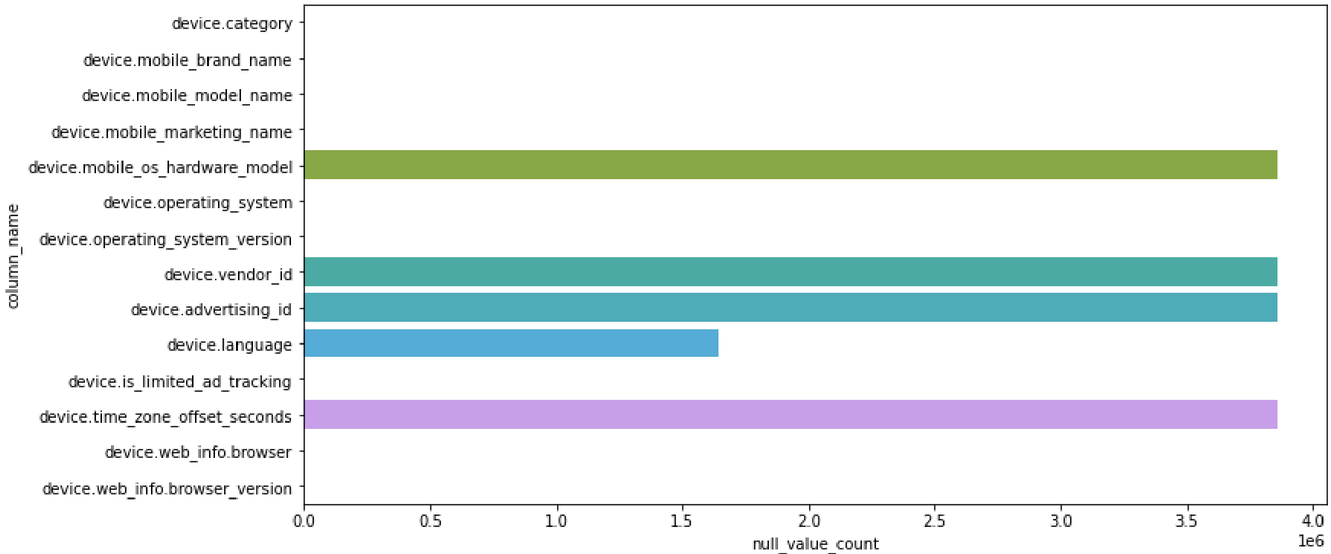
In [ ]:

```
field_list=['device.category','device.mobile_brand_name','device.mobile_model_name',
            'device.mobile_marketing_name','device.mobile_os_hardware_model',
            'device.operating_system','device.operating_system_version',
            'device.vendor_id','device.advertising_id','device.language','device.is_limited_a
            'device.time_zone_offset_seconds','device.web_info.browser','device.web_info.brow

device_null_count=query_to_df(field_list)
```

In [ ]:

```
null_count_plot(device_null_count,'null_value_count','column_name')
```



In [ ]:

```
device_null_count
```

Out[ ]:

	column_name	null_value_count
0	device.category	0
1	device.mobile_brand_name	0
2	device.mobile_model_name	0
3	device.mobile_marketing_name	0
4	device.mobile_os_hardware_model	3859763
5	device.operating_system	0
6	device.operating_system_version	0
7	device.vendor_id	3859763
8	device.advertising_id	3859763
9	device.language	1645027
10	device.is_limited_ad_tracking	0
11	device.time_zone_offset_seconds	3859763
12	device.web_info.browser	0
13	device.web_info.browser_version	0

The above information exlain that the device.mobile_os_hardware_model, device.vendor_id, device.advertising_id , device.time_zone_offset_seconds parameter's non null value is zero.

device. language parameter has 1645027 null values.

Let's see the geo column and its field null value count.

In [ ]:
```
field_list=['geo.continent','geo.sub_continent','geo.country','geo.region','geo.city','geo.me
geo_null_count=query_to_df(field_list)
```

In [ ]:
```
geo_null_count
```

Out[ ]:

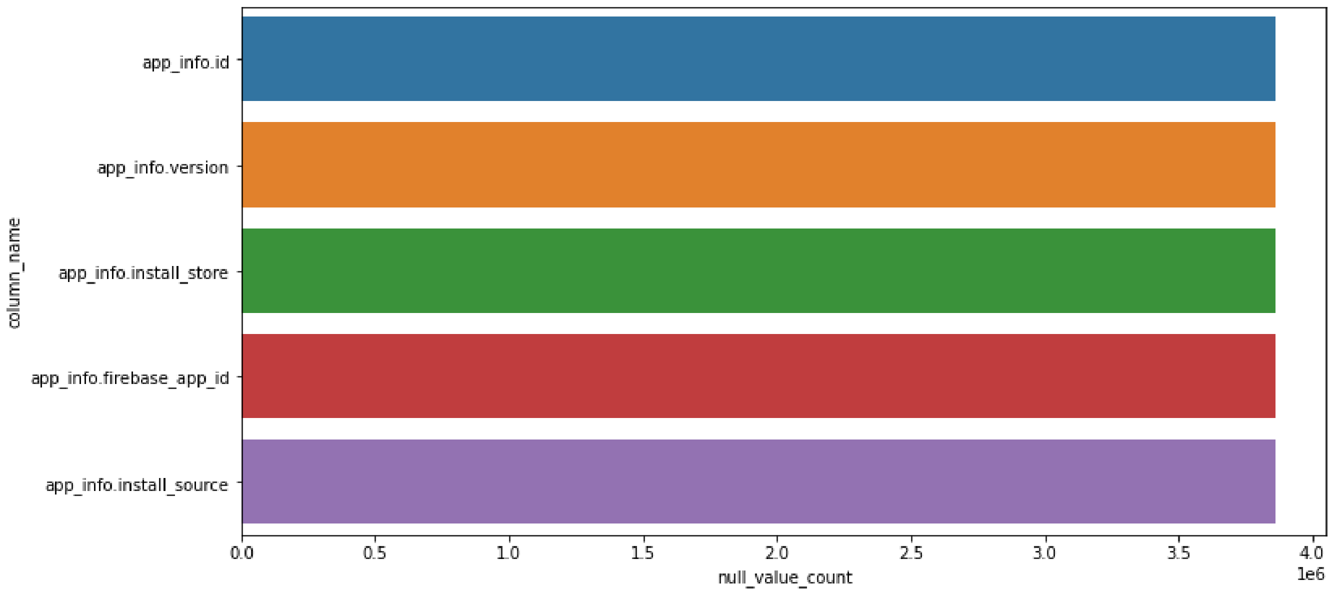
	column_name	null_value_count
0	geo.continent	0
1	geo.sub_continent	0
2	geo.country	0
3	geo.region	0
4	geo.city	0
5	geo.metro	0

The above information explains that the geo fields don't have any null values.

Let's see the app_info column and its field null value count.

In [ ]:
```
field_list=['app_info.id','app_info.version','app_info.install_store','app_info.firebase_app_
app_info_null_count=query_to_df(field_list)
```

In [ ]:
```
null_count_plot(app_info_null_count,'null_value_count','column_name')
```



In [ ]:
```
app_info_null_count
```

Out[ ]:

	column_name	null_value_count
0	app_info.id	3859763
1	app_info.version	3859763
2	app_info.install_store	3859763
3	app_info.firebase_app_id	3859763
4	app_info.install_source	3859763

The above information explains that all the app_info fields don't have any values.

Let's see the traffic_source column and its field null value count.

In [ ]:
```
field_list=['traffic_source.medium','traffic_source.name','traffic_source.source']
traffic_source_null_count=query_to_df(field_list)
```

In [ ]:
```
traffic_source_null_count
```

Out[ ]:

	column_name	null_value_count
0	traffic_source.medium	0
1	traffic_source.name	0
2	traffic_source.source	0

The above information explains that traffic_source column fields don't have any null values.

Let's see event_diemesntions column null value count.

In [ ]:
```
query=f"""SELECT
  sum(CASE WHEN event_dimensions.hostname IS NULL THEN 1 ELSE 0 END) as event_dimensions_host
  FROM
    `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}`
  """
query_job = client.query(query)
query_job.to_dataframe()
```

Out[ ]:

	event_dimensions_hostname
0	3859763

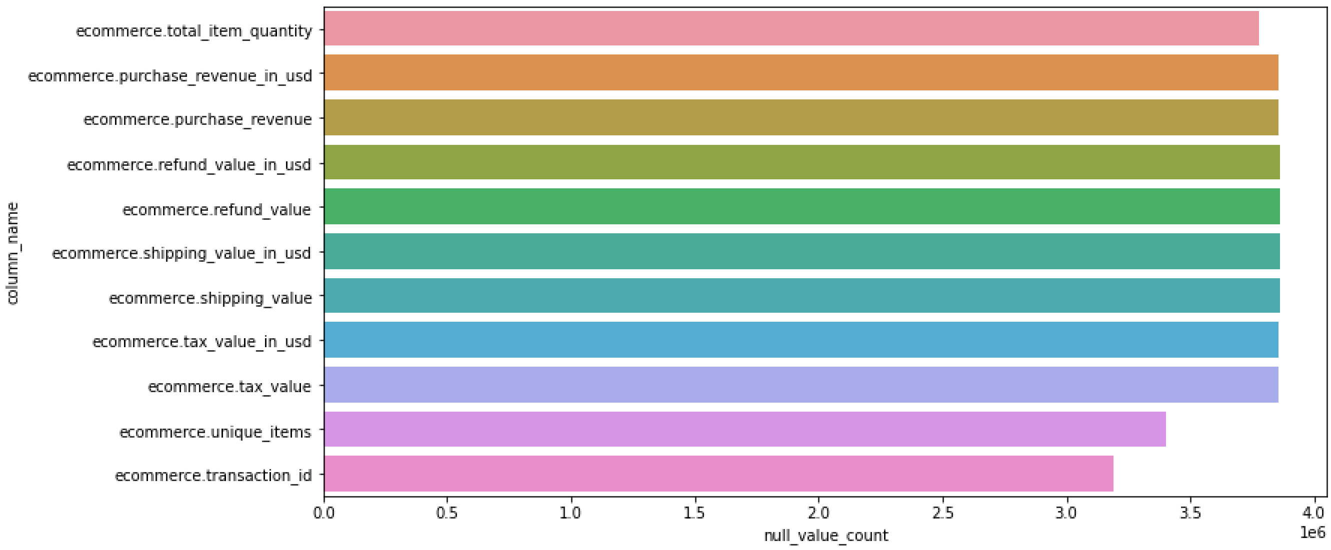
The above information exlains that the event_dimensions column has zero non null value.

Let's see the ecommerce column and its field null value count.

In [ ]:
```
field_list=['ecommerce.total_item_quantity','ecommerce.purchase_revenue_in_usd','ecommerce.pu
            'ecommerce.refund_value_in_usd','ecommerce.refund_value','ecommerce.shipping_valu
            'ecommerce.shipping_value','ecommerce.tax_value_in_usd','ecommerce.tax_value',
            'ecommerce.unique_items','ecommerce.transaction_id']

ecommerce_null_count=query_to_df(field_list)
```

In [ ]:
```
null_count_plot(ecommerce_null_count,'null_value_count','column_name')
```



In [ ]:
```
ecommerce_null_count
```

Out[ ]:

	column_name	null_value_count
0	ecommerce.total_item_quantity	3775348
1	ecommerce.purchase_revenue_in_usd	3854613
2	ecommerce.purchase_revenue	3855007
3	ecommerce.refund_value_in_usd	3859763
4	ecommerce.refund_value	3859763
5	ecommerce.shipping_value_in_usd	3859763
6	ecommerce.shipping_value	3859763
7	ecommerce.tax_value_in_usd	3855007
8	ecommerce.tax_value	3855007
9	ecommerce.unique_items	3399893
10	ecommerce.transaction_id	3190157

In [ ]:
```
ecommerce_null_count.apply(lambda x:x['null_value_count']/3859763,axis=1)
```

Out[ ]:
```
0     0.978129
1     0.998666
2     0.998768
3     1.000000
4     1.000000
5     1.000000
6     1.000000
7     0.998768
8     0.998768
9     0.880855
10    0.826516
dtype: float64
```

The above information explains that all e-commerce fields have more than 80% null values.

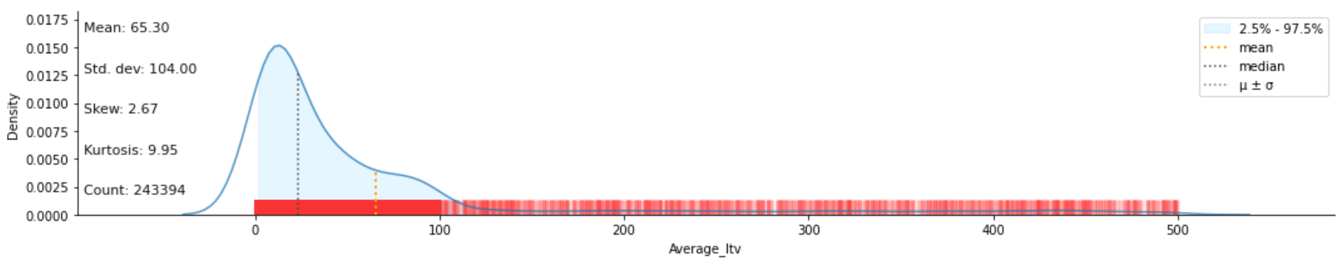
Let's see the users's average long term value distribution

In [ ]:
```
query=f"""
  SELECT
  AVG(ltv) as Average_ltv
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` group by user_pseudo_id
  """
query_job = client.query(query)
avg_ltv = query_job.to_dataframe()
```

In [ ]:
```
klib.dist_plot(avg_ltv['Average_ltv'])
```

```
Large dataset detected, using 10000 random samples for the plots. Summary statistics are stil
l based on the entire dataset.
```

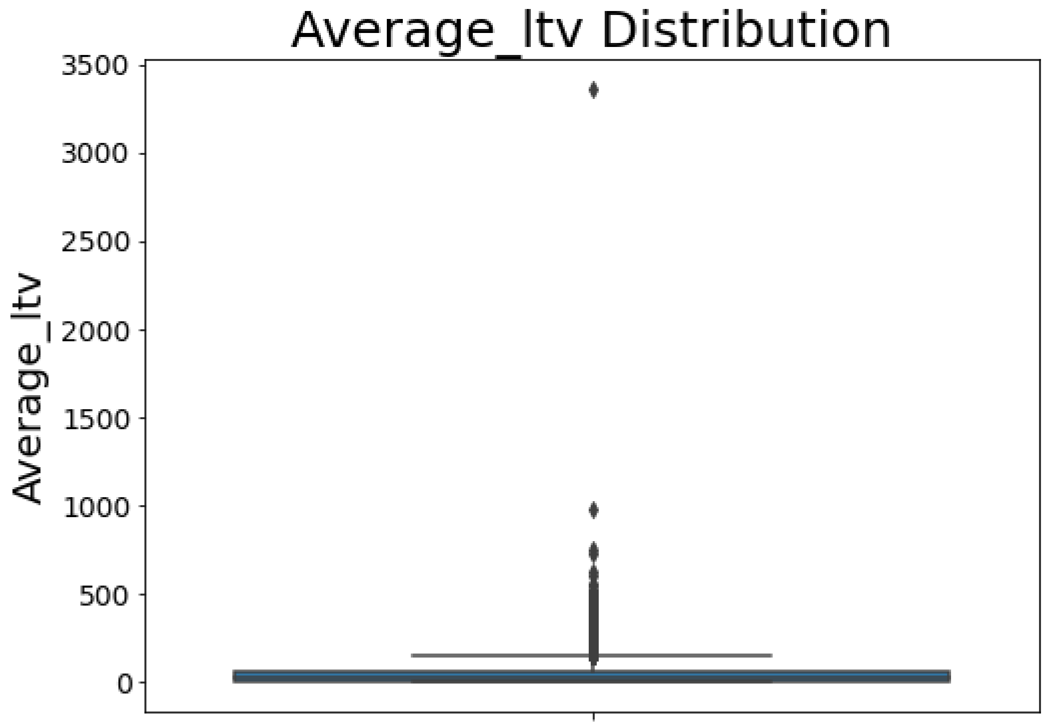
Out[ ]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f983b76ab80>`



In [ ]:
```
avg_ltv['Average_ltv'].describe()
```

Out[ ]:
```
count    243394.000000
mean         65.296333
std         104.004247
min           0.000210
25%          10.328171
50%          23.225712
75%          67.625533
max        3360.000000
Name: Average_ltv, dtype: float64
```

In [ ]:
```
box_plot(avg_ltv,'Average_ltv',rot=90)
```



The above histogram plot and summary explain that the average ltv is right-skewed (positively skewed).

The average ltv ranges from 0.000210 to 3360.

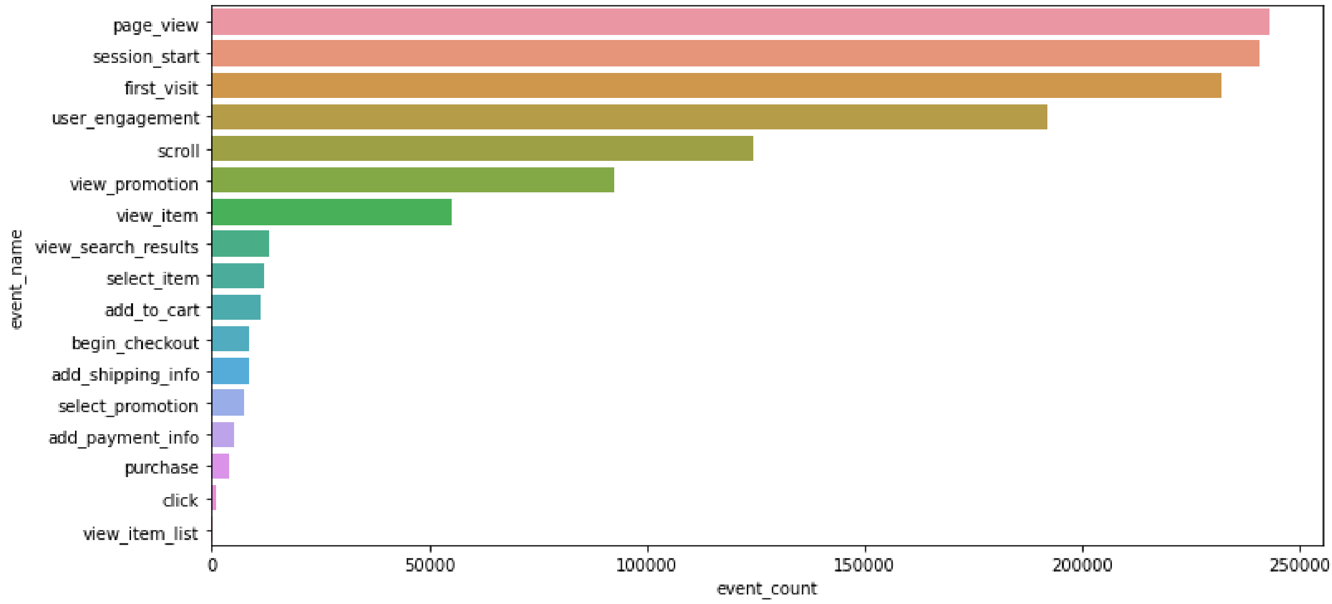
The average of average ltv is 65.296.

The boxplot explains that there are outliers above the third quartile.

Let's see the available event informations

In [ ]:
```
query=f"""
  SELECT event_name,
  count(distinct user_pseudo_id) as event_count
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` group by event_name
  """
query_job=client.query(query)
event_type=query_job.to_dataframe()
event_type=event_type.sort_values('event_count',ascending=False)
```

In [ ]:
```
_=plt.figure(figsize=(12,6))
_=sns.barplot(data=event_type  ,
              y='event_name',x='event_count');
```



In [ ]:
```
event_type
```

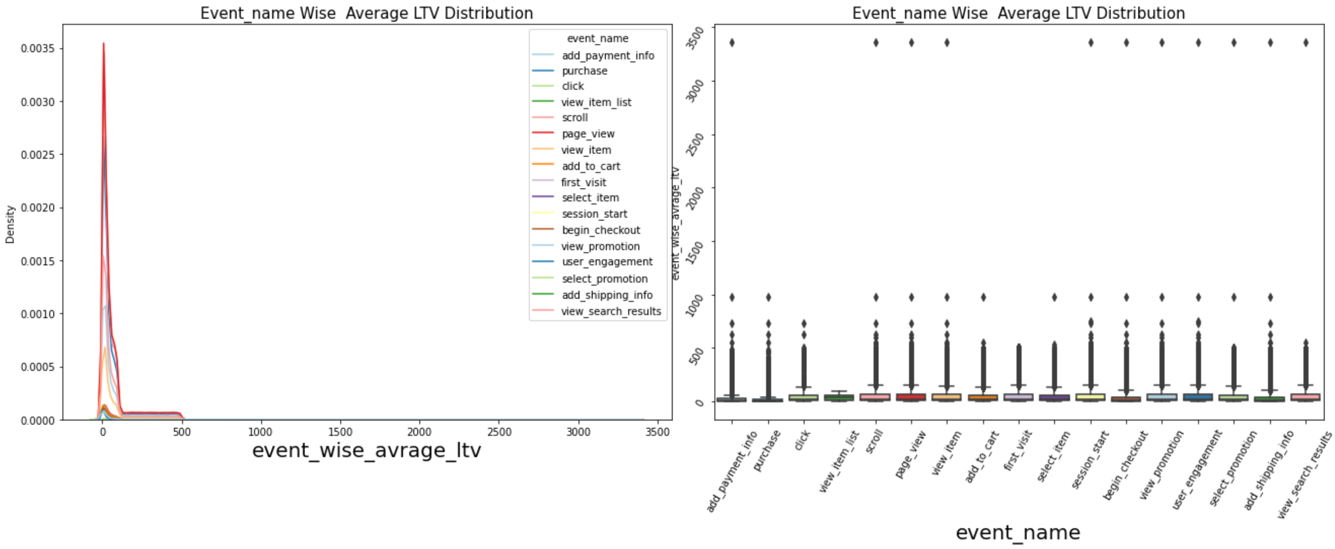
Out[ ]:

	event_name	event_count
3	page_view	243068
1	session_start	240668
4	first_visit	231830
0	user_engagement	191970
2	scroll	124415
6	view_promotion	92300
5	view_item	55207
7	view_search_results	12970
13	select_item	11872
15	add_to_cart	11281
11	begin_checkout	8739
9	add_shipping_info	8738
12	select_promotion	7371
8	add_payment_info	5193
14	purchase	3998
10	click	919
16	view_item_list	37

Let's see the user's average ltv distribution by different events.

In [ ]:
```
query=f"""
  select user_pseudo_id, event_name , avg(ltv) as event_wise_avrage_ltv
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` group by user_pseudo_id, event_name
  """
query_job=client.query(query)
event_avg_ltv=query_job.to_dataframe()
```

In [ ]:
```
side_by_side_plot(event_avg_ltv,'event_name','event_wise_avrage_ltv',rot=60,title=' Average L
```



In [ ]:
```
event_avg_ltv.groupby('event_name')['event_wise_avrage_ltv'].describe()
```

Out[ ]:

event_name	count	mean	std	min	25%	50%	75%	max
add_payment_info	5193.0	30.038233	80.112460	0.012971	1.811321	7.151163	26.089959	3360.000000
add_shipping_info	8738.0	44.592329	92.675723	0.001387	3.408611	13.512791	43.409149	3360.000000
add_to_cart	11281.0	54.350028	94.562943	0.012971	6.297957	17.930704	55.912941	981.194690
begin_checkout	8739.0	44.587928	92.671333	0.001387	3.408771	13.511789	43.406214	3360.000000
click	919.0	61.380490	101.475386	0.112037	10.399877	24.651282	62.004593	731.909091
first_visit	231830.0	65.289657	103.802204	0.000210	10.339539	23.185543	67.625292	499.993208
page_view	243068.0	65.280284	103.981843	0.000210	10.326923	23.217286	67.616204	3360.000000
purchase	3998.0	20.686482	51.057887	0.032252	1.398058	4.323493	16.757347	981.194690
scroll	124415.0	64.288642	103.406325	0.000210	10.078233	22.514066	66.607495	3360.000000
select_item	11872.0	58.297828	97.932479	0.007471	7.878200	19.344283	59.888329	981.194690
select_promotion	7371.0	62.587792	109.511335	0.014876	9.369737	20.393112	62.886939	3360.000000
session_start	240668.0	65.277614	103.986637	0.000210	10.326043	23.213717	67.622594	3360.000000
user_engagement	191970.0	65.096580	103.977261	0.000210	10.240451	23.032826	67.395950	3360.000000
view_item	55207.0	62.444117	102.463257	0.000275	9.383827	21.017276	64.806028	3360.000000
view_item_list	37.0	39.139336	28.784023	0.799886	15.031859	36.622741	58.367239	97.423547
view_promotion	92300.0	64.350569	103.637263	0.000210	9.879568	22.428427	66.784070	3360.000000
view_search_results	12970.0	63.149595	105.672073	0.001387	9.562740	21.768476	66.407390	3360.000000

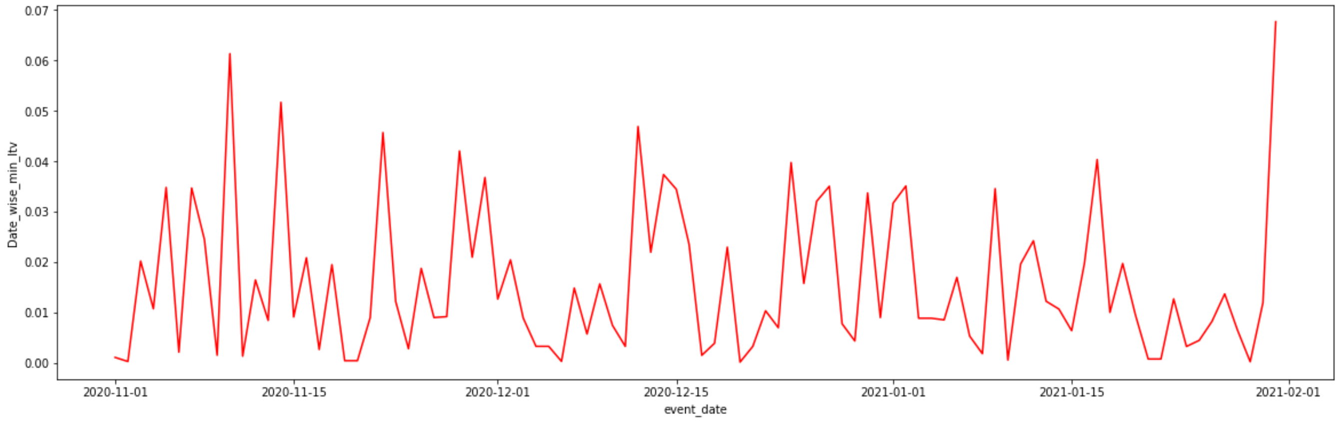
The above plot and summary explain that there is a mean difference between the different events.

Let's the ltv distrubtion by the date

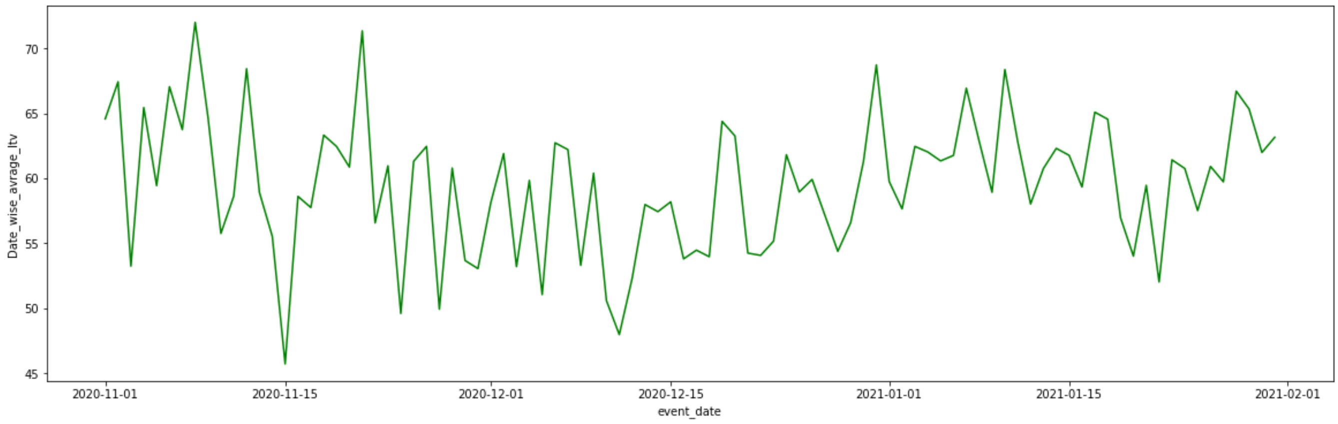
In [ ]:
```
query=f"""
  SELECT event_date,
  min(ltv) as Date_wise_min_ltv,
  avg(ltv) as Date_wise_avrage_ltv,
  max(ltv) as Date_wise_max_ltv
  FROM
```

```
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` group by event_date
  """
query_job=client.query(query)
event_date_ltv=query_job.to_dataframe()
```

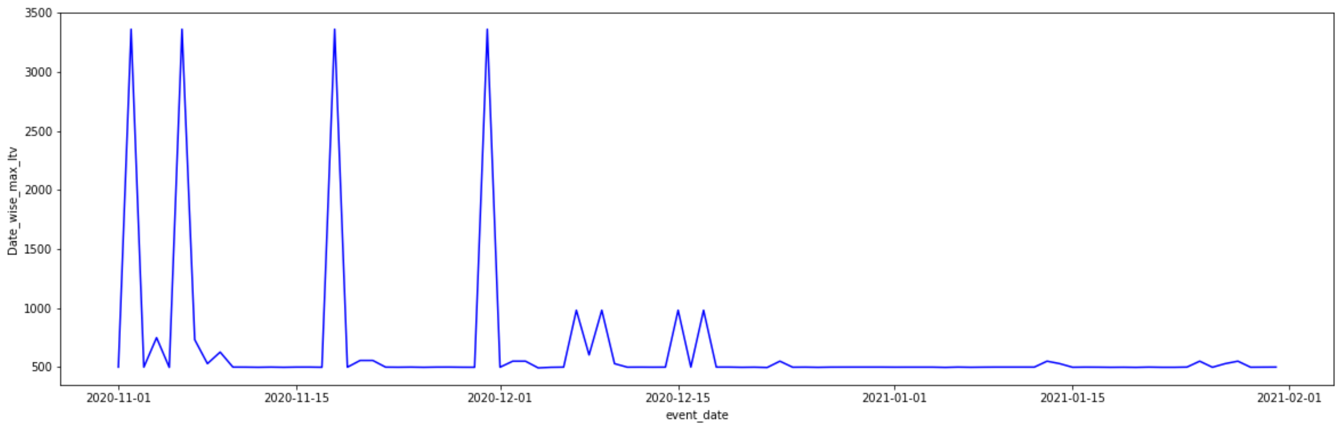
In [ ]:
```
_=plt.figure(figsize=(20,6))
sns.lineplot(x=pd.to_datetime(event_date_ltv['event_date']),y=event_date_ltv['Date_wise_min_l
```



In [ ]:
```
_=plt.figure(figsize=(20,6));
sns.lineplot(x=pd.to_datetime(event_date_ltv['event_date']),y=event_date_ltv['Date_wise_avrag
```



In [ ]:
```
_=plt.figure(figsize=(20,6));
sns.lineplot(x=pd.to_datetime(event_date_ltv['event_date']),y=event_date_ltv['Date_wise_max_l
```



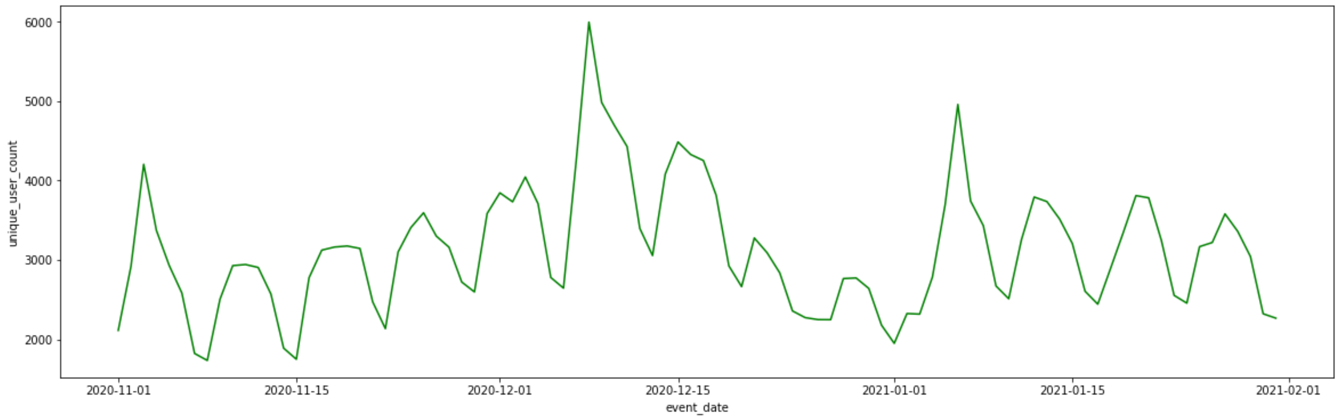
The above plots explains the minimum, average, and maximum ltv of users over the 3 months.

Let's see the number of unique users over the 3 months.

In [ ]:
```
query=f"""
  SELECT event_date,
  count(distinct user_pseudo_id) unique_user_count
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` group by event_date
```

```
    """
query_job=client.query(query)
event_date_unique_user=query_job.to_dataframe()
```

In [ ]:
```
_=plt.figure(figsize=(20,6));
sns.lineplot(x=pd.to_datetime(event_date_unique_user['event_date']),y=event_date_unique_user[
```

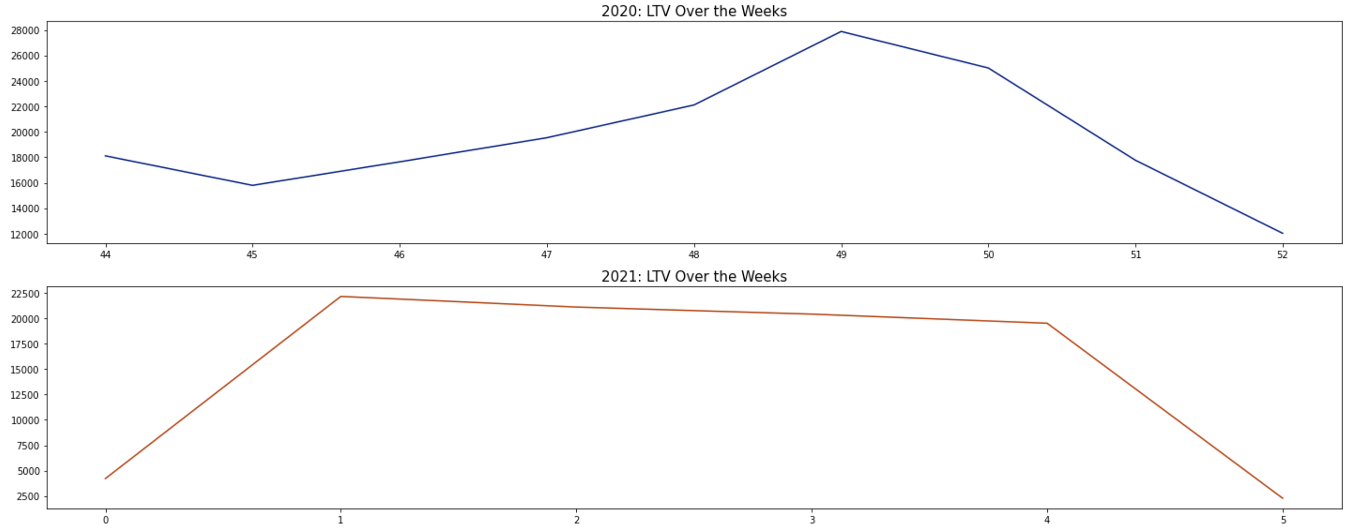


The above plot explains that more user activity was recorded in the month of December '2020 and January '2021.

Let's see the unique users count over the weeks of 3 months.

In [ ]:
```
query=f"""
    select EXTRACT(WEEK FROM TIMESTAMP_MICROS(event_timestamp)) as week,
    EXTRACT(YEAR FROM PARSE_DATE("%Y%m%d",event_date)) as Year,
    count(distinct user_pseudo_id) unique_user_count_week
    FROM
    `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` group by 1,2
    """
query_job=client.query(query)
event_date_week_unique_user=query_job.to_dataframe()
event_date_week_unique_user=event_date_week_unique_user.sort_values('Year',ascending=True)
```

In [ ]:
```
fig=plt.subplots(figsize=(20, 8))
custom_palette = sns.color_palette("dark", 2)
colors=[col for col in custom_palette.as_hex()]
for i,(col,clr) in enumerate(zip(event_date_week_unique_user['Year'].unique(),colors)):
    _=plt.subplot(2,1,i+1)
    data=event_date_week_unique_user[event_date_week_unique_user['Year']==col]
    _=sns.lineplot(x=data['week'],y=data['unique_user_count_week'],color=clr)
    _=plt.title(str(col)+": LTV Over the Weeks",fontsize=15)
    _=plt.xlabel("")
    _=plt.ylabel("")
    _#=plt.xticks(fontsize=15)
    _=plt.tight_layout()
plt.show()
```



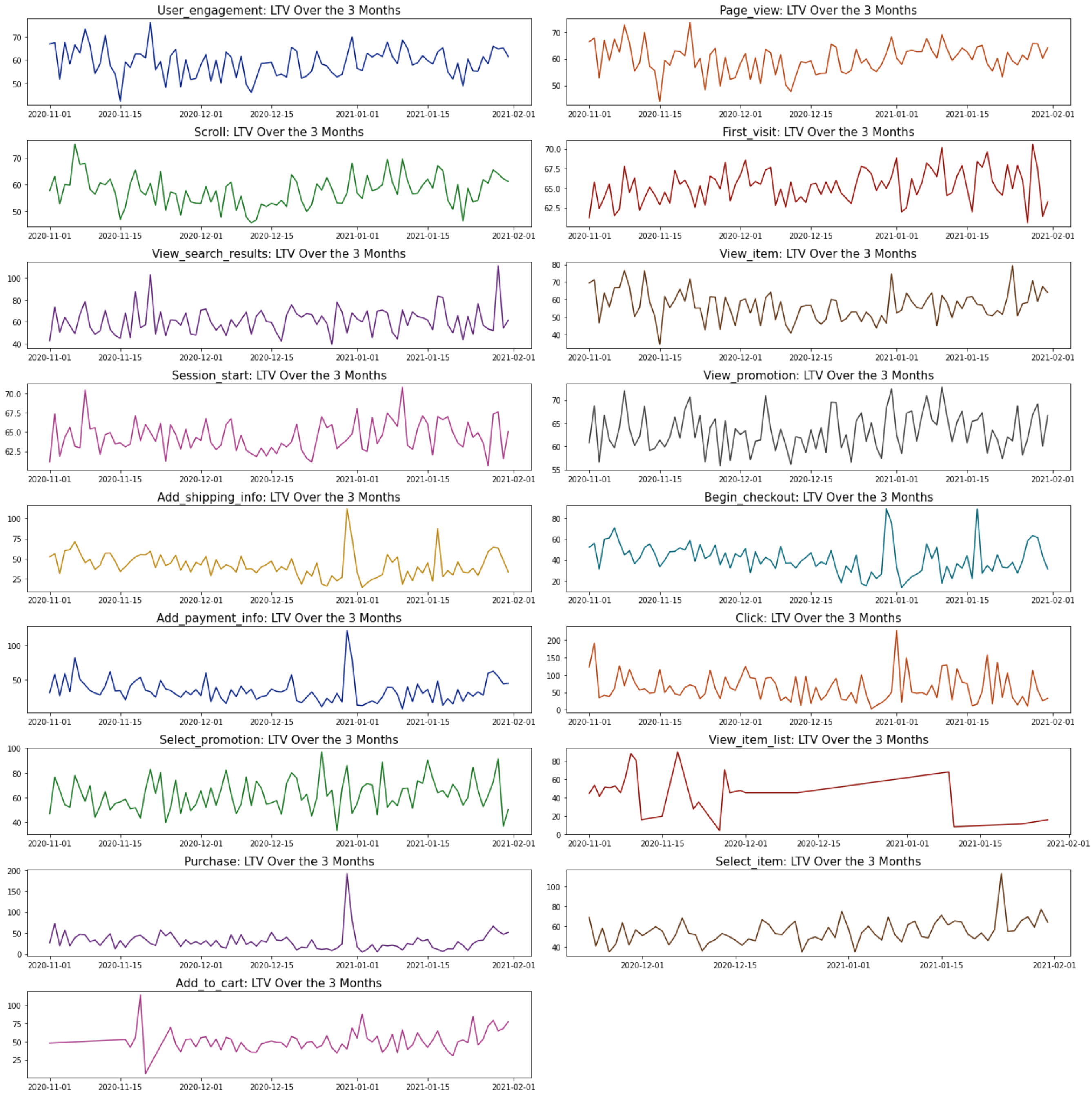
The above plot explains that in the year 2020, more activity was in weeks 49 and 50 of the year.

In the year 2021, more activity was in the weeks from 1 to 4 of the year.

Let's see the users average ltv over the 3 months by different events.

In [ ]:
```
query=f"""
  SELECT event_date,event_name,
  avg(ltv) as Event_type_date_wise_average_ltv,
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` group by event_date,event_name
  """
query_job=client.query(query)
event_type_date_ltv=query_job.to_dataframe()
```

In [ ]:
```
fig=plt.subplots(figsize=(20, 20))
custom_palette = sns.color_palette("dark", 17)
colors=[col for col in custom_palette.as_hex()]
for i,(col,clr) in enumerate(zip(event_type_date_ltv['event_name'].unique(),colors)):
    _=plt.subplot(9,2,i+1)
    data=event_type_date_ltv[event_type_date_ltv['event_name']==col]
    _=sns.lineplot(x=pd.to_datetime(data['event_date']),y=data['Event_type_date_wise_average_
    _=plt.title(col.capitalize()+": LTV Over the 3 Months",fontsize=15)
    _=plt.xlabel("")
    _=plt.ylabel("")
    _#=plt.xticks(fontsize=15)
    _=plt.tight_layout()
plt.show()
```



Let's see the user's average ltv over the hours of the day.

In [ ]:
```
query=f"""
  select EXTRACT(HOUR FROM TIMESTAMP_MICROS(event_timestamp)) as Hour, avg(ltv) as Hour_wise_
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` group by 1
  """
query_job=client.query(query)
hour_ltv=query_job.to_dataframe()
hour_ltv=hour_ltv.sort_values('Hour',ascending=True)
hour_ltv=hour_ltv.replace({'Hour':{0:24}})
```

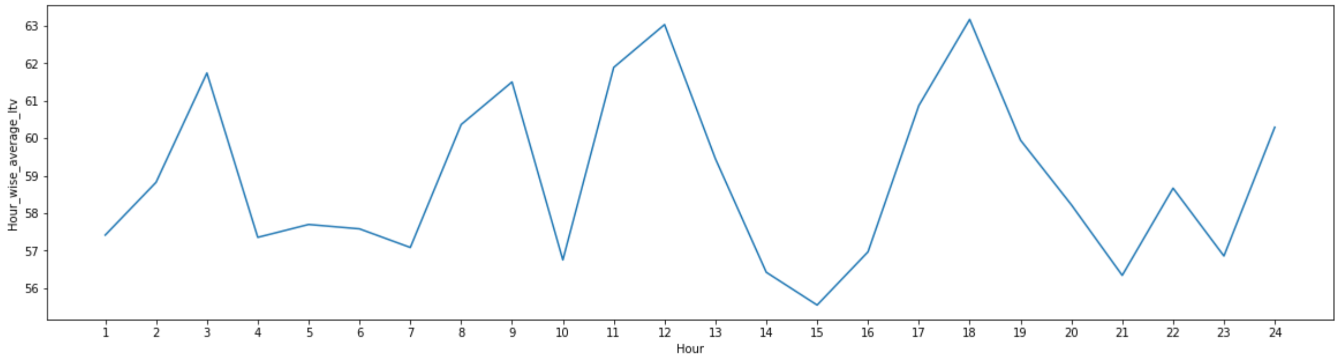
Let's create a function to convert the hour to part of the day.

In [ ]:
```
def hours2timing(x):
    if x in range(20,23):
        timing = 'Night'
    elif x in range(5,12):
        timing = 'Morning'
    elif x in range(12, 16):
        timing = 'Afternoon'
    elif x in range(16, 20):
        timing = 'Evening'
```

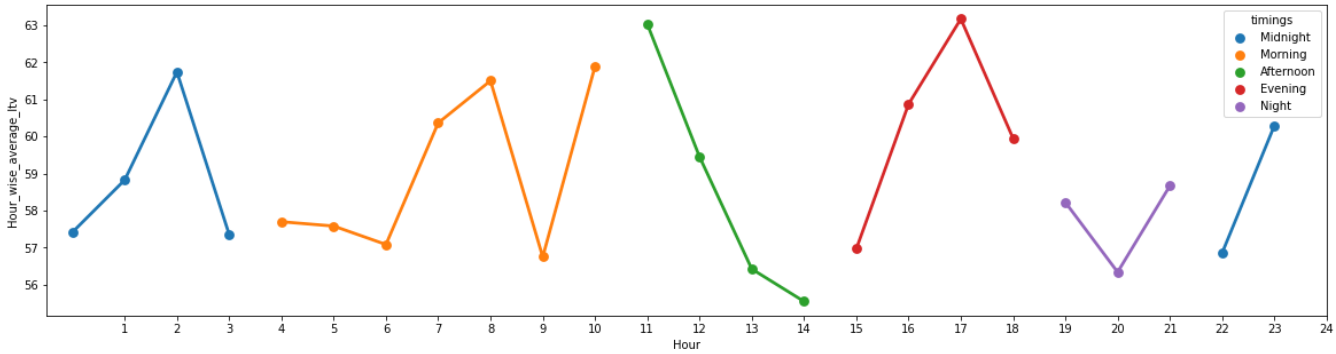
```
    elif x in [23,24,1,2,3,4]:
        timing = 'Midnight'
    else:
        timing = 'X'
    return timing
```

In [ ]:
```
hour_ltv['timings']=hour_ltv['Hour'].apply(hours2timing)
```

In [ ]:
```
_=plt.figure(figsize=(20,5))
sns.lineplot(x=hour_ltv['Hour'],y=hour_ltv['Hour_wise_average_ltv']);
_=plt.xticks([hour for hour in range(1,25)])
```



In [ ]:
```
_=plt.figure(figsize=(20,5))
sns.pointplot(data=hour_ltv,x='Hour',y='Hour_wise_average_ltv',hue='timings');
_=plt.xticks([hour for hour in range(1,25)])
```



The above plot explains that the user's average ltv is higher in the following,

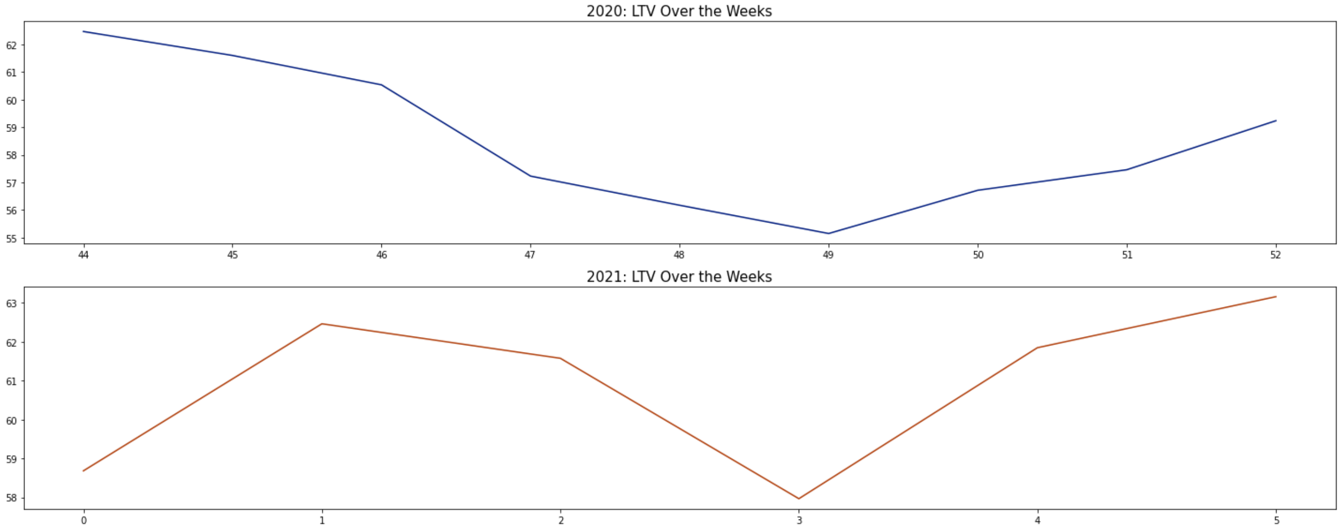
- Midnight 2'o clock
- Morning 8 and 10'o clock
- Afternoon 11'o clock(Highest)
- Evening 5'o clock(Highest)

Let's see the users average ltv over the weeks of 3 months.

In [ ]:
```
query=f"""
  select EXTRACT(WEEK FROM PARSE_DATE("%Y%m%d",event_date)) as Week,
  EXTRACT(YEAR FROM PARSE_DATE("%Y%m%d",event_date)) as Year,
  avg(ltv) as Week_wise_average_ltv,
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` group by 1,2
  """
query_job=client.query(query)
week_ltv=query_job.to_dataframe()
```

In [ ]:
```
fig=plt.subplots(figsize=(20, 8))
custom_palette = sns.color_palette("dark", 2)
colors=[col for col in custom_palette.as_hex()]
for i,(col,clr) in enumerate(zip(week_ltv['Year'].unique(),colors)):
```

```
    _=plt.subplot(2,1,i+1)
    data=week_ltv[week_ltv['Year']==col]
    _=sns.lineplot(x=data['Week'],y=data['Week_wise_average_ltv'],color=clr)
    _=plt.title(str(col)+": LTV Over the Weeks",fontsize=15)
    _=plt.xlabel("")
    _=plt.ylabel("")
    _#=plt.xticks(fontsize=15)
    _=plt.tight_layout()
plt.show()
```



Let's see users average ltv over the day of the 3 months week.

In [ ]:
```
query=f"""
  select EXTRACT(WEEK FROM PARSE_DATE("%Y%m%d",event_date)) as Week,
  EXTRACT(DAYOFWEEK FROM PARSE_DATE("%Y%m%d",event_date)) as day_of_week,
  EXTRACT(YEAR FROM PARSE_DATE("%Y%m%d",event_date)) as year,
  avg(ltv) as day_week_wise_average_ltv,
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` group by 1,2,3 order by year asc,Wee
  """
query_job=client.query(query)
day_week_ltv=query_job.to_dataframe()
```

In [ ]:
```
fig=plt.subplots(figsize=(20, 15))
custom_palette = sns.color_palette("dark", 15)
colors=[col for col in custom_palette.as_hex()]
for i,(col,clr) in enumerate(zip(day_week_ltv['Week'].unique(),colors)):
    _=plt.subplot(5,3,i+1)
    data=day_week_ltv[day_week_ltv['Week']==col]
    _=sns.lineplot(x=data['day_of_week'],y=data['day_week_wise_average_ltv'],color=clr)
    _=plt.title(str(data['year'].unique()[0])+' week: '+str(col)+": LTV Over the Day of the W
    _=plt.xlabel("")
    _=plt.ylabel("")
    _#=plt.xticks(fontsize=15)
    _=plt.tight_layout()
plt.show()
```

The above plot shows that in each week of 3 months, the average ltv of users shows a different pattern.

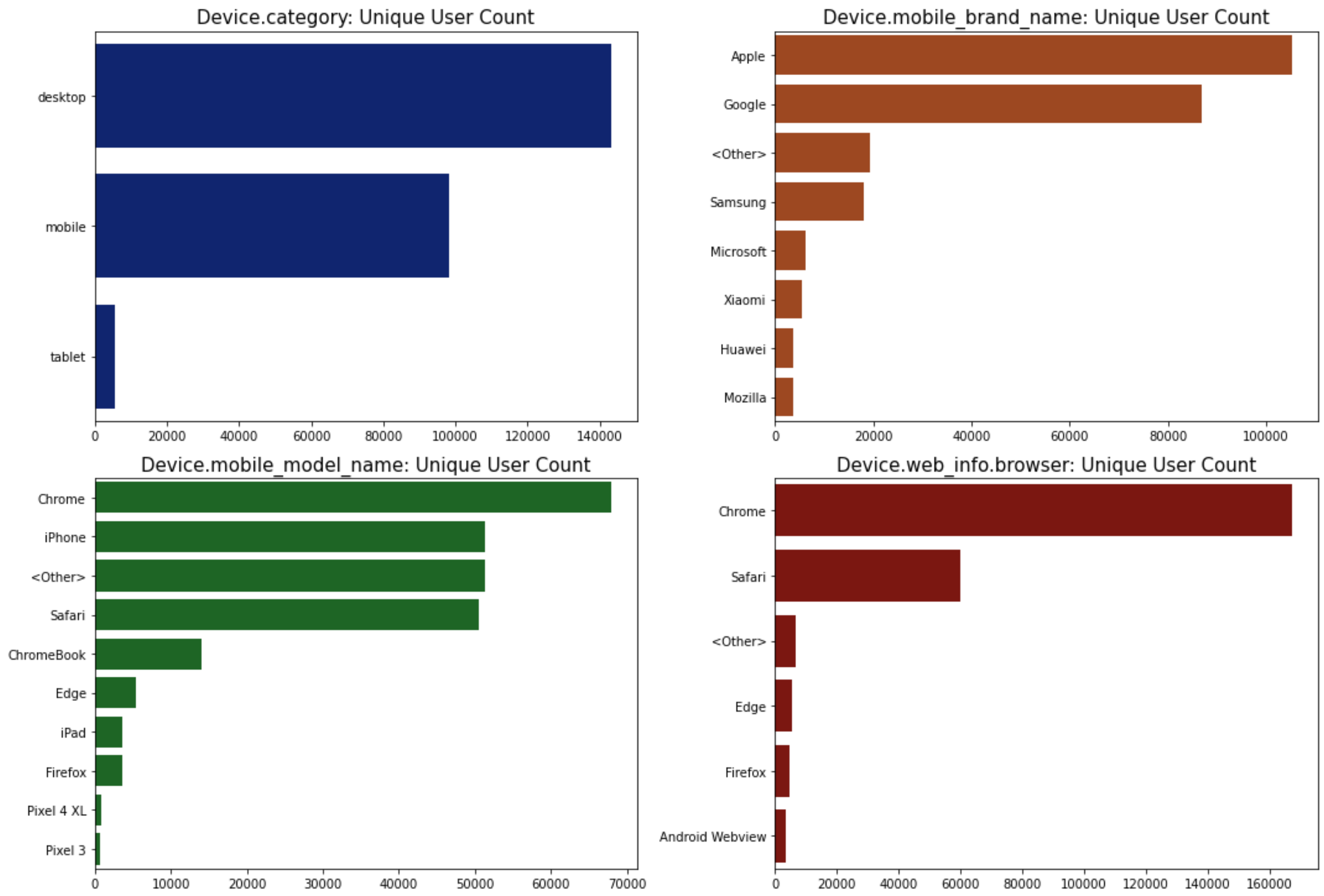
Let's explore the device column and its field.

In [ ]:
```
query=f"""
  select avg(event_value_in_usd) as avg_event_value ,avg(ltv) as average_ltv
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}` where event_value_in_usd is not null
  """
query_job=client.query(query)
val_ltv=query_job.to_dataframe()
```

In [ ]:
```
device_field_list=['device.category','device.mobile_brand_name','device.mobile_model_name',
                   'device.web_info.browser']
```

In [ ]:
```
fig=plt.subplots(figsize=(15, 20))
custom_palette = sns.color_palette("dark", 4)
colors=[col for col in custom_palette.as_hex()]
for i,(col,clr) in enumerate(zip(device_field_list,colors)):
    query=f"""
    select {col} ,count(distinct user_pseudo_id) as user_count
    FROM
    `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}`  group by {col} order by user_coun
    """
    query_job=client.query(query)
    df=query_job.to_dataframe()
    _=plt.subplot(4,2,i+1)
    _=sns.barplot(x=df['user_count'],y=df.iloc[:,0],color=clr)
    _=plt.title(col.capitalize()+": Unique User Count",fontsize=15)
    _=plt.xlabel("")
    _=plt.ylabel("")
    _#=plt.xticks(fontsize=15)
```

```
    _=plt.tight_layout()
plt.show()
```



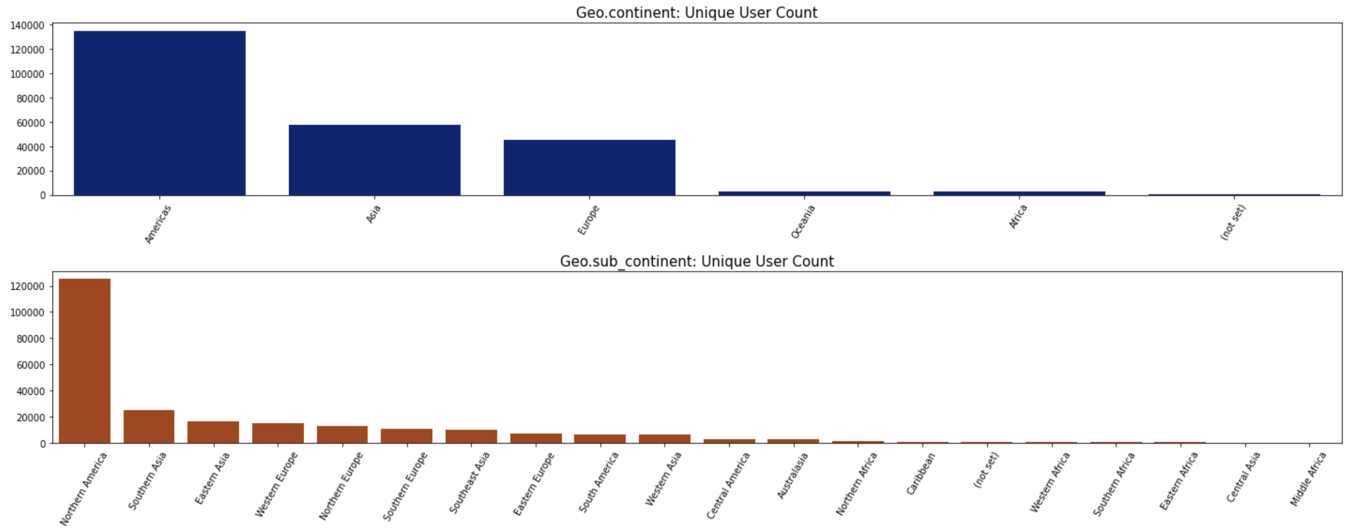
The above plot explains that the more users using the,

- desktop device
- apple, google brand mobile is most common among the users
- Chrome and iPhone are the most common mobile model name
- Chrome browser is the most common web browser among all devices.

Let's explore geo fields column and unique user by categories.

In [ ]:

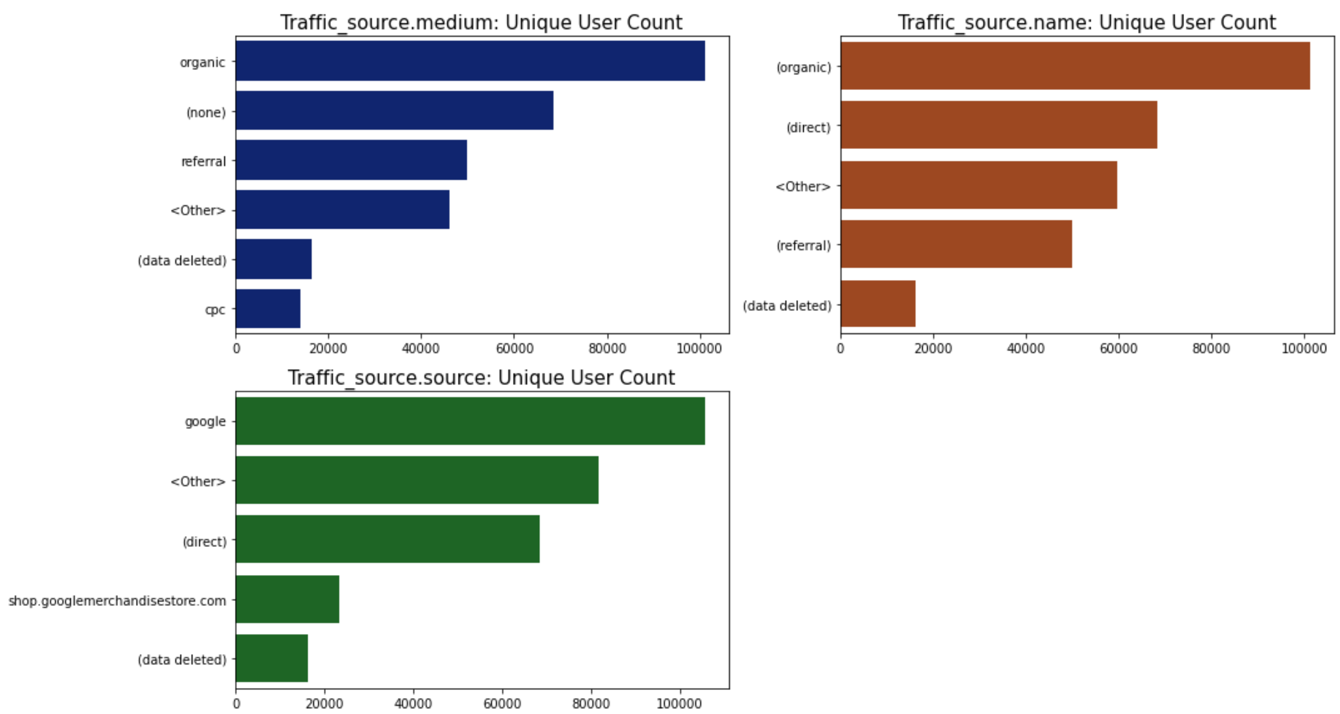
```
geo_field_list=['geo.continent','geo.sub_continent']
fig=plt.subplots(figsize=(20, 8))
custom_palette = sns.color_palette("dark", 2)
colors=[col for col in custom_palette.as_hex()]
for i,(col,clr) in enumerate(zip(geo_field_list,colors)):
    query=f"""
    select {col} ,count(distinct user_pseudo_id) as user_count
    FROM
    `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}`  group by {col} order by user_coun
    """
    query_job=client.query(query)
    df=query_job.to_dataframe()
    _=plt.subplot(2,1,i+1)
    _=sns.barplot(x=df.iloc[:,0],y=df['user_count'],color=clr)
    _=plt.title(col.capitalize()+": Unique User Count",fontsize=15)
    _=plt.xlabel("")
    _=plt.ylabel("")
    _=plt.xticks(rotation=60)
    _=plt.tight_layout()
plt.show()
```



The above plot explains that most of the users are from north America.

Let's explore traffic_source fields column and unique user by categories.

In [ ]:
```
traffic_source_field_list=['traffic_source.medium','traffic_source.name','traffic_source.sour
fig=plt.subplots(figsize=(15, 8))
custom_palette = sns.color_palette("dark", 3)
colors=[col for col in custom_palette.as_hex()]
for i,(col,clr) in enumerate(zip(traffic_source_field_list,colors)):
    query=f"""
    select {col} ,count(distinct user_pseudo_id) as user_count
    FROM
    `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}`  group by {col} order by user_coun
    """
    query_job=client.query(query)
    df=query_job.to_dataframe()
    _=plt.subplot(2,2,i+1)
    _=sns.barplot(x=df['user_count'],y=df.iloc[:,0],color=clr)
    _=plt.title(col.capitalize()+": Unique User Count",fontsize=15)
    _=plt.xlabel("")
    _=plt.ylabel("")
    _#=plt.xticks(fontsize=15)
    _=plt.tight_layout()
plt.show()
```



Let's see event_params nested field's page_title column and see how many unique user by page_title category.

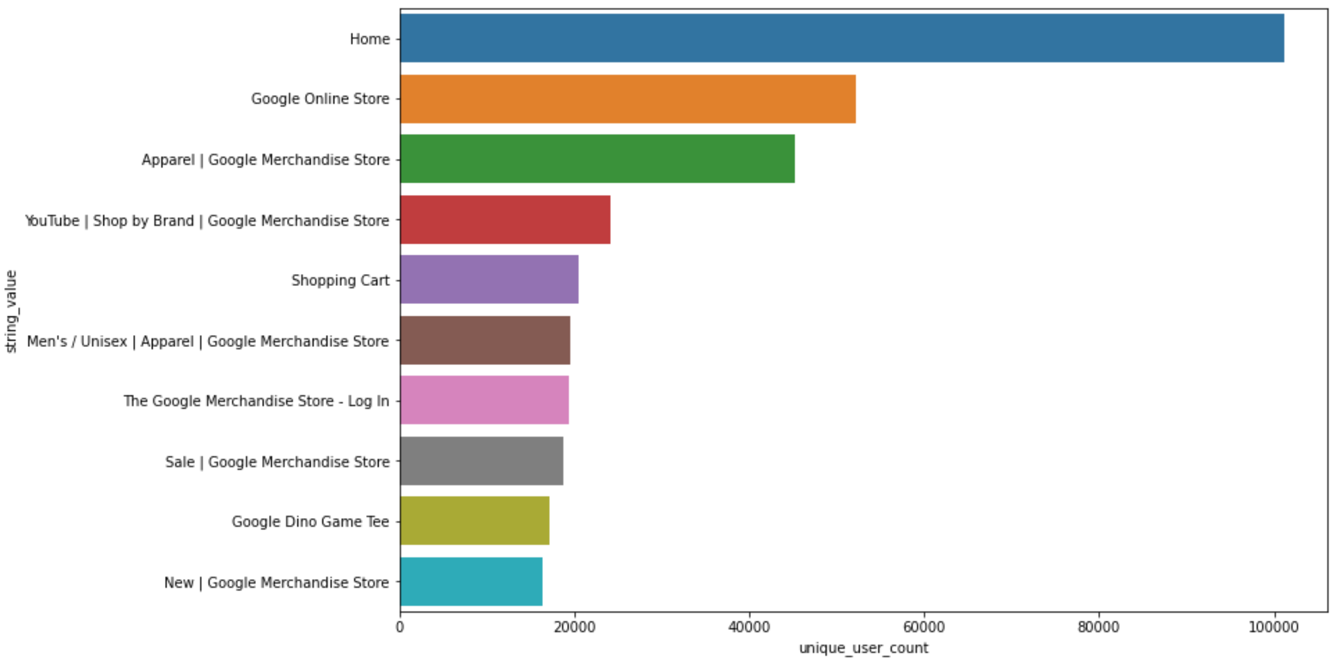
In [ ]:
```
query=f"""
  select ep.value.string_value, count(distinct user_pseudo_id) as unique_user_count
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}`,unnest(event_params) as ep
   where ep.key='page_title' group by 1 order by unique_user_count desc
  """
query_job=client.query(query)
event_page_unique_user=query_job.to_dataframe()
```

In [ ]:
```
event_page_unique_user.head()
```

Out[ ]:

	string_value	unique_user_count
0	Home	101108
1	Google Online Store	52243
2	Apparel | Google Merchandise Store	45237
3	YouTube | Shop by Brand | Google Merchandise S...	24049
4	Shopping Cart	20462

In [ ]:
```
_=plt.figure(figsize=(12,8))
_=sns.barplot(data=event_page_unique_user.nlargest(10,'unique_user_count') ,
              x='unique_user_count',y='string_value');
```



The above plot explains that the most of user activity was recorded from google home and google store.

Let's see the user's average engagement time distribution.

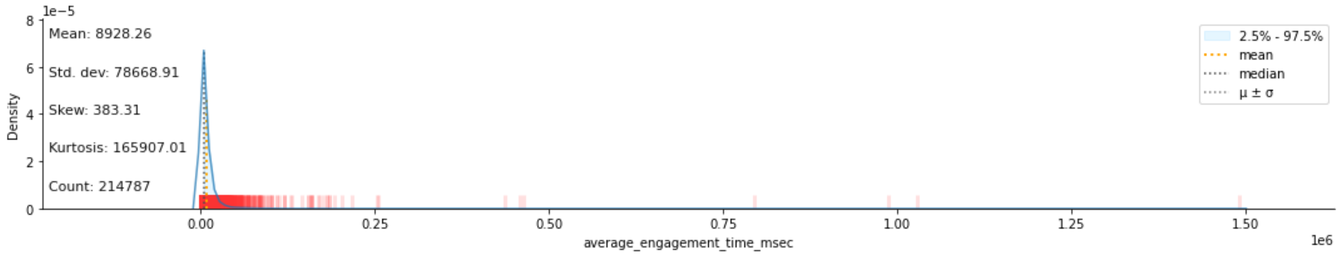
In [ ]:
```
query=f"""
  select user_pseudo_id as user_id, avg(ep.value.int_value) average_engagement_time_msec
  FROM
  `{PROJECT_ID_DATA}.{DATASET_ID_DATA}.{TABLE_ID_TRAIN}`,unnest(event_params) as ep
  where ep.key='engagement_time_msec' group by 1
  """
```

```
query_job=client.query(query)
user_avg_eng_time=query_job.to_dataframe()
```

In [ ]:
```
klib.dist_plot(user_avg_eng_time['average_engagement_time_msec'])
```

```
Large dataset detected, using 10000 random samples for the plots. Summary statistics are stil
l based on the entire dataset.
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f983a6a0cd0>
```



In [ ]:
```
user_avg_eng_time['average_engagement_time_msec'].describe()
```

Out[ ]:
```
count    2.147870e+05
mean     8.928263e+03
std      7.866891e+04
min      1.000000e+00
25%      2.361833e+03
50%      4.970333e+03
75%      9.934946e+03
max      3.418754e+07
Name: average_engagement_time_msec, dtype: float64
```

The above histogram plot and summary explain that the average engagement time is right-skewed (positively skewed).