In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
```

In [2]:

```
#Import libraries

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

In [3]:

```
# Importing the files
case_file_1 = pd.read_csv("/content/drive/MyDrive/Bike_share/2020_07_tripdata.csv")
case_file_2 = pd.read_csv("/content/drive/MyDrive/Bike_share/2020_08_tripdata.csv")
case_file_3 = pd.read_csv("/content/drive/MyDrive/Bike_share/2020_09_tripdata.csv")
case_file_4 = pd.read_csv("/content/drive/MyDrive/Bike_share/2020_10_tripdata.csv")
case_file_5 = pd.read_csv("/content/drive/MyDrive/Bike_share/2020_11_tripdata.csv")
case_file_6 = pd.read_csv("/content/drive/MyDrive/Bike_share/2020_12_tripdata.csv")
case_file_7 = pd.read_csv("/content/drive/MyDrive/Bike_share/2021_01_tripdata.csv")
case_file_8 = pd.read_csv("/content/drive/MyDrive/Bike_share/2021_02_tripdata.csv")
case_file_9 = pd.read_csv("/content/drive/MyDrive/Bike_share/2021_03_tripdata.csv")
case_file_10 = pd.read_csv("/content/drive/MyDrive/Bike_share/2021_04_tripdata.csv")
case_file_11 = pd.read_csv("/content/drive/MyDrive/Bike_share/2021_05_tripdata.csv")
case_file_12 = pd.read_csv("/content/drive/MyDrive/Bike_share/2021_06_tripdata.csv")
```

In [4]:

```
# Merging all the dataframes into a single dataframe.

all_dfs = [case_file_1, case_file_2, case_file_3, case_file_4, case_file_5, case_file_6, ca
case_file = pd.concat(all_dfs)
df = case_file
```

In [5]:

```
# Reading the merged file

case_file_5000 = df
case_file_5000
```

Out[5]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_stat
0	762198876D69004D	docked_bike	2020-07-09 15:22:02	2020-07-09 15:25:52	Ritchie Ct & Banks St	
1	BEC9C9FBA0D4CF1B	docked_bike	2020-07-24 23:56:30	2020-07-25 00:20:17	Halsted St & Roscoe St	
2	D2FD8EA432C77EC1	docked_bike	2020-07-08 19:49:07	2020-07-08 19:56:22	Lake Shore Dr & Diversey Pkwy	
3	54AE594E20B35881	docked_bike	2020-07-17 19:06:42	2020-07-17 19:27:38	LaSalle St & Illinois St	
4	54025FDC7440B56F	docked_bike	2020-07-04 10:39:57	2020-07-04 10:45:05	Lake Shore Dr & North Blvd	
...	...	...	...	...	...	
729590	CB282292CCFCE74F	electric_bike	2021-06-14 00:17:31	2021-06-14 00:56:46	Wells St & Polk St	S
729591	47BD346FAFB9BE6D	classic_bike	2021-06-30 17:35:10	2021-06-30 17:43:20	Clark St & Chicago Ave	
729592	52467C23D17C6AFE	classic_bike	2021-06-13 19:24:30	2021-06-13 19:34:11	Indiana Ave & 26th St	TA13070
729593	7DF6D74420D7D9E6	electric_bike	2021-06-08 15:44:28	2021-06-08 16:15:01	Clark St & Chicago Ave	
729594	0C01F8BA99E512E5	electric_bike	2021-06-03 16:18:38	2021-06-03 16:47:49	Clark St & Chicago Ave	

4460151 rows × 13 columns

In [6]:

```
# To get information about the dataframe.
case_file_5000.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4460151 entries, 0 to 729594
Data columns (total 13 columns):
 #   Column              Dtype  
---  ------              -----  
 0   ride_id             object 
 1   rideable_type       object 
 2   started_at          object 
 3   ended_at            object 
 4   start_station_name  object 
 5   start_station_id    object 
 6   end_station_name    object 
 7   end_station_id      object 
 8   start_lat           float64
 9   start_lng           float64
 10  end_lat             float64
 11  end_lng             float64
 12  member_casual       object 
dtypes: float64(4), object(9)
memory usage: 476.4+ MB
```

In [7]:

```
#To convert object data type of start_time and end_time columns into datetime data type
case_file_5000["start_time"] = pd.to_datetime(case_file_5000["started_at"])
case_file_5000["end_time"] = pd.to_datetime(case_file_5000["ended_at"])
```

In [8]:

```
#Adding new column - ride_length (ended_at - started_at) to calculate the time travelled
case_file_5000["ride_length"] = case_file_5000["end_time"] - case_file_5000["start_time"]
case_file_5000["ride_length"]
```

Out[8]:

```
0         0 days 00:03:50
1         0 days 00:23:47
2         0 days 00:07:15
3         0 days 00:20:56
4         0 days 00:05:08
                ...
729590    0 days 00:39:15
729591    0 days 00:08:10
729592    0 days 00:09:41
729593    0 days 00:30:33
729594    0 days 00:29:11
Name: ride_length, Length: 4460151, dtype: timedelta64[ns]
```

In [9]:

```
# To get the updated information about the changes made in the data type of dataframe.
case_file_5000.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4460151 entries, 0 to 729594
Data columns (total 16 columns):
 #   Column              Dtype         
---  ------              -----         
 0   ride_id             object        
 1   rideable_type       object        
 2   started_at          object        
 3   ended_at            object        
 4   start_station_name  object        
 5   start_station_id    object        
 6   end_station_name    object        
 7   end_station_id      object        
 8   start_lat           float64       
 9   start_lng           float64       
 10  end_lat             float64       
 11  end_lng             float64       
 12  member_casual       object        
 13  start_time          datetime64[ns]
 14  end_time            datetime64[ns]
 15  ride_length         timedelta64[ns]
dtypes: datetime64[ns](2), float64(4), object(9), timedelta64[ns](1)
memory usage: 578.5+ MB
```

In [10]:

```
# Splitting and extracting the date from "started_at" column
case_file_5000["start_date1"] = [i.split(" ")[0] for i in case_file_5000["started_at"]]
case_file_5000["start_date1"]
```

Out[10]:

```
0         2020-07-09
1         2020-07-24
2         2020-07-08
3         2020-07-17
4         2020-07-04
             ...    
729590    2021-06-14
729591    2021-06-30
729592    2021-06-13
729593    2021-06-08
729594    2021-06-03
Name: start_date1, Length: 4460151, dtype: object
```

In [11]:

```
# Converting start_date1 into datetime data type
case_file_5000["start_date1"] = pd.to_datetime(case_file_5000["start_date1"])
```

In [12]:

```
# Finding the weekday of the date
case_file_5000['week_day'] = case_file_5000["start_date1"].apply(lambda x: x.weekday())
case_file_5000['week_day'].unique()
```

Out[12]:

```
array([3, 4, 2, 5, 1, 0, 6])
```

In [13]:

```
# Converting weekday values into weekday
dict1 = {0: "Sunday", 1: "Monday", 2: "Tuesday", 3: "Wednesday", 4: "Thursday", 5: "Friday"
case_file_5000['week_day'] = case_file_5000['week_day'].apply(lambda y: dict1[y])
case_file_5000['week_day']
```

Out[13]:

```
0         Wednesday
1          Thursday
2           Tuesday
3          Thursday
4            Friday
            ...
729590       Sunday
729591      Tuesday
729592     Saturday
729593       Monday
729594    Wednesday
Name: week_day, Length: 4460151, dtype: object
```

In [14]:

```
# To sort the dataframe by start date in ascending order
case_file_5000.sort_values(by=['started_at'], inplace=True, ascending=True)
```

In [15]:

```
# Drop all rows with any NaN values
case_file_5000_cleaned = case_file_5000.dropna()
```

In [16]:

```
# To save the cleaned file for further transformation of data
case_file_5000_cleaned.to_csv('cleaned.csv')
```

In [17]:

```
# To import the cleaned data file
cleaned_file = pd.read_csv("cleaned.csv")
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:271
8: DtypeWarning: Columns (6,8) have mixed types.Specify dtype option on impo
rt or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [18]:

```
# Print the cleaned dataframe
cleaned_file
```

Out[18]:

	Unnamed: 0	ride_id	rideable_type	started_at	ended_at	start_station_nam
0	275150	C66CC4DD911DDFBD	docked_bike	2020-07-01 00:00:14	2020-07-01 01:28:12	Buckingha Founta
1	259045	BD6363747C00FAD3	docked_bike	2020-07-01 00:00:15	2020-07-01 02:44:58	Wells St & Conco L
2	236929	1856297CBD809E60	docked_bike	2020-07-01 00:00:49	2020-07-01 00:45:04	Wabash Ave Wacker
3	217019	06B27D89815F5031	docked_bike	2020-07-01 00:00:50	2020-07-01 02:52:16	Wells St & Conco L
4	195188	7F17B8FF6F418A7F	docked_bike	2020-07-01 00:01:11	2020-07-01 00:08:03	State St & 33rd
...	...	...	...	...	...	
4025498	675978	E25F3A2D6AF8F001	classic_bike	2021-06-30 23:59:15	2021-07-01 00:06:09	Calumet Ave & 18
4025499	582358	F67C4ACB3CCEC518	classic_bike	2021-06-30 23:59:16	2021-07-01 00:04:21	Ashland Ave Chicago Av
4025500	16931	B6A748160720820F	classic_bike	2021-06-30 23:59:26	2021-07-01 00:06:11	Calumet Ave & 18
4025501	626434	5F69DFB0122DC3D8	classic_bike	2021-06-30 23:59:35	2021-07-01 00:27:54	Michigan Ave Lake
4025502	156646	428577054CA0238A	classic_bike	2021-06-30 23:59:59	2021-07-01 00:36:07	Wabash Ave Cermak F

4025503 rows × 19 columns

In [19]:

```
# To calculate euclidean distance between start and end point.
cleaned_file["x"] = (cleaned_file["end_lat"] - cleaned_file["start_lat"])**2
cleaned_file["y"] = (cleaned_file["end_lng"] - cleaned_file["start_lng"])**2
cleaned_file["distance"] = (cleaned_file["x"] + cleaned_file["y"])**(1/2)
cleaned_file["distance"]
```

Out[19]:

```
0          2.290298e-07
1          2.638434e-02
2          0.000000e+00
3          2.638434e-02
4          1.145522e-02
              ...
4025498    1.163711e-02
4025499    1.104382e-02
4025500    1.163711e-02
4025501    2.093745e-03
4025502    7.064478e-02
Name: distance, Length: 4025503, dtype: float64
```

In [20]:

```
# To extract year and month from start date column
cleaned_file['year'] = pd.DatetimeIndex(cleaned_file["start_date1"]).year
cleaned_file['month'] = pd.DatetimeIndex(cleaned_file["start_date1"]).month
```

In [21]:

```
# To convert object datatype to timedelta
cleaned_file["ride_len"] = pd.to_timedelta(cleaned_file["ride_length"])
```

In [22]:

```
# Print the cleaned dataframe
cleaned_file
```

Out[22]:

	Unnamed: 0	ride_id	rideable_type	started_at	ended_at	start_station_nam
0	275150	C66CC4DD911DDFBD	docked_bike	2020-07-01 00:00:14	2020-07-01 01:28:12	Buckingha Founta
1	259045	BD6363747C00FAD3	docked_bike	2020-07-01 00:00:15	2020-07-01 02:44:58	Wells St & Conco L
2	236929	1856297CBD809E60	docked_bike	2020-07-01 00:00:49	2020-07-01 00:45:04	Wabash Ave Wacker
3	217019	06B27D89815F5031	docked_bike	2020-07-01 00:00:50	2020-07-01 02:52:16	Wells St & Conco L
4	195188	7F17B8FF6F418A7F	docked_bike	2020-07-01 00:01:11	2020-07-01 00:08:03	State St & 33rd
...	...	...	...	...	...	
4025498	675978	E25F3A2D6AF8F001	classic_bike	2021-06-30 23:59:15	2021-07-01 00:06:09	Calumet Ave & 18
4025499	582358	F67C4ACB3CCEC518	classic_bike	2021-06-30 23:59:16	2021-07-01 00:04:21	Ashland Ave Chicago Av
4025500	16931	B6A748160720820F	classic_bike	2021-06-30 23:59:26	2021-07-01 00:06:11	Calumet Ave & 18
4025501	626434	5F69DFB0122DC3D8	classic_bike	2021-06-30 23:59:35	2021-07-01 00:27:54	Michigan Ave Lake
4025502	156646	428577054CA0238A	classic_bike	2021-06-30 23:59:59	2021-07-01 00:36:07	Wabash Ave Cermak F

4025503 rows × 25 columns

In [23]:

```
# To check if there are null values in dataframe
cleaned_file.isnull().sum()
```

Out[23]:

```
Unnamed: 0            0
ride_id               0
rideable_type         0
started_at            0
ended_at              0
start_station_name    0
start_station_id      0
end_station_name      0
end_station_id        0
start_lat             0
start_lng             0
end_lat               0
end_lng               0
member_casual         0
start_time            0
end_time              0
ride_length           0
start_date1           0
week_day              0
x                     0
y                     0
distance              0
year                  0
month                 0
ride_len              0
dtype: int64
```

In [24]:

```
# To check if there are any duplicate values in dataframe
cleaned_file.duplicated().any()
```

Out[24]:

```
False
```

Analysis

In [25]:

```
#To calculate mean of ride length of all types of users
mean_ride_length = cleaned_file["ride_len"].mean()
mean_ride_length
```

Out[25]:

```
Timedelta('0 days 00:24:09.360428994')
```

In [26]:

```
# Calculate the max ride_length of all types of users
max_ride_length = cleaned_file["ride_len"].max()
max_ride_length
```

Out[26]:

```
Timedelta('38 days 20:24:09')
```

In [27]:

```
# To calculate mean ride length for member
mean_ride_length_member = cleaned_file[cleaned_file["member_casual"] == "member"]
mean_ride_length_member = mean_ride_length_member["ride_len"].mean()
mean_ride_length_member
```

Out[27]:

```
Timedelta('0 days 00:11:35.488927758')
```

In [28]:

```
# To calculate mean ride length for casual riders
mean_ride_length_casual = cleaned_file[cleaned_file["member_casual"] == "casual"]
mean_ride_length_casual = mean_ride_length_casual["ride_len"].mean()
mean_ride_length_casual
```

Out[28]:

```
Timedelta('0 days 00:40:51.870952414')
```

In [29]:

```
# To calculate max ride length for member
max_ride_length_member = cleaned_file[cleaned_file["member_casual"] == "member"]
max_ride_length_member = max_ride_length_member["ride_len"].max()
max_ride_length_member
```

Out[29]:

```
Timedelta('23 days 05:01:22')
```

In [30]:

```
# To calculate max ride length for casual riders
max_ride_length_casual = cleaned_file[cleaned_file["member_casual"] == "casual"]
max_ride_length_casual = max_ride_length_casual["ride_len"].max()
max_ride_length_casual
```

Out[30]:

```
Timedelta('38 days 20:24:09')
```

In [31]:

```
# Calculate the mode of week_day for all types of users
mode_week_day = cleaned_file["week_day"].mode()
mode_week_day
```

Out[31]:

```
0    Friday
dtype: object
```

In [32]:

```
# To calculate mode of week_day for member
mode_week_day_member = cleaned_file[cleaned_file["member_casual"] == "member"]
mode_week_day_member = mode_week_day_member["week_day"].mode()
mode_week_day_member
```

Out[32]:

```
0    Tuesday
dtype: object
```

In [33]:

```
# To calculate mode of week_day for casual
mode_week_day_casual = cleaned_file[cleaned_file["member_casual"] == "casual"]
mode_week_day_casual = mode_week_day_casual["week_day"].mode()
mode_week_day_casual
```

Out[33]:

```
0    Friday
dtype: object
```

In [34]:

```
# Calculate the number of rides for users by week_day by adding Count of ride_id to Values
dict1 = {"Sunday": 0, "Monday": 1, "Tuesday": 2, "Wednesday": 3, "Thursday": 4, "Friday":5,
cleaned_file['week_day'] = cleaned_file['week_day'].apply(lambda y: dict1[y])

count_users_week_day = cleaned_file.groupby('week_day')['ride_id'].count()
count_users_week_day = count_users_week_day.sort_index()
count_users_week_day
```

Out[34]:

```
week_day
0    492594
1    511069
2    544916
3    518999
4    590976
5    740118
6    626831
Name: ride_id, dtype: int64
```

In [35]:

```
# Calculate the number of rides for member by week_day by adding Count of ride_id to Values
count_member_week_day = cleaned_file[cleaned_file["member_casual"] == "member"].groupby('we
count_member_week_day
```

Out[35]:

```
week_day
0    306654
1    331260
2    354757
3    331126
4    340117
5    340617
6    293152
Name: ride_id, dtype: int64
```

In [36]:

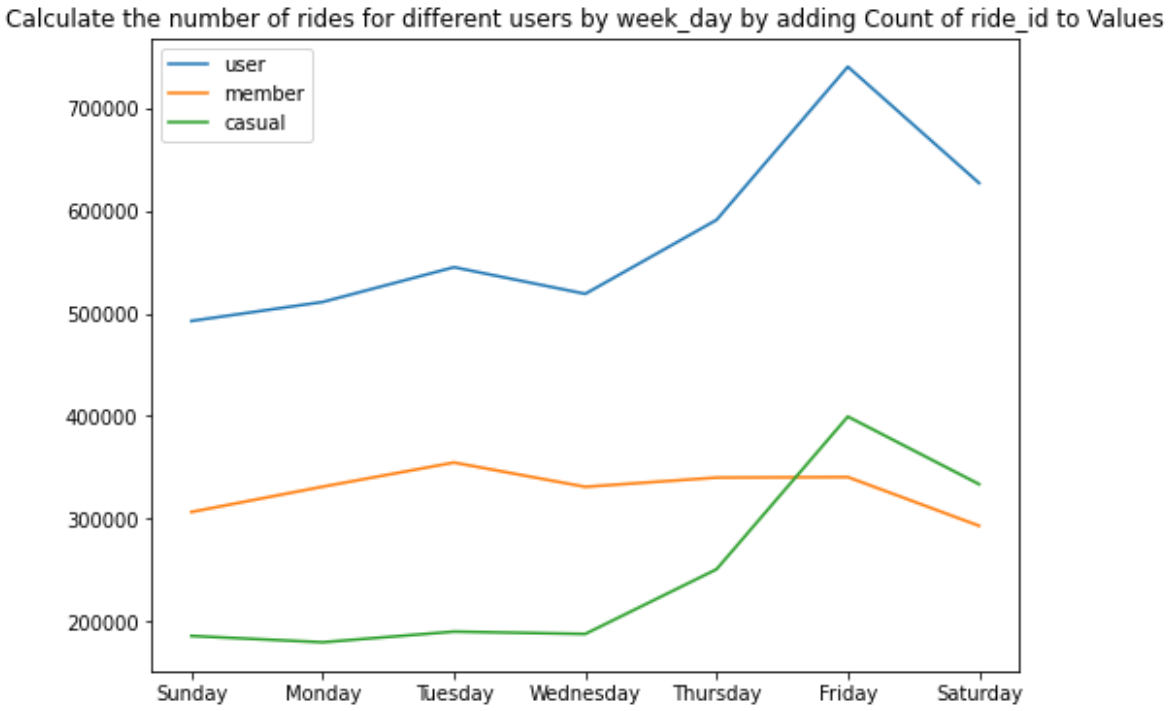
```
# Calculate the number of rides for casual by week_day by adding Count of ride_id to Values
count_casual_week_day = cleaned_file[cleaned_file["member_casual"] == "casual"].groupby('we
count_casual_week_day
```

Out[36]:

```
week_day
0    185940
1    179809
2    190159
3    187873
4    250859
5    399501
6    333679
Name: ride_id, dtype: int64
```

In [37]:

```
# To plot graph of Weekday Vs Number Of Rides
plt.figure(figsize = (8,6))
plt.plot(count_users_week_day.index, count_users_week_day.values)
plt.plot(count_member_week_day.index, count_member_week_day.values)
plt.plot(count_casual_week_day.index, count_casual_week_day.values)
plt.title("Calculate the number of rides for different users by week_day by adding Count of
plt.legend(["user", "member", "casual"])
labels = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
plt.xticks(count_casual_week_day.index, labels)
plt.show()
```



In [38]:

```
# Calculate the average ride_length for users by week_day
user_avg_ride_len_week_day = cleaned_file.groupby("week_day")['ride_len'].mean(numeric_only
user_avg_ride_len_week_day
```

Out[38]:

```
week_day
0   0 days 00:24:06.710595338
1   0 days 00:04:27.933680187
2   0 days 00:22:23.169934448
3   0 days 00:22:39.126884637
4   0 days 00:25:21.690127856
5   0 days 00:31:24.290038885
6   0 days 00:33:19.983402224
Name: ride_len, dtype: timedelta64[ns]
```

In [39]:

```
# Calculate the average ride_length for members by week_day
member_avg_ride_len_week_day = cleaned_file[cleaned_file["member_casual"] == "member"].grou
member_avg_ride_len_week_day
```

Out[39]:

```
week_day
0     0 days 00:14:13.133508775
1    -1 days +23:51:37.289627483
2     0 days 00:14:01.688383879
3     0 days 00:13:58.298834884
4     0 days 00:14:32.475189420
5     0 days 00:16:19.305604241
6     0 days 00:16:51.198811538
Name: ride_len, dtype: timedelta64[ns]
```

In [40]:

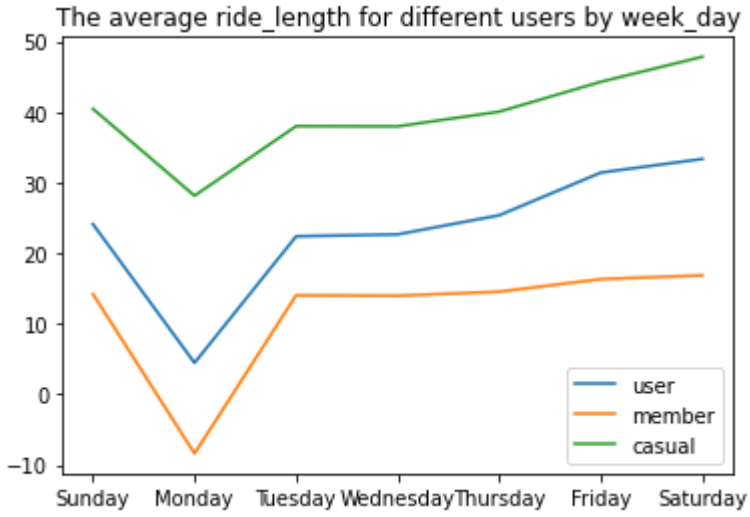
```
# Calculate the average ride_length for casual by week_day
casual_avg_ride_len_week_day = cleaned_file[cleaned_file["member_casual"] == "casual"].grou
casual_avg_ride_len_week_day
```

Out[40]:

```
week_day
0   0 days 00:40:25.643519414
1   0 days 00:28:07.682129370
2   0 days 00:37:58.724341209
3   0 days 00:37:57.085871838
4   0 days 00:40:01.901869177
5   0 days 00:44:15.885311926
6   0 days 00:47:48.675109911
Name: ride_len, dtype: timedelta64[ns]
```

In [41]:

```
# To plot graph of Weekday Vs Average ride length
plt.plot(user_avg_ride_len_week_day/pd.Timedelta(minutes=1))
plt.plot(member_avg_ride_len_week_day/pd.Timedelta(minutes=1))
plt.plot(casual_avg_ride_len_week_day/pd.Timedelta(minutes=1))
plt.title("The average ride_length for different users by week_day")
plt.legend(["user", "member", "casual"])
labels = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
plt.xticks(user_avg_ride_len_week_day.index, labels)
plt.show()
```



In [42]:

```
cleaned_file
```

Out[42]:

	Unnamed: 0	ride_id	rideable_type	started_at	ended_at	start_station_nam
0	275150	C66CC4DD911DDFBD	docked_bike	2020-07-01 00:00:14	2020-07-01 01:28:12	Buckingha Founta
1	259045	BD6363747C00FAD3	docked_bike	2020-07-01 00:00:15	2020-07-01 02:44:58	Wells St & Conco
2	236929	1856297CBD809E60	docked_bike	2020-07-01 00:00:49	2020-07-01 00:45:04	Wabash Ave Wacker
3	217019	06B27D89815F5031	docked_bike	2020-07-01 00:00:50	2020-07-01 02:52:16	Wells St & Conco
4	195188	7F17B8FF6F418A7F	docked_bike	2020-07-01 00:01:11	2020-07-01 00:08:03	State St & 33rd
...	...	...	...	...	...	
4025498	675978	E25F3A2D6AF8F001	classic_bike	2021-06-30 23:59:15	2021-07-01 00:06:09	Calumet Ave & 18
4025499	582358	F67C4ACB3CCEC518	classic_bike	2021-06-30 23:59:16	2021-07-01 00:04:21	Ashland Ave Chicago Av
4025500	16931	B6A748160720820F	classic_bike	2021-06-30 23:59:26	2021-07-01 00:06:11	Calumet Ave & 18
4025501	626434	5F69DFB0122DC3D8	classic_bike	2021-06-30 23:59:35	2021-07-01 00:27:54	Michigan Ave Lake
4025502	156646	428577054CA0238A	classic_bike	2021-06-30 23:59:59	2021-07-01 00:36:07	Wabash Ave Cermak F

4025503 rows × 25 columns

In [43]:

```
# To extract important features of the dataframe into a new dataframe
final_file = cleaned_file[["ride_id", "rideable_type", "ride_len", "week_day", "distance",
```

In [44]:

```
# To get the information of the dataframe
final_file.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4025503 entries, 0 to 4025502
Data columns (total 8 columns):
 #   Column         Dtype          
---  ------         -----          
 0   ride_id        object         
 1   rideable_type  object         
 2   ride_len       timedelta64[ns]
 3   week_day       int64          
 4   distance       float64        
 5   year           int64          
 6   month          int64          
 7   member_casual  object         
dtypes: float64(1), int64(3), object(3), timedelta64[ns](1)
memory usage: 245.7+ MB
```

data analysis

In [45]:

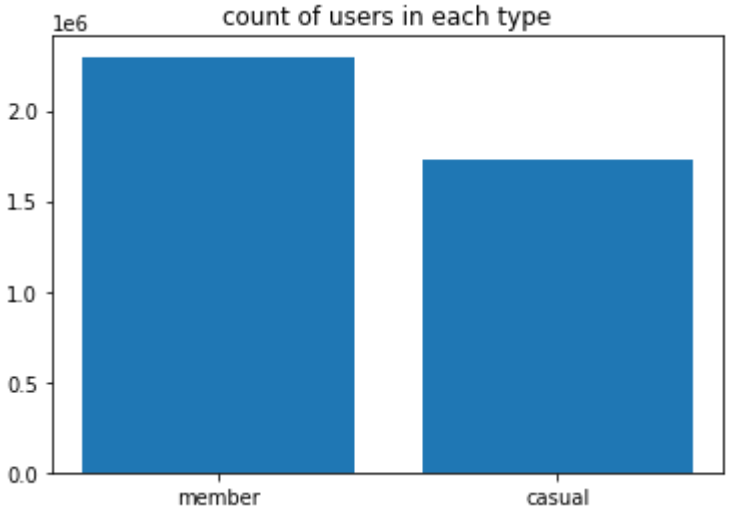
```
# To check the number of users in each type over past 12 months
member_type = final_file["member_casual"].value_counts()
member_type
```

Out[45]:

```
member    2297683
casual    1727820
Name: member_casual, dtype: int64
```

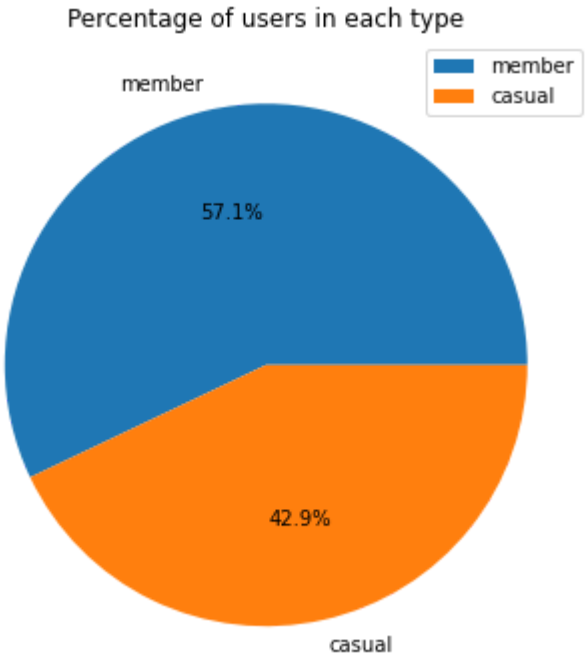
In [46]:

```
# To plot the count plot of users in each type
plt.title("count of users in each type")
plt.bar(member_type.index, member_type.values)
plt.show()
```



In [47]:

```
# To plot the pie chart of users in each type
plt.figure(figsize = (8,6))
plt.pie(member_type.values, labels = member_type.index, autopct='%1.1f%%')
plt.title("Percentage of users in each type")
plt.legend(member_type.index)
plt.show()
```



In [48]:

```
# To check the number of users over years
users_over_year = final_file.groupby('year')['member_casual'].value_counts()
users_over_year
```

Out[48]:

```
year  member_casual
2020  member           1347894
      casual            987434
2021  member            949789
      casual            740386
Name: member_casual, dtype: int64
```

In [49]:

```
# To segregate the number of users into member or casual riders over the years
list_year = []
list_x = []
list_y = list(users_over_year.values)
for i, j in users_over_year.index:
  list_x.append(j)
  list_year.append(str(i))
```

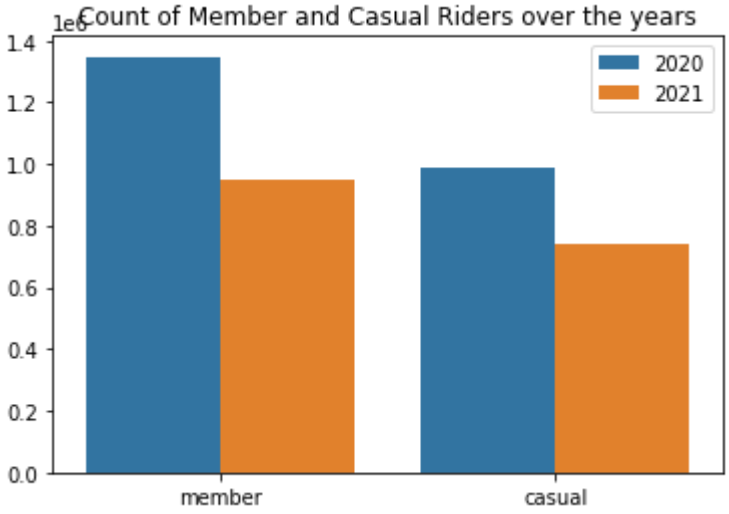
In [50]:

```
sns.barplot(list_x, list_y, hue = list_year)
plt.title("Count of Member and Casual Riders over the years")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarn
ing: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[50]:

```
Text(0.5, 1.0, 'Count of Member and Casual Riders over the years')
```



In [51]:

```
# To check the number of users over months
users_over_month = final_file.groupby('month')['member_casual'].value_counts()
users_over_month
```

Out[51]:

```
month  member_casual
1      member            68819
       casual            14690
2      member            34383
       casual             8613
3      member           130049
       casual            75642
4      member           177787
       casual           120420
5      member           234165
       casual           216829
6      member           304586
       casual           304192
7      member           281692
       casual           268733
8      member           325504
       casual           283006
9      member           285090
       casual           215300
10     member           216493
       casual           122810
11     member           149756
       casual            73033
12     member            89359
       casual            24552
Name: member_casual, dtype: int64
```

In [52]:

```
# To segregate the number of users into member or casual riders over the months
list_month = []
list_x = []
list_y = list(users_over_month.values)
for i, j in users_over_month.index:
  list_x.append(j)
  list_month.append(str(i))
```

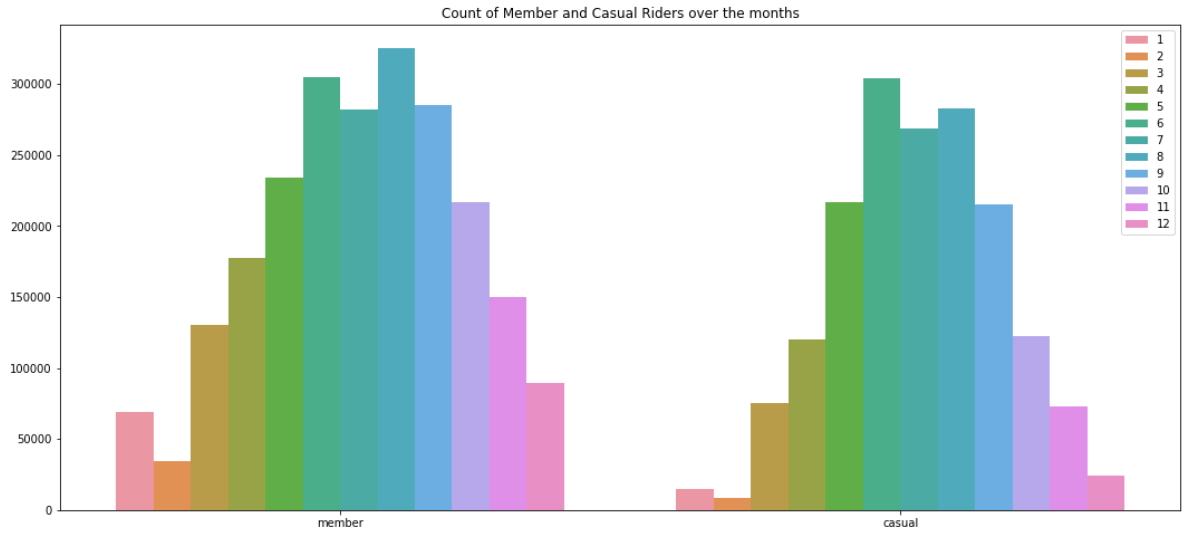
In [53]:

```
# Plot count of Member and Casual Riders over the months
plt.figure(figsize = (18,8))
sns.barplot(list_x, list_y, hue = list_month)
plt.title("Count of Member and Casual Riders over the months")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarn
ing: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[53]:

```
Text(0.5, 1.0, 'Count of Member and Casual Riders over the months')
```



In [54]:

```
# To check the number of users for a particulat rideable_type
ride_type = final_file.groupby("member_casual")["rideable_type"].value_counts()
ride_type
```

Out[54]:

```
member_casual  rideable_type
casual         docked_bike       969539
               classic_bike      451826
               electric_bike     306455
member         docked_bike      1077368
               classic_bike      823033
               electric_bike     397282
Name: rideable_type, dtype: int64
```

In [55]:

```
# To segregate the number of users into member or casual riders based on ride type
list_member = []
list_x = []
list_y = list(ride_type.values)
for i, j in ride_type.index:
  list_x.append(j)
  list_member.append(str(i))
```

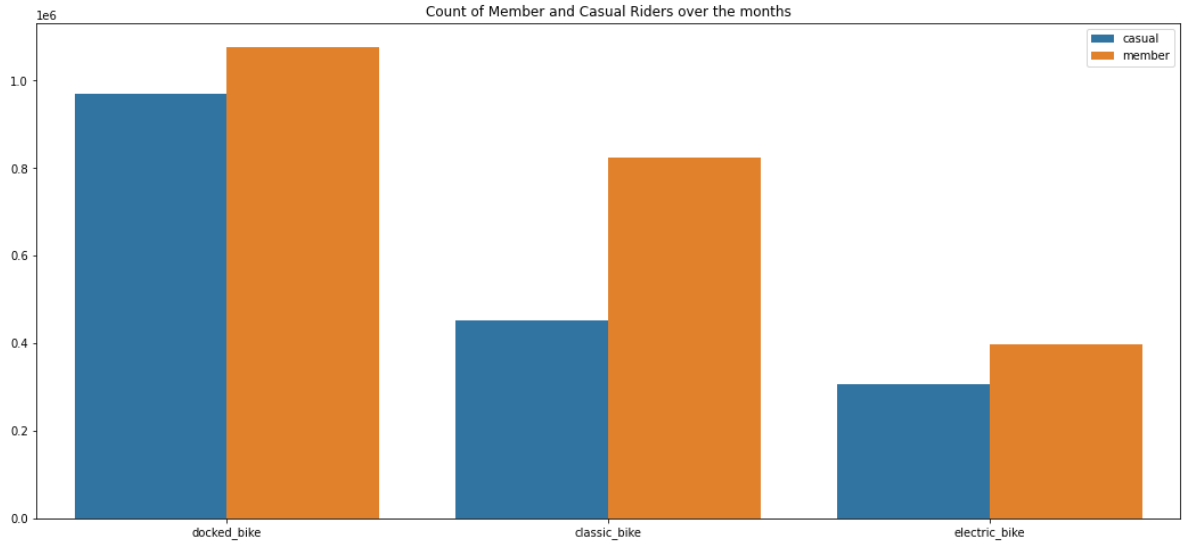
In [56]:

```
# To plot number of users into member or casual riders based on ride type
plt.figure(figsize = (18,8))
sns.barplot(list_x, list_y, hue = list_member)
plt.title("Count of Member and Casual Riders over the months")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarn
ing: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[56]:

```
Text(0.5, 1.0, 'Count of Member and Casual Riders over the months')
```



In [56]: