Library

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing and Reading the Dataset

```
df=pd.read_csv("/content/website.csv")
```

```
df.head()
```

	# -----------	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
0	Session primary channel group (Default	Date + hour (YYYYMMDDHH)	Users	Sessions	Engaged sessions	Average engagement time per session	Engaged sessions per user	Events per session	Engagement rate

Next steps: Generate code with df | View recommended plots | New interactive sheet

Proper Headers

```
df.columns=df.iloc[0]
print("********** Columns After Promoting The Headers*****************")

df.head()
```

```
********** Columns After Promoting The Headers*****************
```

	Session primary channel group (Default channel group)	Date + hour (YYYYMMDDHH)	Users	Sessions	Engaged sessions	Average engagement time per session	Engaged sessions per user	Events per session	Engagement rate	E c
0	Session primary channel group	Date + hour (YYYYMMDDHH)	Users	Sessions	Engaged sessions	Average engagement time per session	Engaged sessions per user	Events per session	Engagement rate	E c

Next steps: Generate code with df | View recommended plots | New interactive sheet

Now Droping The 0 index

```
df=df.drop(index=0)
df.head()
```

	Session primary channel group (Default channel group)	Date + hour (YYYYMMDDHH)	Users	Sessions	Engaged sessions	Average engagement time per session	Engaged sessions per user	Events per session	Engagement rate	Eve cou
1	Direct	2024041623	237	300	144	47.526666666666700	0.6075949367088610	4.673333333333330	0.48	14
2	Organic	2024041719	208	267	132	32.09737827715360	0.6346153846153850	4.295880149812730	0.4943820224719100	11

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.reset_index(drop=True,inplace=True)
```

```
df.head()
```

	Session primary channel group (Default channel group)	Date + hour (YYYYMMDDHH)	Users	Sessions	Engaged sessions	Average engagement time per session	Engaged sessions per user	Events per session	Engagement rate	Eve cou
0	Direct	2024041623	237	300	144	47.526666666666700	0.6075949367088610	4.673333333333330	0.48	14
1	Organic	2024041719	208	267	132	32.09737827715360	0.6346153846153850	4.295880149812730	0.4943820224719100	11

Re-naming the columns

```
df.columns=['channel group','DateHour','Users','Sessions','Engaged Sessions','Avg eng time per session','Engaged sessions per user',    'Eve
```

```
df.head()
```

	channel group	DateHour	Users	Sessions	Engaged Sessions	Avg eng time per session	Engaged sessions per user	Events per session	Engagement rate	Event coun
0	Direct	2024041623	237	300	144	47.526666666666700	0.6075949367088610	4.673333333333330	0.48	1402
1	Organic Social	2024041719	208	267	132	32.09737827715360	0.6346153846153850	4.295880149812730	0.4943820224719100	1147
2	Direct	2024041723	188	233	115	39.93991416309010	0.6117021276595740	4.587982832618030	0.49356223175965700	1069
	Organic									

Checking Datatype of Columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3182 entries, 0 to 3181
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   channel group              3182 non-null   object
 1   DateHour                   3182 non-null   object
 2   Users                      3182 non-null   object
 3   Sessions                   3182 non-null   object
 4   Engaged Sessions           3182 non-null   object
 5   Avg eng time per session   3182 non-null   object
 6   Engaged sessions per user  3182 non-null   object
 7   Events per session         3182 non-null   object
 8   Engagement rate            3182 non-null   object
 9   Event coun                 3182 non-null   object
dtypes: object(10)
memory usage: 248.7+ KB
```

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

```
'2024042209', '2024041502', '2024041602', '2024042202',
'2024042308', '2024050101', '2024050208', '2024050302',
'2024050207', '2024041508', '2024040909', '2024041309',
'2024041802', '2024041902', '2024041406', '2024042608',
'2024040610', '2024041008', '2024041601', '2024041702',
'2024041706', '2024041808', '2024043008', '2024040707',
'2024041206', '2024042208', '2024050102', '2024050201',
'2024050307', '2024041204', '2024042100', '2024042609',
'2024040602', '2024041007', '2024041102', '2024041207',
'2024041907', '2024042301', '2024042407', '2024043007',
'2024050306', '2024040702', '2024041107', '2024041402',
'2024042401', '2024040801', '2024040808', '2024041404',
'2024043002', '2024050107', '2024050206', '2024040902',
'2024041301', '2024041506', '2024041607', '2024042501',
'2024050305', '2024041202', '2024041205', '2024041308',
'2024042300', '2024042507', '2024042601', '2024042807',
'2024050108', '2024042901', '2024040807', '2024040903',
'2024040907', '2024040908', '2024041807', '2024042402',
'2024042508', '2024042709', '2024040703', '2024041005',
'2024042002', '2024050202', '2024050304', '2024040704',
'2024041507', '2024041603', '2024042207', '2024042302',
'2024040603', '2024041108', '2024041203', '2024042307',
'2024042802', '2024042603', '2024041504', '2024042404',
'2024042502', '2024042905', '2024050103', '2024041105',
'2024041606', '2024041006', '2024041704', '2024040803',
'2024042007', '2024042306', '2024042702', '2024041803',
'2024040802', '2024041003', '2024041806', '2024042004',
'2024042101', '2024042406', '2024042607', '2024042703',
'2024041903', '2024041904', '2024040604', '2024040608',
'2024041302', '2024042104', '2024042107', '2024042109',
'2024042206', '2024042602', '2024042907', '2024043003',
'2024050205', '2024042405', '2024041104', '2024041503',
'2024041906', '2024042003', '2024042008', '2024042903',
'2024040601', '2024041306', '2024041605', '2024040607',
'2024041307', '2024041505', '2024042106', '2024042203',
'2024042303', '2024042403', '2024042902', '2024043005',
'2024043006', '2024050104', '2024041106', '2024040906',
'2024041804', '2024042504', '2024042506', '2024043004',
'2024050204', '2024042108', '2024042906', '2024040706',
'2024041705', '2024041805', '2024042707', '2024042804',
'2024050203', '2024041405', '2024042103', '2024042204',
'2024042605', '2024042806', '2024040606', '2024041103',
'2024042706', '2024042803', '2024050106', '2024041004',
'2024041604', '2024042102', '2024042205', '2024042505',
```

```
df['DateHour'] = pd.to_datetime(df['DateHour'], format='%Y%m%d%H', errors='coerce')
```

```
df.head()
```

	channel group	DateHour	Users	Sessions	Engaged Sessions	Avg eng time per session	Engaged sessions per user	Events per session	Engagement rate	Event coun
0	Direct	2024-04-16 23:00:00	237	300	144	47.526666666666700	0.6075949367088610	4.673333333333330	0.48	1402
1	Organic Social	2024-04-17 19:00:00	208	267	132	32.09737827715360	0.6346153846153850	4.295880149812730	0.4943820224719100	1147

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
num_col=['Users','Sessions','Engaged Sessions','Avg eng time per session','Engaged sessions per user','Events per session','Engagement rate'
df[num_col]=df[num_col].apply(pd.to_numeric,errors='coerce')
df['hour']=df['DateHour'].dt.hour
```

```
df.head()
```

	channel group	DateHour	Users	Sessions	Engaged Sessions	Avg eng time per session	Engaged sessions per user	Events per session	Engagement rate	Event coun	hour
0	Direct	2024-04-16 23:00:00	237	300	144	47.526667	0.607595	4.673333	0.480000	1402	23
1	Organic Social	2024-04-17 19:00:00	208	267	132	32.097378	0.634615	4.295880	0.494382	1147	19

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3182 entries, 0 to 3181
Data columns (total 11 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   channel group             3182 non-null   object
 1   DateHour                  3182 non-null   datetime64[ns]
 2   Users                     3182 non-null   int64
 3   Sessions                  3182 non-null   int64
 4   Engaged Sessions          3182 non-null   int64
 5   Avg eng time per session  3182 non-null   float64
 6   Engaged sessions per user 3182 non-null   float64
 7   Events per session        3182 non-null   float64
 8   Engagement rate           3182 non-null   float64
 9   Event coun                3182 non-null   int64
 10  hour                      3182 non-null   int32
dtypes: datetime64[ns](1), float64(4), int32(1), int64(4), object(1)
memory usage: 261.2+ KB
```

Basic Descriptive Statistics

```
df.describe()
```

	DateHour	Users	Sessions	Engaged Sessions	Avg eng time per session	Engaged sessions per user	Events per session	Engagement rate	Event coun	hou
count	3182	3182.000000	3182.000000	3182.000000	3182.000000	3182.000000	3182.000000	3182.000000	3182.000000	3182.00000
mean	2024-04-20 01:17:07.278441216	41.935889	51.192646	28.325581	66.644581	0.606450	4.675969	0.503396	242.272470	11.80704
min	2024-04-06 00:00:00	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000	0.00000
25%	2024-04-13 02:15:00	20.000000	24.000000	13.000000	32.103034	0.561404	3.750000	0.442902	103.000000	6.00000
50%	2024-04-20 02:00:00	42.000000	51.000000	27.000000	49.020202	0.666667	4.410256	0.545455	226.000000	12.00000
75%	2024-04-26 22:00:00	60.000000	71.000000	41.000000	71.487069	0.750000	5.217690	0.633333	339.000000	18.00000

Session and user over Time:-

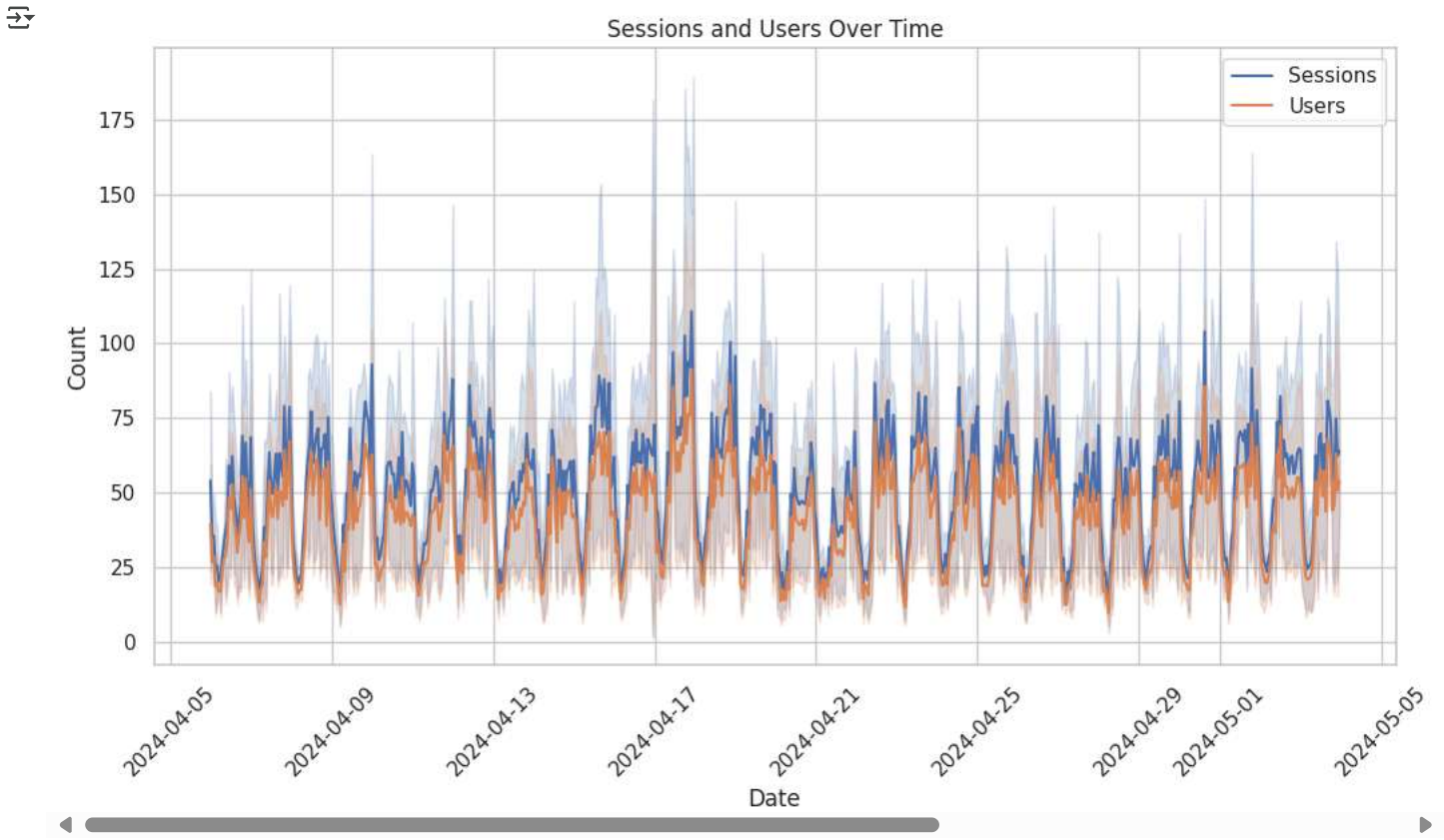
1. What patterns or trends can you observe in website sessions and users over time?

```
time=df.groupby('DateHour').agg({'Sessions':'sum','Users':'sum'})
time.head()
```

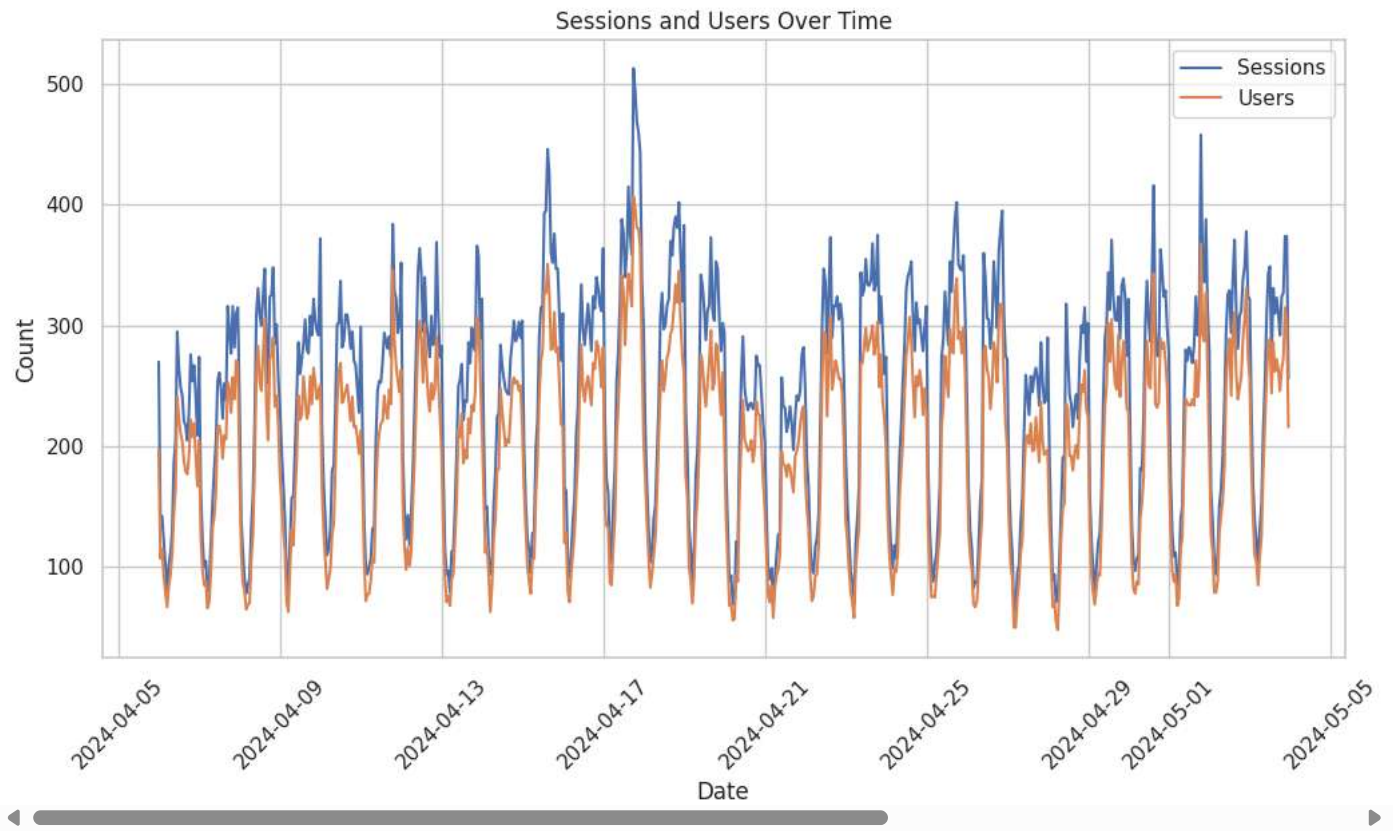
DateHour	Sessions	Users
2024-04-06 00:00:00	270	197
2024-04-06 01:00:00	142	107
2024-04-06 02:00:00	142	115
2024-04-06 03:00:00	122	93
2024-04-06 04:00:00	102	79

Next steps: Generate code with time | View recommended plots | New interactive sheet

```
sns.set(style='whitegrid')
plt.figure(figsize=(12,6))
sns.lineplot(data=df,x='DateHour',y='Sessions',label='Sessions')
sns.lineplot(data=df,x='DateHour',y='Users',label='Users')
plt.title('Sessions and Users Over Time')
plt.xlabel('Date')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```




```
sns.set(style='whitegrid')
plt.figure(figsize=(12,6))
sns.lineplot(data=time,x=time.index,y='Sessions',label='Sessions')
sns.lineplot(data=time,x=time.index,y='Users',label='Users')
plt.title('Sessions and Users Over Time')
plt.xlabel('Date')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



Observation:-shows fluctuations in both sessions and users over the recorded dates. There appear to be daily patterns, with peaks and troughs. This suggests that website traffic varies significantly throughout the day or across different days Certain days (like around April 17 to 21) show notable spikes, possibly due to campaigns, promotions, or external events..

2.Which marketing channel brought the highest number of users to the website, and how can we use this insight to improve traffic from other sources?

```
df.head()
```

	channel group	DateHour	Users	Sessions	Engaged Sessions	Avg eng time per session	Engaged sessions per user	Events per session	Engagement rate	Event coun	hour
0	Direct	2024-04-16 23:00:00	237	300	144	47.526667	0.607595	4.673333	0.480000	1402	23
1	Organic Social	2024-04-17 19:00:00	208	267	132	32.097378	0.634615	4.295880	0.494382	1147	19

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
channel=df.groupby("channel group")['Users'].sum().sort_values(ascending=False)
channel
```

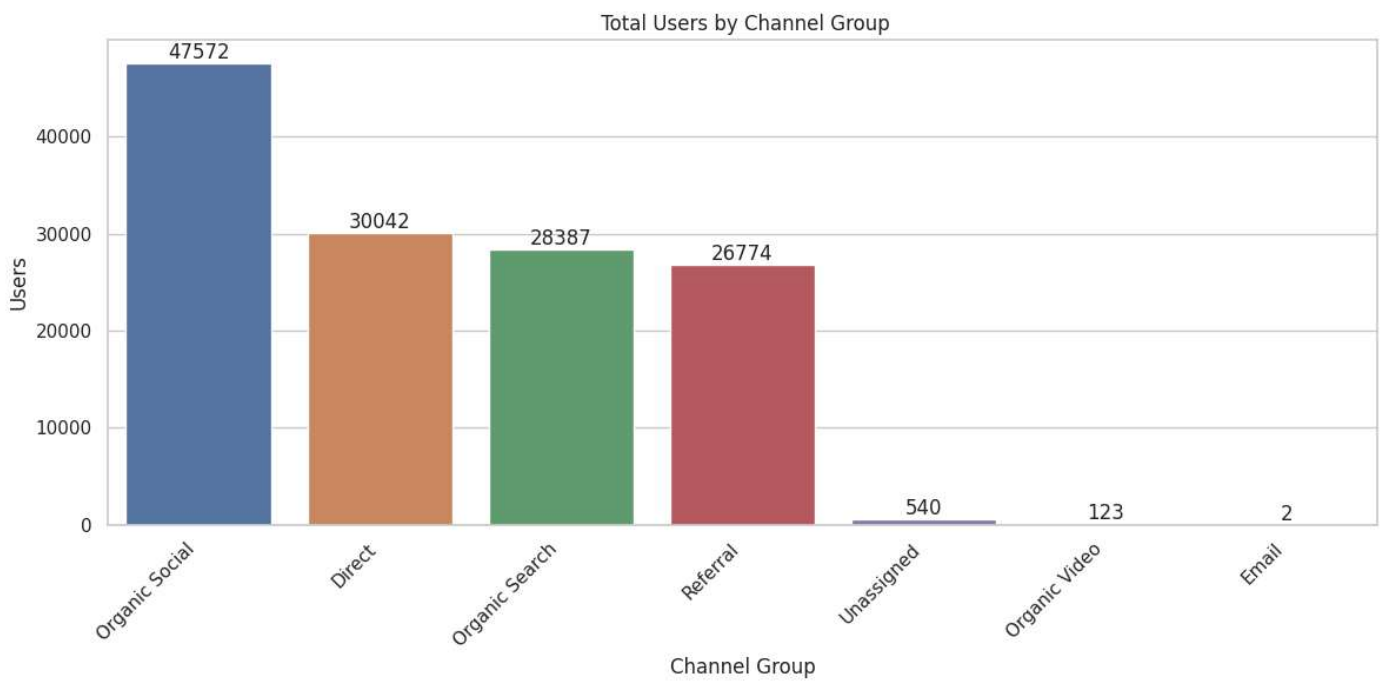
channel group	Users
Organic Social	47572
Direct	30042
Organic Search	28387
Referral	26774
Unassigned	540
Organic Video	123
Email	2

dtype: int64

```
plt.figure(figsize=(12,6))
a = sns.barplot(x=channel.index, y=channel.values,hue=channel.index)
plt.title(' Total Users by Channel Group')
plt.xlabel('Channel Group')
plt.ylabel('Users')
plt.xticks(rotation=45, ha='right')

# Add count labels on top of the bars
for container in a.containers:
    a.bar_label(container, fmt='%.0f')

plt.tight_layout()
plt.show()
```



Most User Come through Channel Like Organic Social & Direct Channel Marketing Team Needs to work on channels like Unassigned, Organic Video and Email

Avg Engagment time by Channel

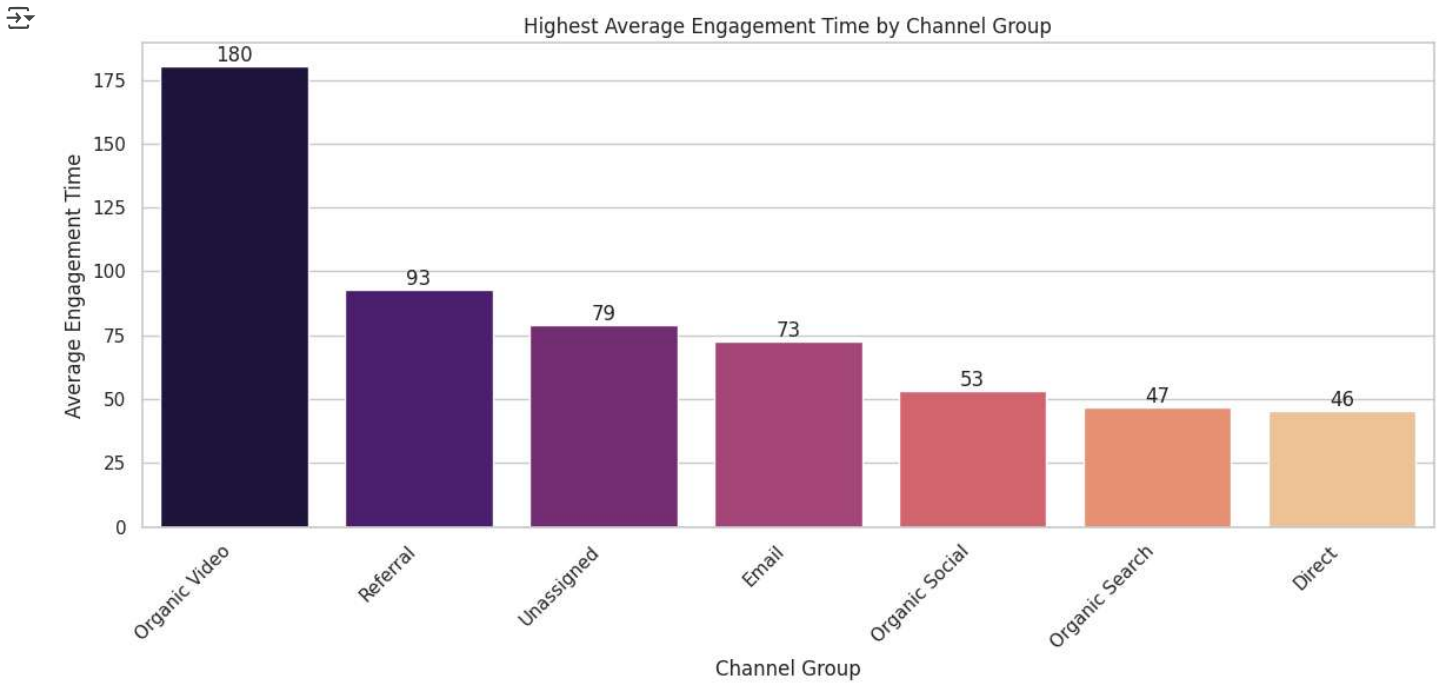
3.Which channel has the highest average engagement time, and what does that tell us about user behavior and content effectiveness?

```
highest_average_engagement_time=df.groupby('channel group')['Avg eng time per session'].mean().sort_values(ascending=False)
```

```
plt.figure(figsize=(12,6))
a = sns.barplot(x=highest_average_engagement_time.index, y=highest_average_engagement_time.values,hue=highest_average_engagement_time.index,
plt.title('Highest Average Engagement Time by Channel Group')
plt.xlabel('Channel Group')
plt.ylabel('Average Engagement Time')
plt.xticks(rotation=45, ha='right')

# Add count labels on top of the bars
for container in a.containers:
    a.bar_label(container, fmt='%.0f')

plt.tight_layout()
plt.show()
```



So Highest User Engagment time is for organic video and Referal are high .so need to spend more on this in order to aquire more user to website

Engagment Rate Distribution Across various Channels

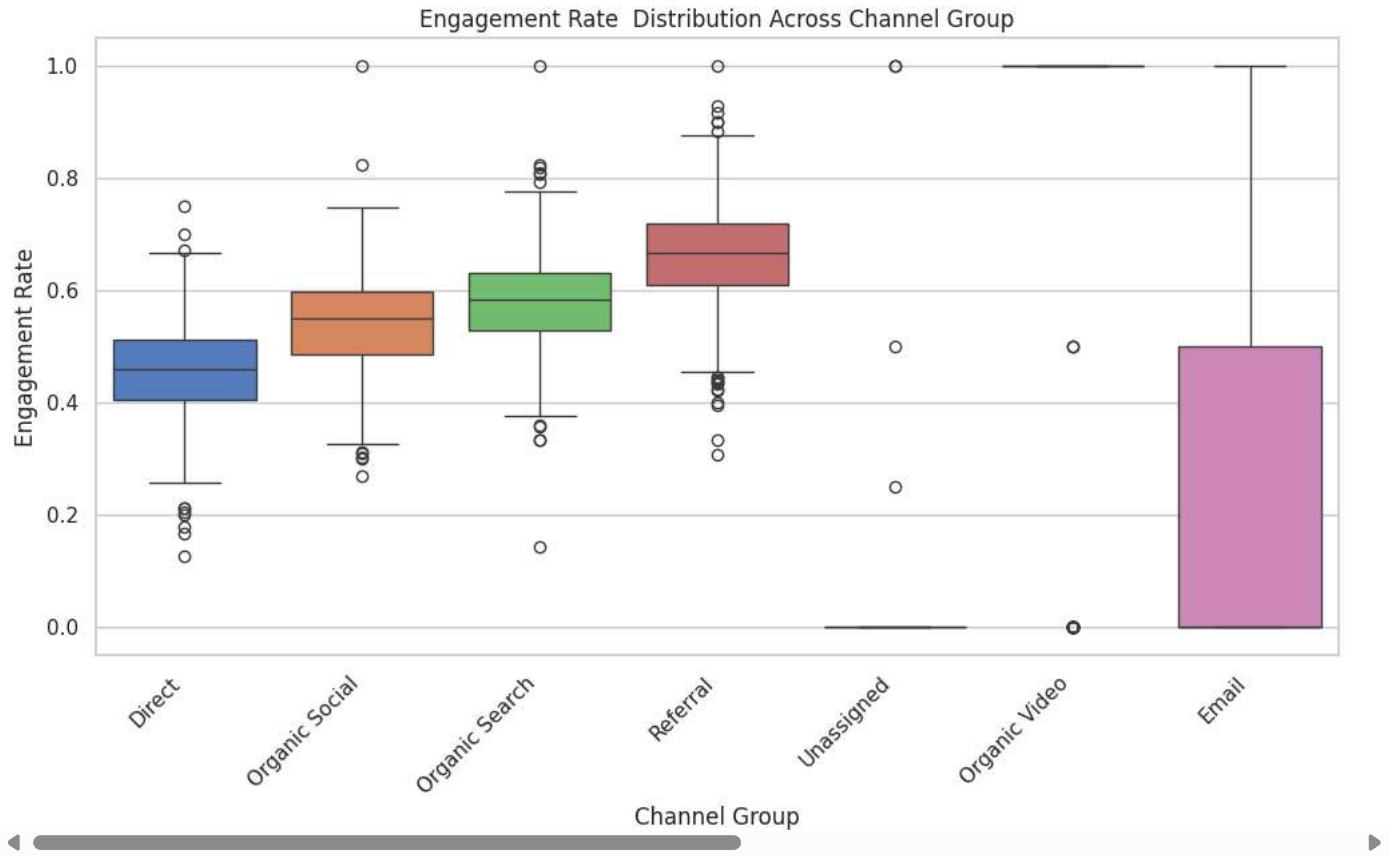
4.How does engagement rate vary across different traffic channels?

```
plt.figure(figsize=(12,6))
sns.boxplot(x='channel group',y='Engagement rate',data=df,palette='muted')
plt.title('Engagement Rate  Distribution Across Channel Group')
plt.xlabel('Channel Group')
plt.ylabel('Engagement Rate')
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
/tmp/ipython-input-65-1428263602.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legenc

  sns.boxplot(x='channel group',y='Engagement rate',data=df,palette='muted')
```



Referral, Organic Search, and Organic Social channels show higher and more consistent engagement rates. Email, Unassigned, and Organic Video channels have lower or highly variable engagement, indicating areas for improvement.

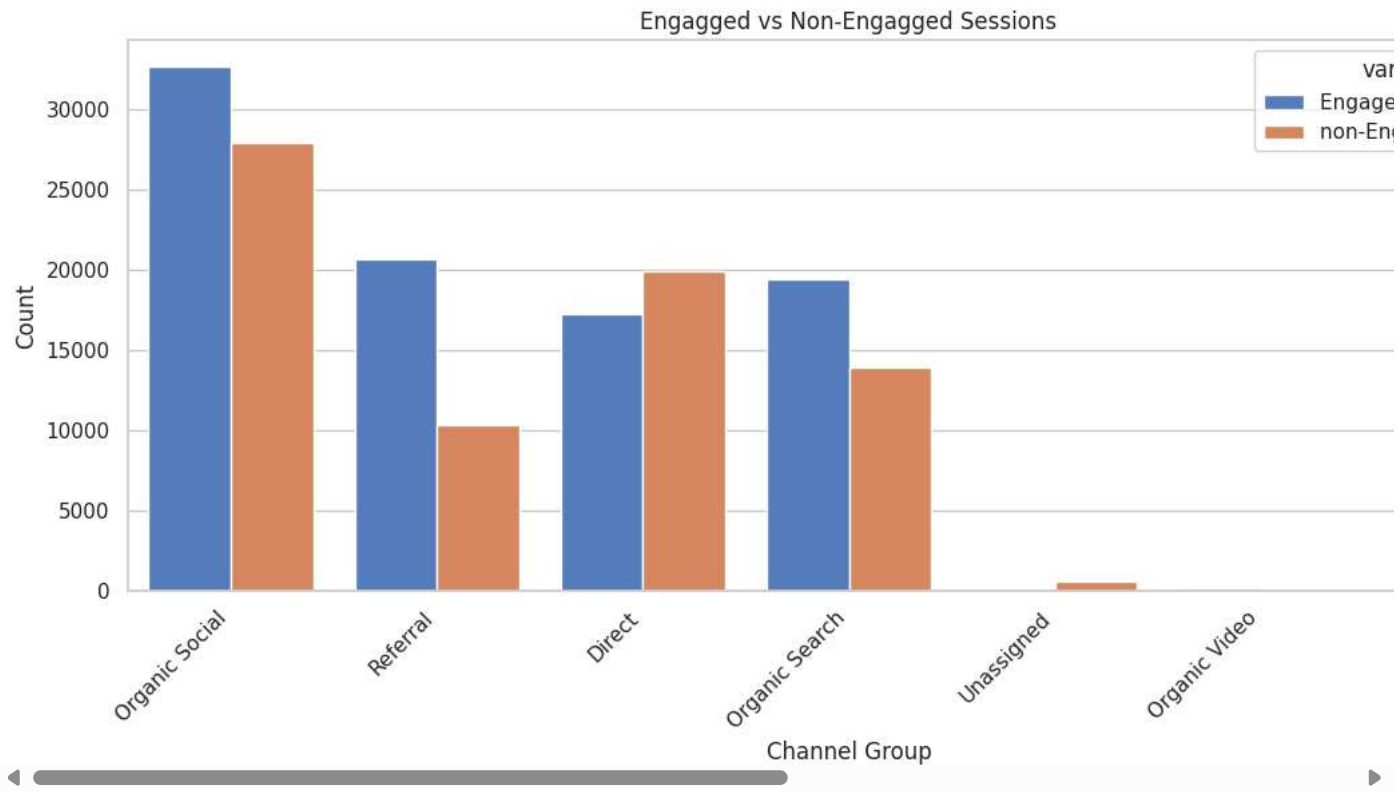
Engagged vs Non-Engagged Sessions

Which channels are driving more engaged sessions compared to non-engaged ones, and what strategies can improve engagement in underperforming channels?

```
session_df=df.groupby('channel group').agg({'Sessions':'sum','Engaged Sessions':'sum'}).reset_index()
session_df['non-Engaged Sessions']=session_df['Sessions']-session_df['Engaged Sessions']
session_df['Engagement Rate']=session_df['Engaged Sessions']/session_df['Sessions']*100
session_df_melted=pd.melt(session_df,id_vars='channel group',value_vars=['Engaged Sessions','non-Engaged Sessions']).sort_values(by='value',
session_df
```

```
plt.figure(figsize=(12,6))
sns.barplot(x='channel group',y='value',hue='variable',data=session_df_melted,palette='muted')
plt.title('Engagged vs Non-Engagged Sessions')
plt.xlabel('Channel Group')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')

plt.tight_layout()
plt.show()
```



Referral, Organic Search, and Organic Social channels are driving more engaged sessions, as indicated by their higher median engagement rates and consistent performance. To improve underperforming channels like Email, Unassigned, and Organic Video, consider personalized content, better targeting, reclassification of traffic sources, and A/B testing campaign strategies.

Traffic by Hour and Channel

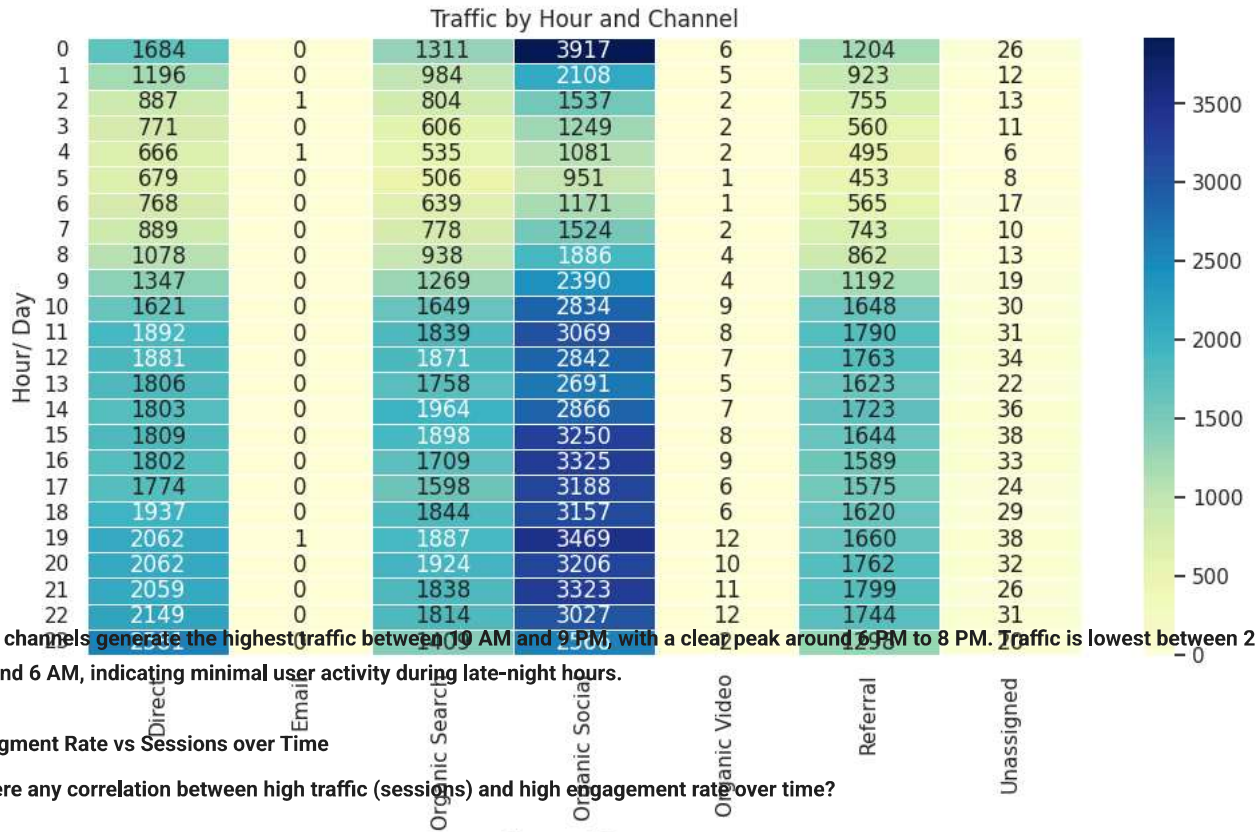
At what hours of the day does each channel drive the most traffic?

```
heat_map=df.groupby(["hour","channel group"])["Sessions"].sum().unstack().fillna(0)
heat_map.head(3)
```

channel group	Direct	Email	Organic Search	Organic Social	Organic Video	Referral	Unassigned
hour							
0	1684.0	0.0	1311.0	3917.0	6.0	1204.0	26.0
1	1196.0	0.0	984.0	2108.0	5.0	923.0	12.0
2	887.0	1.0	804.0	1537.0	2.0	755.0	13.0

Next steps: Generate code with heat_map | View recommended plots | New interactive sheet

```
plt.figure(figsize=(12,6))
sns.heatmap(heat_map,cmap='YlGnBu',linewidths=.5,annot=True,fmt='.0f')
plt.title('Traffic by Hour and Channel')
plt.xlabel('Channel Group')
plt.ylabel('Hour/ Day')
plt.show()
```



Most channels generate the highest traffic between 10 AM and 9 PM, with a clear peak around 6 PM to 8 PM. Traffic is lowest between 2 AM and 6 AM, indicating minimal user activity during late-night hours.

Engagment Rate vs Sessions over Time

Is there any correlation between high traffic (sessions) and high engagement rate over time?

```
df
```

	channel group	DateHour	Users	Sessions	Engaged Sessions	Avg eng time per session	Engaged sessions per user	Events per session	Engagement rate	Event coun	hour
0	Direct	2024-04-16 23:00:00	237	300	144	47.526667	0.607595	4.673333	0.480000	1402	23
1	Organic Social	2024-04-17 19:00:00	208	267	132	32.097378	0.634615	4.295880	0.494382	1147	19
2	Direct	2024-04-17 23:00:00	188	233	115	39.939914	0.611702	4.587983	0.493562	1069	23
3	Organic Social	2024-04-17 18:00:00	187	256	125	32.160156	0.668449	4.078125	0.488281	1044	18
4	Organic Social	2024-04-17 20:00:00	175	221	112	46.918552	0.640000	4.529412	0.506787	1001	20
...	...	...	...	...	...	...	...	...	...	...	...
3177	Unassigned	2024-04-28 06:00:00	0	1	0	0.000000	0.000000	2.000000	0.000000	2	6

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df_plot=df.groupby('DateHour')[['Sessions','Engagement rate']].mean().reset_index()

plt.figure(figsize=(12,6))
sns.lineplot(data=df_plot,x='DateHour',y='Sessions',label='Sessions')
sns.lineplot(data=df_plot,x='DateHour',y='Engagement rate',label='Engagement rate')
plt.title('Engagement Rate vs Sessions over Time')
plt.xlabel('Date')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend()
```