AGENDA: Help Yulu find the variables which are significant to predict the demand of Users.

In [1]:
```
import pandas as pd, numpy as np, seaborn as sns, matplotlib.pyplot as plt
%matplotlib inline
```

Column Profiling:

datetime: datetime season: season (1: spring, 2: summer, 3: fall, 4: winter) holiday: whether day is a holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule) workingday: if day is neither weekend nor holiday is 1, otherwise is 0. weather: 1: Clear, Few clouds, partly cloudy, partly cloudy 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog temp: temperature in Celsius atemp: feeling temperature in Celsius humidity: humidity windspeed: wind speed casual: count of casual users registered: count of registered users count: count of total rental bikes including both casual and registered

In [2]:
```
df = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000
```

EDA

In [3]:
```
df.head()
```

Out[3]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casu
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  int64
 2   holiday     10886 non-null  int64
 3   workingday  10886 non-null  int64
 4   weather     10886 non-null  int64
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.7 KB
```

In [5]:
```
df['day'] = df.datetime.dt.day
df['day_name'] = df.datetime.dt.day_name()
df['month'] =df.datetime.dt.month_name()
df['year'] = df.datetime.dt.year
df['hour'] = df.datetime.dt.hour
df['date'] = df.datetime.dt.date
```

In [36]:
```
df.head(3)
```

Out[36]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casu
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	

In [6]:
```
print('Column_Name -- > Unique Values')
for i in df.columns:
    print(f'{i} -- > {df[i].nunique()}')
```

```
Column_Name -- > Unique Values
datetime -- > 10886
season -- > 4
holiday -- > 2
workingday -- > 2
weather -- > 4
temp -- > 49
atemp -- > 60
humidity -- > 89
windspeed -- > 28
casual -- > 309
registered -- > 731
count -- > 822
day -- > 19
day_name -- > 7
month -- > 12
year -- > 2
hour -- > 24
date -- > 456
```

In [16]:
```python
numerical_columns = ['temp', 'atemp', 'humidity', 'windspeed']
df[numerical_columns].describe()
```

Out[16]:

	temp	atemp	humidity	windspeed
count	10886.00000	10886.000000	10886.000000	10886.000000
mean	20.23086	23.655084	61.886460	12.799395
std	7.79159	8.474601	19.245033	8.164537
min	0.82000	0.760000	0.000000	0.000000
25%	13.94000	16.665000	47.000000	7.001500
50%	20.50000	24.240000	62.000000	12.998000
75%	26.24000	31.060000	77.000000	16.997900
max	41.00000	45.455000	100.000000	56.996900

In [63]:
```python
categorical_columns = ['season', 'holiday', 'workingday', 'weather']
for i in categorical_columns:
    print(i)
    print(df[i].unique())
    print()
```

```
season
[1 2 3 4]

holiday
[0 1]

workingday
[0 1]

weather
[1 2 3 4]
```

In [64]:
```python
date_variables = ['year', 'month', 'day_name', 'day', 'hour']
for column in date_variables:
```

```
    print(column)
    print(df[column].unique())
    print()
```

```
year
[2011 2012]

month
['January' 'February' 'March' 'April' 'May' 'June' 'July' 'August'
 'September' 'October' 'November' 'December']

day_name
['Saturday' 'Sunday' 'Monday' 'Tuesday' 'Wednesday' 'Thursday' 'Friday']

day
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]

hour
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

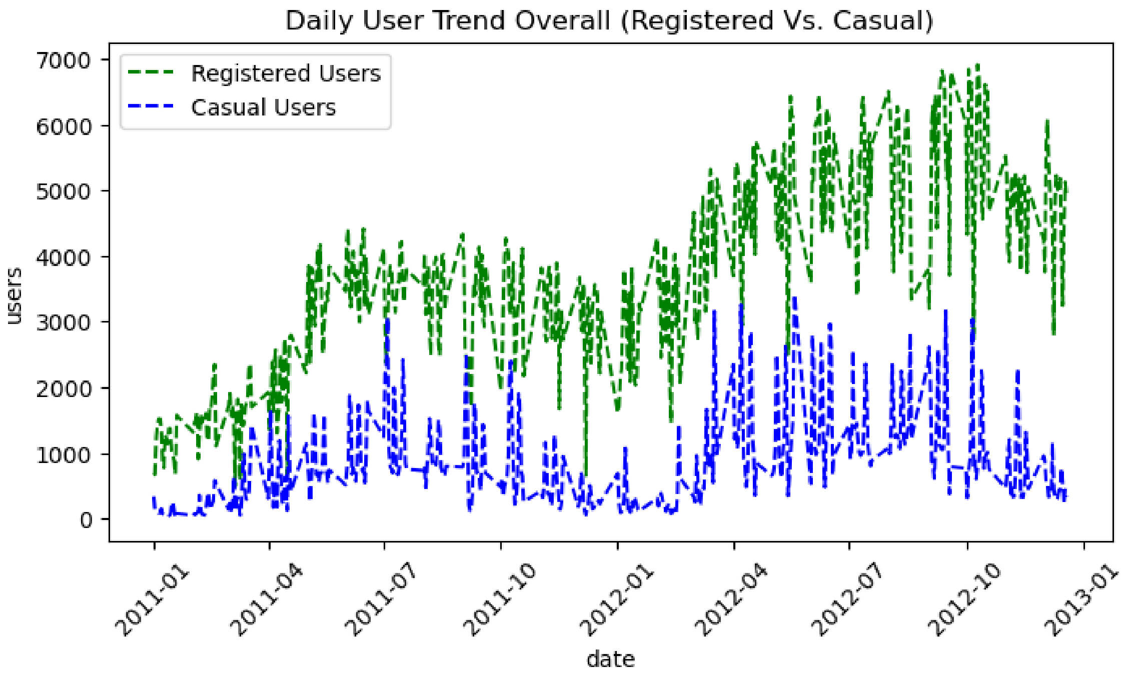
Bivariate Analysis

In [57]:
```
df.groupby("year").agg(casual_users = ('casual', 'sum'), registered_users= ('reg
```

Out[57]:

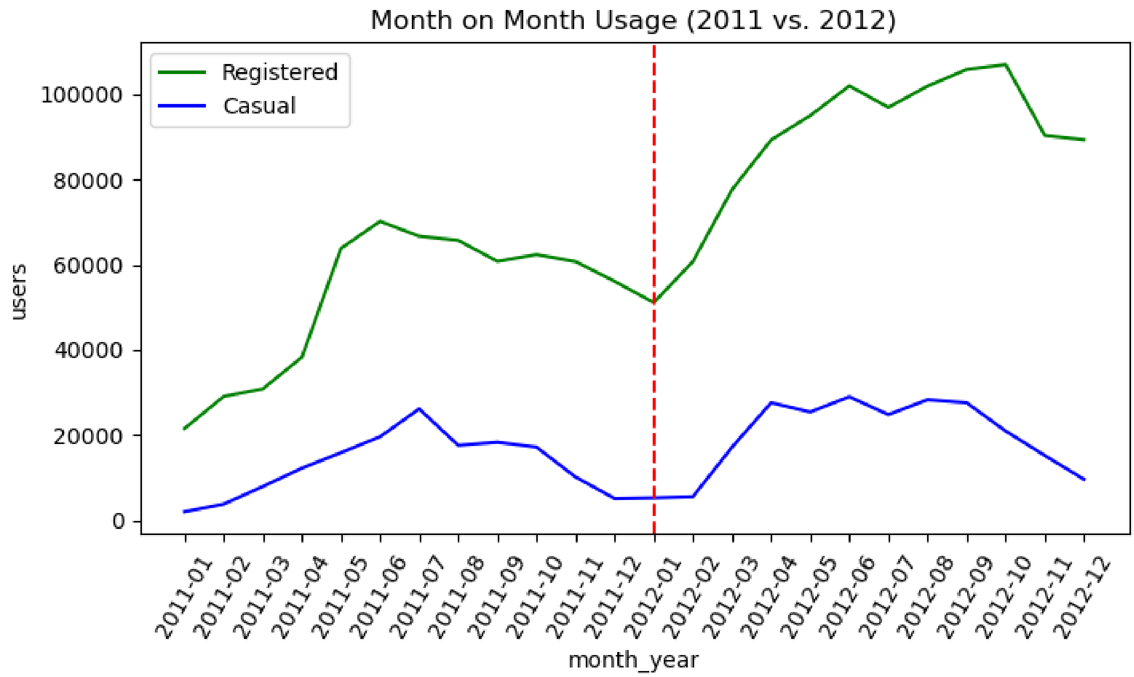
year	casual_users	registered_users	total_users
2011	155817	626162	781979
2012	236318	1067179	1303497

In [61]:
```
plt.figure(figsize = (8,4))
grouped1 = df.groupby('date').agg(users=('registered', 'sum')).reset_index()
grouped2 = df.groupby('date').agg(users=('casual', 'sum')).reset_index()
sns.lineplot(x='date', y='users', linestyle='--', data=grouped1, color='green',
sns.lineplot(x='date', y='users', linestyle='--', data=grouped2, color='blue', l
plt.legend()
plt.xticks(rotation = 45);
plt.title('Daily User Trend Overall (Registered Vs. Casual)');
```



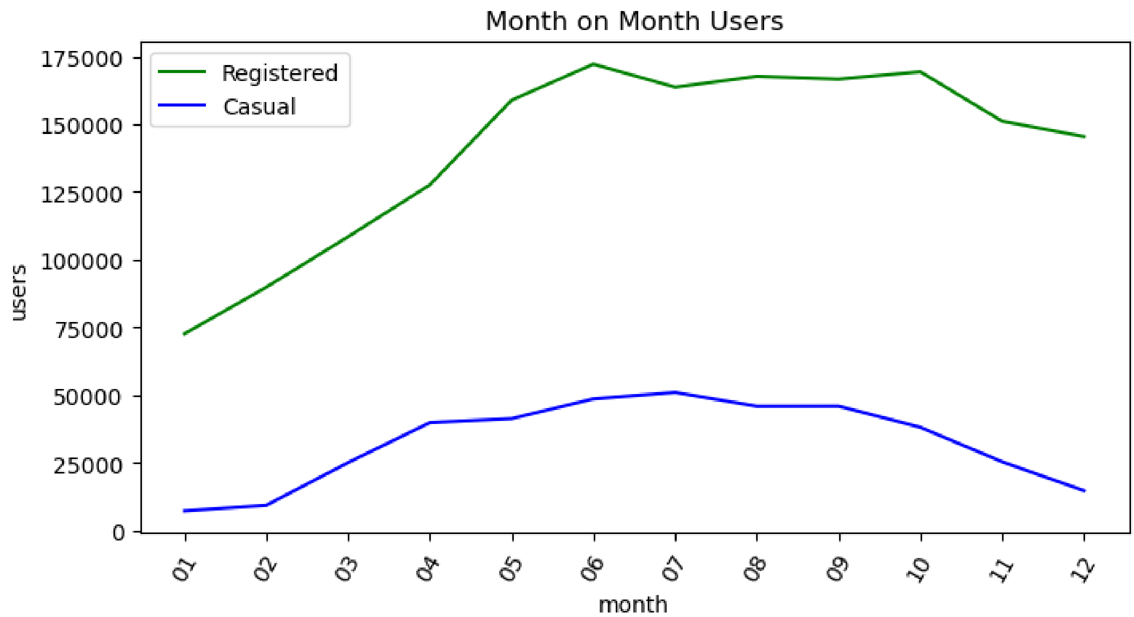
Observation - There has been an Increasing Trend in the number of users who have registered. The Casual Users seems to have been on the same level though.

In [54]:
```
plt.figure(figsize = (8,4))
new_df1 = df[['datetime','registered']]
new_df2 = df[['datetime','casual']]
with pd.option_context('mode.chained_assignment', None):
    new_df1.loc[:,'month_year']= new_df['datetime'].dt.strftime('%Y-%m')
    new_df2.loc[:,'month_year']= new_df['datetime'].dt.strftime('%Y-%m')
grouped1 = new_df1.groupby('month_year').agg(users = ('registered', 'sum')).rese
grouped2 = new_df2.groupby('month_year').agg(users = ('casual', 'sum')).reset_in
sns.lineplot(x = 'month_year', y = 'users', data = grouped1, color = 'green', la
sns.lineplot(x = 'month_year', y = 'users', data = grouped2, color = 'blue', lab
plt.legend()
plt.axvline(x = '2012-01', color = 'r', linestyle = '--')
plt.xticks(rotation = 60);
plt.title('Month on Month Usage (2011 vs. 2012)');
```



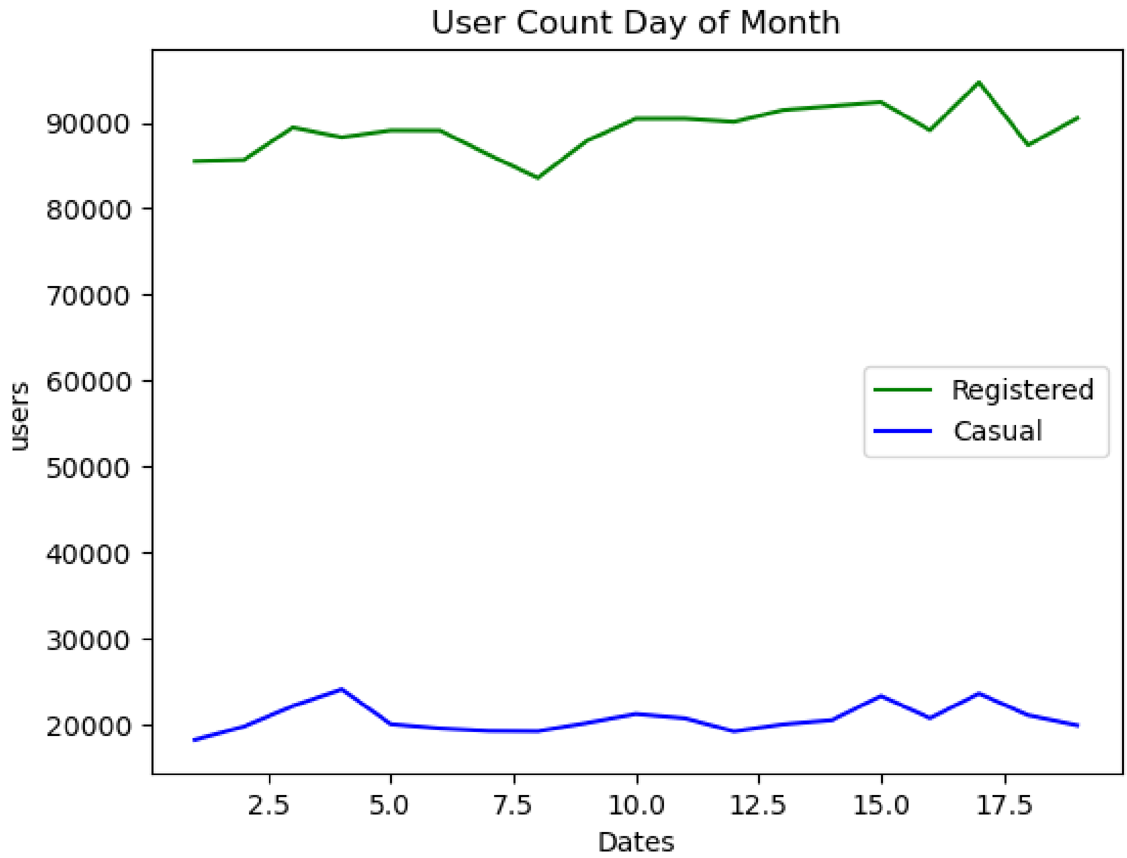
Observation: This plot shows that the number of monthly registered users have increased significantly in 2012 from 2011.

In [60]:
```
plt.figure(figsize = (8,4))
new_df = df[['datetime','casual', 'registered']]
with pd.option_context('mode.chained_assignment', None):
    new_df.loc[:,'month']= new_df['datetime'].dt.strftime('%m')
grouped1 = new_df.groupby('month').agg(users = ('registered', 'sum')).reset_inde
grouped2 = new_df.groupby('month').agg(users = ('casual', 'sum')).reset_index()
sns.lineplot(x = 'month', y = 'users', data = grouped1, color = 'green', label =
sns.lineplot(x = 'month', y = 'users', data = grouped2, color = 'blue', label =
plt.legend();
plt.xticks(rotation = 60);
plt.title('Month on Month Users');
```



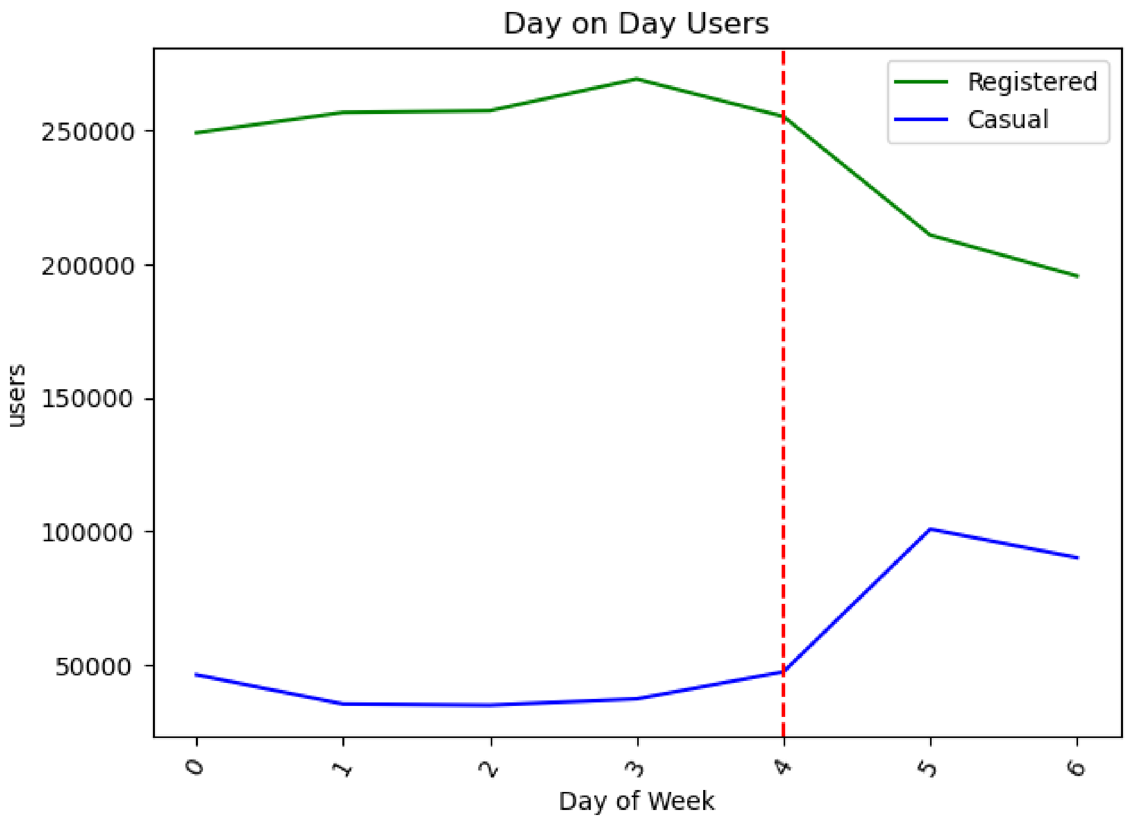
Observation- We can observe some seosonality here. It seems that less number of Users have registered in Months 1,2 and 3 compared to other months. We can also see the Peak around month 6.

In [71]:
```
grouped1 = df.groupby('day').agg(users = ('registered', 'sum')).reset_index()
grouped2 = df.groupby('day').agg(users = ('casual', 'sum')).reset_index()
sns.lineplot(x = 'day', y = 'users', data = grouped1, color = 'green', label = '
sns.lineplot(x = 'day', y = 'users', data = grouped2, color = 'blue', label = 'C
plt.legend();
plt.xlabel('Dates')
plt.title('User Count Day of Month');
```



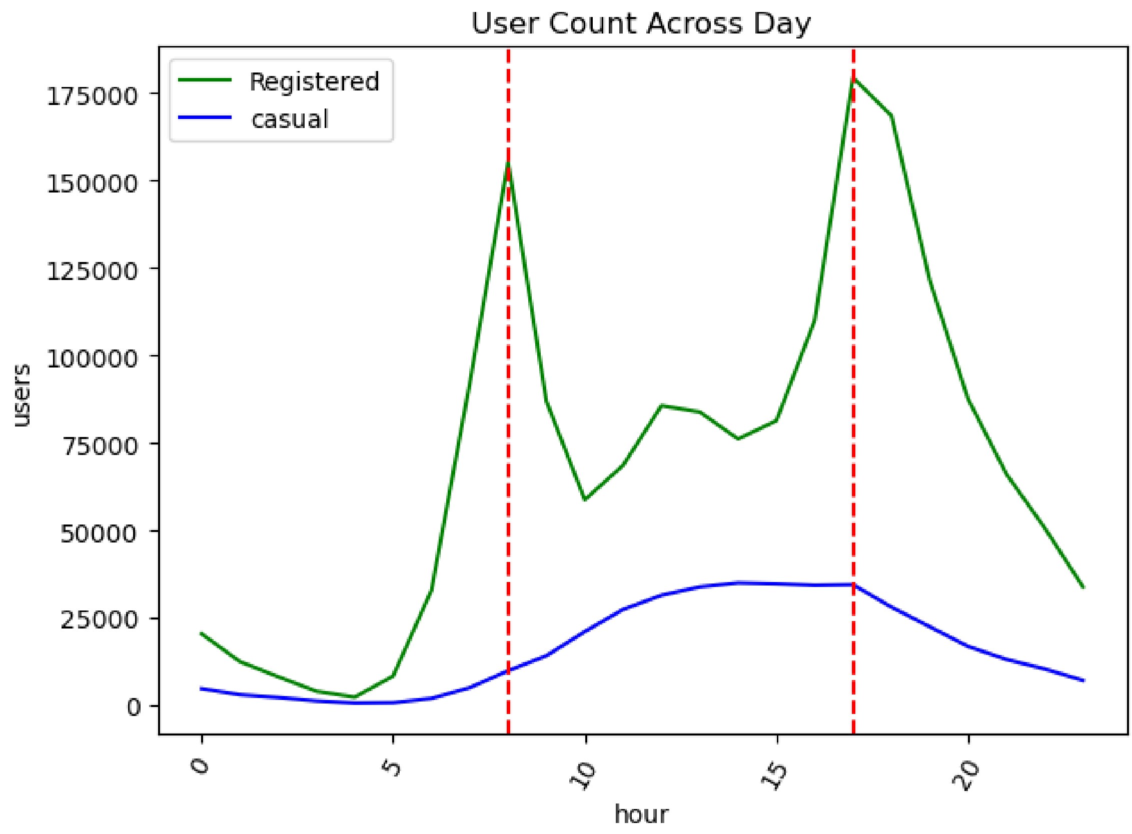
Observation - User count seems to be uniform across Dates. There is no such anomally observed in the number of users for days of months.

In [68]:
```
plt.figure(figsize = (7,5))
new_df = df[['datetime','registered', 'casual']]
with pd.option_context('mode.chained_assignment', None):
    new_df.loc[:,'day']= new_df.datetime.dt.dayofweek
grouped1 = new_df.groupby('day').agg(users = ('registered', 'sum')).reset_index(
grouped2 = new_df.groupby('day').agg(users = ('casual', 'sum')).reset_index()
sns.lineplot(x = 'day', y = 'users', data = grouped1, color = 'green', label = '
sns.lineplot(x = 'day', y = 'users', data = grouped2, color = 'blue', label = 'C
plt.axvline(x = 4, color = 'r', linestyle = '--');
plt.legend()
plt.xlabel('Day of Week')
plt.xticks(rotation = 60);
plt.title('Day on Day Users');
```



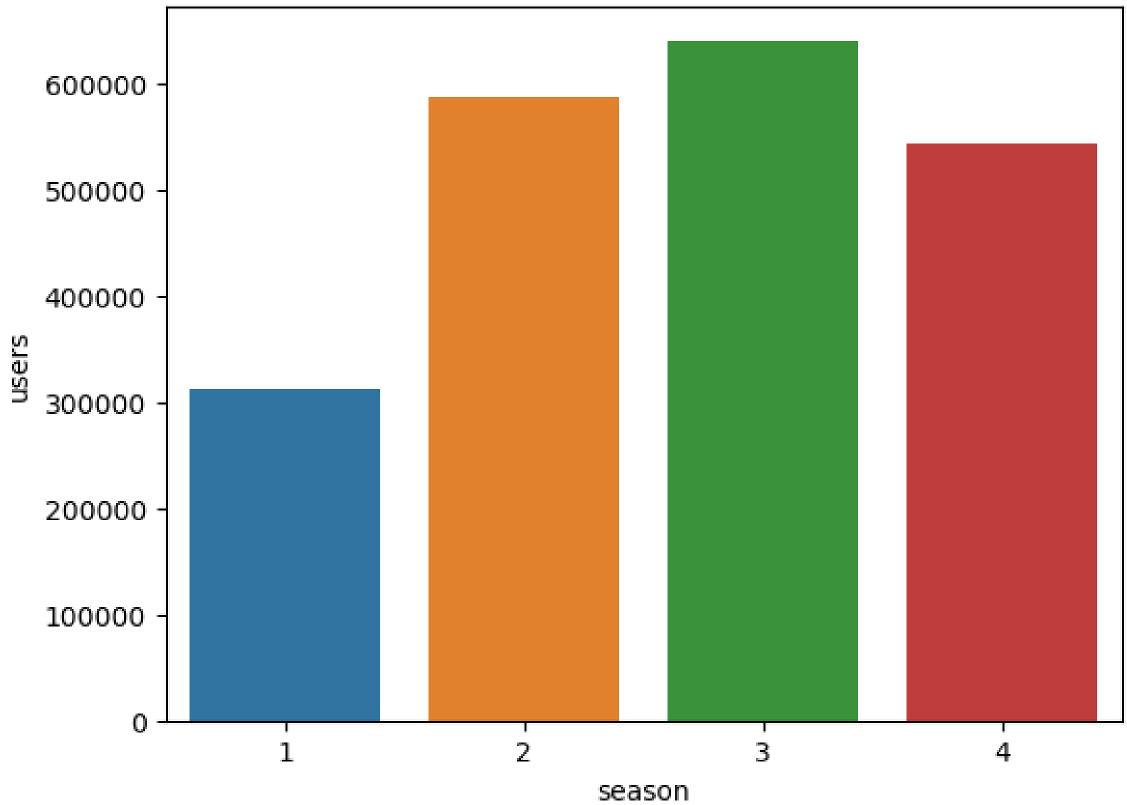
Observation - Here we can observe a contrast in the number of users. Registered Users seems to be uniform from Days 0 -3. However, from Day 4 onwards, the number starts to fall. On the otherhand, Casual Users increases from Day 4. Day 4, 5, 6 are Friday, Saturday and Sunday.

In [65]:
```
plt.figure(figsize = (7,5))
grouped1 = df.groupby('hour').agg(users = ('registered', 'sum')).reset_index()
grouped2 = df.groupby('hour').agg(users = ('casual', 'sum')).reset_index()
sns.lineplot(x = 'hour', y = 'users', data = grouped1, color = 'green', label =
sns.lineplot(x = 'hour', y = 'users', data = grouped2, color = 'blue', label = '
plt.axvline(x = 17, color = 'r', linestyle = '--');
plt.axvline(x = 8, color = 'r', linestyle = '--');
plt.xticks(rotation = 60);
plt.title('User Count Across Day');
```



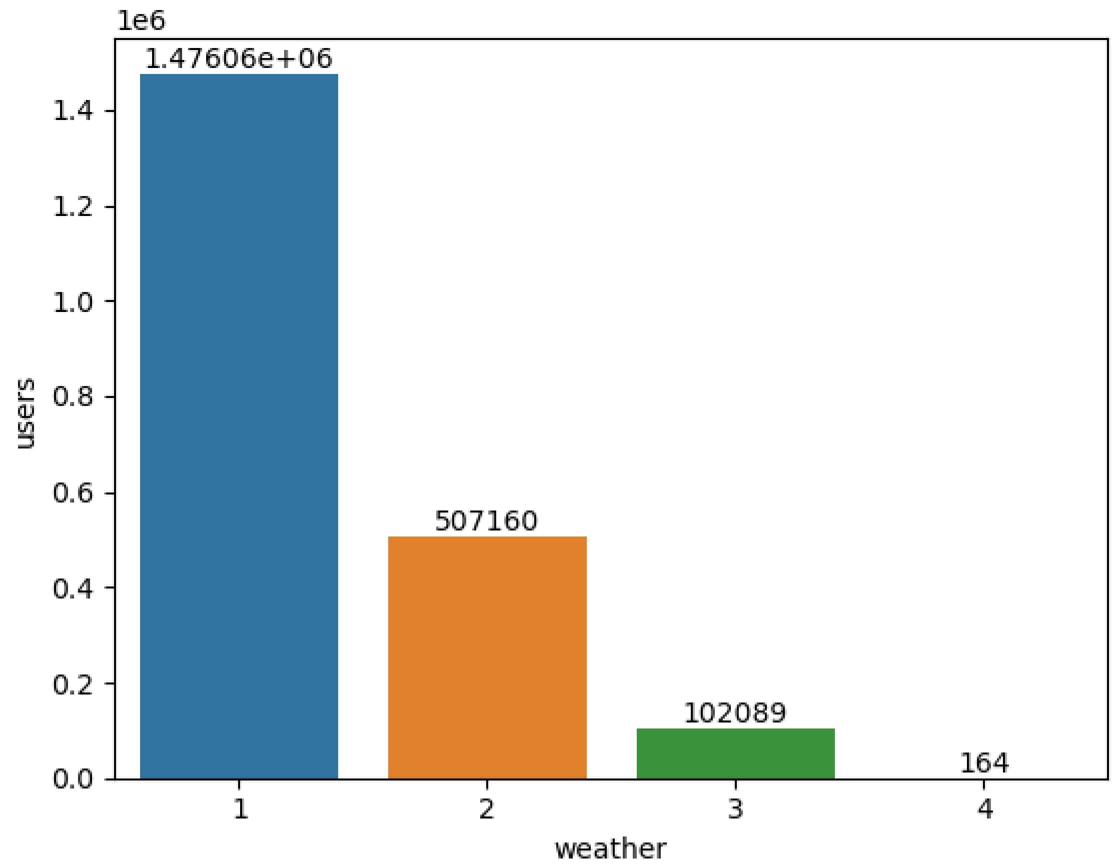
Observation - We can see 2 Peaks, one at around 8am and the other at 5pm.

In [72]:
```
season_grouped = df.groupby('season').agg(users = ('count', 'sum')).reset_index(
sns.barplot(x = 'season', y = 'users', data = season_grouped);
```



Observation - Season 3 (Fall) and Season 2 (Summer) have the most users. Season 4 (Winter) has also got significant users. Season 4 (Spring) has the least number of users. This does make sense with the above observation where we made for Months.

In [74]:
```
weather_grouped = df.groupby('weather').agg(users = ('count', 'sum')).reset_inde
bp = sns.barplot(x = 'weather', y = 'users', data = weather_grouped);
for i in bp.containers:
    bp.bar_label(i)
```



Observation - Weather 1 has got the most users followed by Weather 2 and 3. Weather 4 has got the least number of users.

Hypothesis Testing

Let's now check what features are significant for infering the User Count.

We'll start with Holiday and Working Day columns.

In [76]:
```
workingday_count = df[df.workingday == 1]['count']
non_workingday_count = df[df.workingday != 1]['count']
```

In [77]:
```
from scipy.stats import ttest_ind
```

In [86]:
```
# H0: Average Number of Users is same for both Working Day and Non Working day
# H1: Average Number of Users on Working Day is different than on Non Working da
alpha = 0.05
tstat, pval=ttest_ind(workingday_count, non_workingday_count, alternative = 'twc
if pval <= alpha:
    print(pval, '\n')
    print('Reject Null Hypothesis: Average Number of Users on Working Day is dif
else:
```

```
    print(pval, '\n')
    print('Cannot Reject Null Hypothesis: Average Number of Users is same for bo
```

```
0.22644804226361348

Cannot Reject Null Hypothesis: Average Number of Users is same for both Working
Day and Non Working day
```

In [88]:
```
holiday_count = df[df.holiday == 1]['count']
non_holiday_count = df[df.holiday != 1]['count']
```

In [89]:
```
# H0: Average Number of Users is same for both Holidays and Non holidays.
# H1: Average Number of Users on Non Holidays is higher than on Non holidays.
alpha = 0.05
tstat, pval=ttest_ind(non_holiday_count, holiday_count, alternative = 'two-sided
if pval <= alpha:
    print(pval, '\n')
    print('Reject Null Hypothesis: Average Number of Users on Non Holidays is di
else:
    print(pval, '\n')
    print('Cannot Reject Null Hypothesis: Average Number of Users is same for bo
```

```
0.5736923883271103

Cannot Reject Null Hypothesis: Average Number of Users is same for both Holiday
s and Non holidays
```

Now lets check whether the Average Number of Users are Dependent on Seasons and Weather. We'll use ANOVA for both of these columns but lets check whether the user count of the given dataset follows the assumptions of ANOVA. We'll do the Shapiro's Test and Levene's test.

In [91]:
```
from scipy.stats import shapiro, levene
```

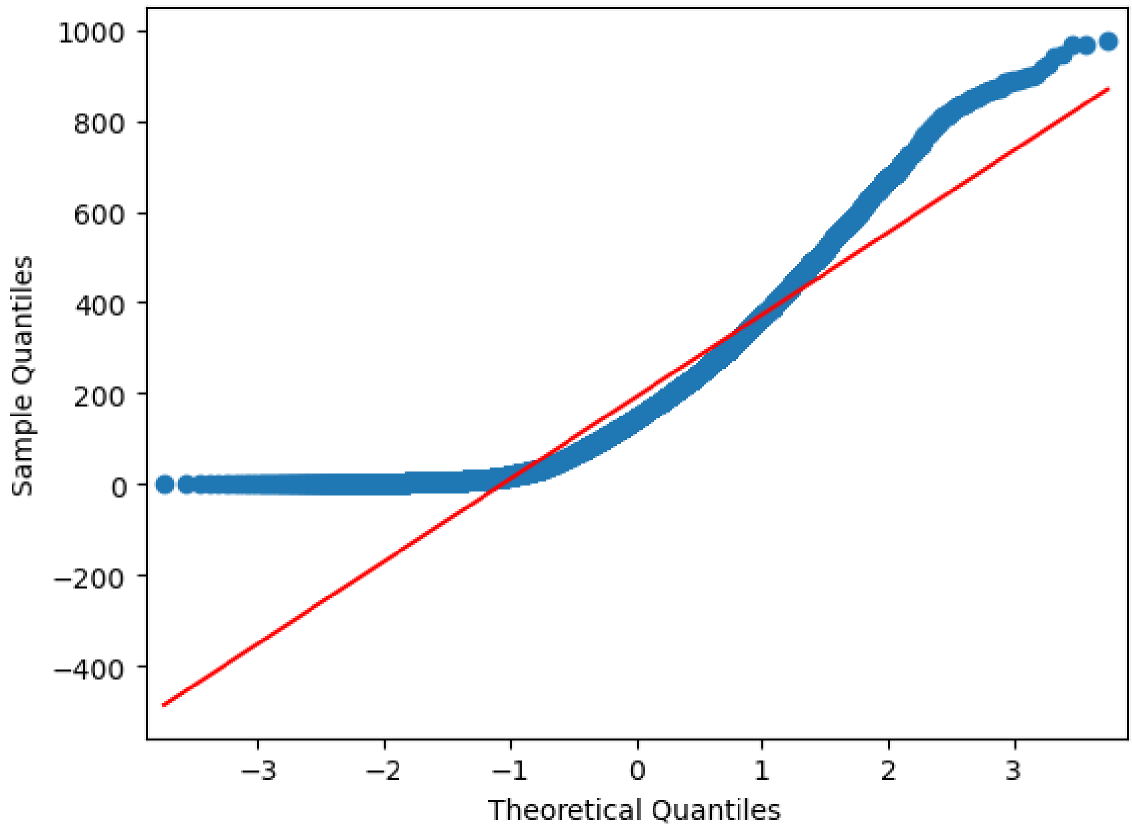
In [94]:
```
test_stat, pval = shapiro(df['count'])
if pval <= alpha:
    print('Data is not normally Distributed')
else:
    print('Data is normally Distributed')
```

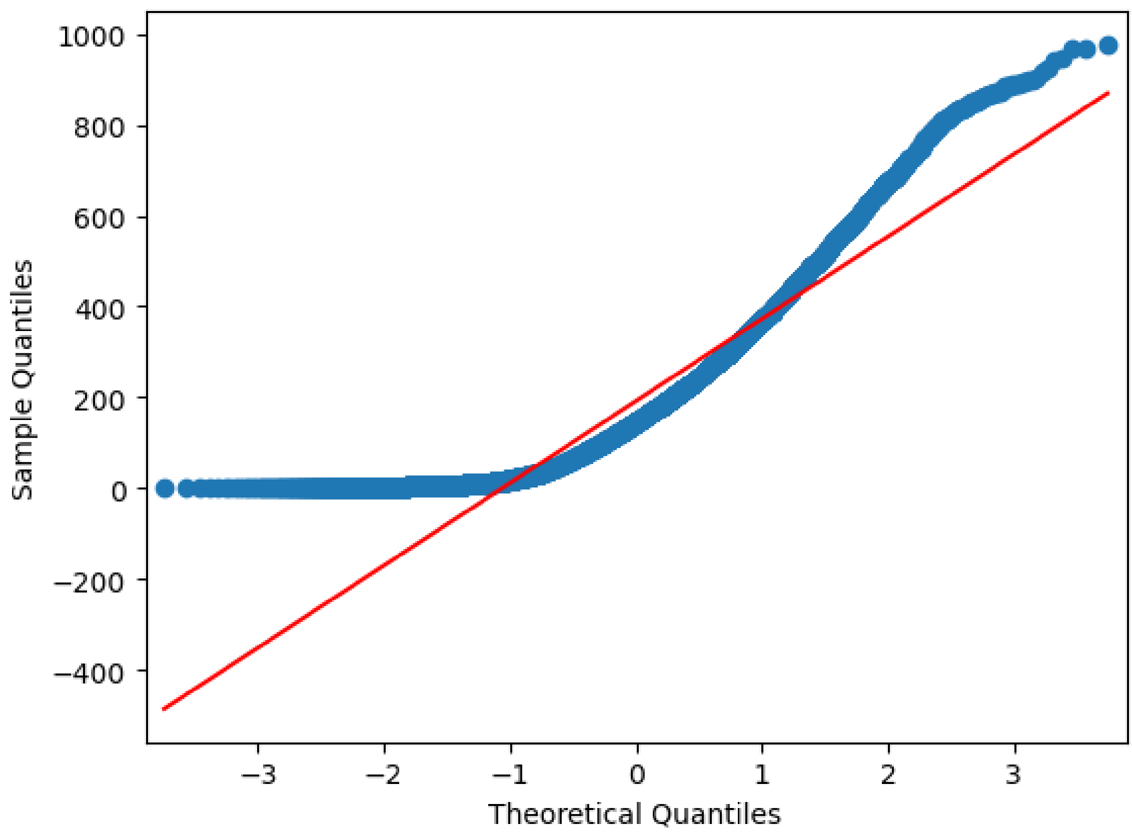
```
Data is not normally Distributed
```

Let's visualize this result.

In [95]:
```
import statsmodels.api as sm
sm.qqplot(df['count'], line = 's')
```

Out[95]:





We can now conclude that the Distribution isn't Normal. Let's check whether all groups have same variance or not. We'll do the Levene's test for Both Seasons and Weathers.

In [96]:

```
season1 = df[df.season == 1]['count']
season2 = df[df.season == 2]['count']
season3 = df[df.season == 3]['count']
season4 = df[df.season == 4]['count']
```

In [99]:
```
test_stat, pval= levene(season1, season2, season3, season4)
if pval > alpha:
    print("Equal variances (fail to reject H0)")
else:
    print("Significant differences in variances (reject H0)")
```

```
Significant differences in variances (reject H0)
```

In [101…
```
weather1 = df[df.weather == 1]['count']
weather2 = df[df.weather == 2]['count']
weather3 = df[df.weather == 3]['count']
weather4 = df[df.weather == 4]['count']
```

In [102…
```
test_stat, pval= levene(weather1, weather2, weather3, weather4)
if pval > alpha:
    print("Equal variances (fail to reject H0)")
else:
    print("Significant differences in variances (reject H0)")
```

```
Significant differences in variances (reject H0)
```

Let's still do the ANOVA to see if we see any significant results.

In [103…
```
from scipy.stats import f_oneway
```

In [104…
```
statistic, p_value = f_oneway(season1, season2, season3, season4)

# Check the p-value against the significance level
if p_value < alpha:
    print("Reject the null hypothesis: There are significant differences between
else:
    print("Fail to reject the null hypothesis: There are no significant differen
```

```
Reject the null hypothesis: There are significant differences between group mea
ns.
```

In [105…
```
statistic, p_value = f_oneway(weather1, weather2, weather3, weather4)

# Check the p-value against the significance level
if p_value < alpha:
    print("Reject the null hypothesis: There are significant differences between
else:
    print("Fail to reject the null hypothesis: There are no significant differen
```

```
Reject the null hypothesis: There are significant differences between group mea
ns.
```

Observation - Although the assumptions for ANOVA failed but ANOVA has still shown that there is some dependency on the number of users with weather and season which we observed when did the plots.

Let's check whether Season and Weather have some relationship or not.

In [109…
```
from scipy.stats import chi2_contingency
```

In [112…
```
table = pd.crosstab(df.season, df.weather)
```

In [116…

```
# H0 : There isn't a significant relationship between Seasons and Weather.
# H1 : There is a significant relationship between Seasons and Weather.
p= chi2_contingency(table)[1]
if p < alpha:
    print("There is a significant relationship between Seasons and Weather.")
else:
    print("There isn't a significant relationship between Seasons and Weather.")
```

```
There is a significant relationship between Seasons and Weather.
```

Recommendations

From the above observations, following recommendations can be made to help Yulu:

1. We have observed that there is seasonality in the User Demand during the Year. The Demand is high in the months between May and October. The company can look to increase the Inventory during this period. In the months, where demand is less, which are Jan, Feb and March, the company can look to plan maintenance activities.
2. There is significant Demand through the weekdays as most people work. Company should target the working class more during the weekdays. On the otherhand, on weekends, company can target Tourist places.
3. There are Peaks at 8am and 5pm. People usually leave for Offices in the morning and leave their office during that time. Inventory should be managed while keeping this in mind.
4. There is demand across Seasons so company should manage inventory while keeping Peak hours for these months in mind.
5. There is very less demand during Advese and Overcast weather conditions so company can look to do maintenance during this weather.

In [ ]: