Cyclistic, a bike-share company in Chicago. Company’s future success depends on maximizing the number of annual memberships.must be backed up with compelling data insights and professional data.

- Cyclistic is a bike-share program that features 5,824 bicycles and 692 docking stations.
- Cyclistic users are more likely to ride for leisure, but about 30% use them to commute to work each day.
- It has 3 flexible pricing plans- single-ride passes, full-day passes, and annual memberships.
- Single-ride or full-day pass customers are casual riders while those who purchase annual memberships are Cyclistic members.

Scenario

As a Junior data analyst, marketing analyst team at Cyclistic believes the company’s future success depends on maximizing the number of annual memberships, as they are much more profitable than casual riders. Instead of targeting all-new customers, focus of the marketing strategy is on converting casual riders to annual members.

Ask

Three questions will guide the future marketing program:

- How do annual members and casual riders use Cyclistic bikes differently?
- Why would casual riders buy Cyclistic annual memberships?
- How can Cyclistic use digital media to influence casual riders to become members?

Business task (Goal)

“Analyzing the difference in usage patterns of casual riders and annual members with the aim to convert casual riders into annual members”

Data

We will use Cyclistic’s historical trip data to analyze and identify trends. The datasets are appropriate and will enable you to answer the business questions.

privacy

This is public data that can use to explore how different customer types are using Cyclistic bikes. But note that data-privacy issues prohibit you from using riders’ personally identifiable information. This means that you won’t be able to connect pass purchases to credit card numbers to determine if casual riders live in the Cyclistic service area or if they have purchased multiple single passes.

Data Location

data was loaded from [https://divvy-tripdata.s3.amazonaws.com/index.html (https://divvy-tripdata.s3.amazonaws.com/index.html)](https://divvy-tripdata.s3.amazonaws.com/index.html)

Data Organization

Data includes previous 12 month historical trip data from April 2020 to March 2021 with one .csv file for each month Each .csv file is organized in rows and columns structure with 13 Columns and variable rows

Credibility of the data/ Data Bias

The data is credible and free of bias. It comes from a reliable source, it is original trip data, comprehensive and current (last 12 months data).

Licensing, Privacy, Security, and Accessibility

- The data has been made available by Motivate International Inc. under this license.
- The data does not contain any private information of the riders, thereby maintaining their privacy.
- The data stands secure in an AWS web portal.
- The data is open-source and accessible to all.

Tools for the project

Since the combined dataset is very large with 3.8 million rows, Python Pandas has been chosen as the tool for data manipulation, cleaning, aggregation, analysis and visualization.

Tableau has been chosen as the tool for Interactive Dashboard creation.

[https://public.tableau.com/app/profile/nayem.hasan (https://public.tableau.com/app/profile/nayem.hasan)](https://public.tableau.com/app/profile/nayem.hasan)

Preparing phase

1. Importing required pacakages.

In [1]:
```
import pandas as pd     #pandas dataframe
import geopy.distance   #distance of coordinates
import numpy as np      #calculation numeriacal
import glob             #for specific pattern recognition
import matplotlib.pyplot as plt     #for plotting
```

Collecting all files

Here we are working with data from a cyclistic 12 months dataset. all are in same type (.csv) files.

In [2]:
```
all_files=glob.glob(r'C:\Users\mahad\Downloads\capstone_project\project_1\*.csv')
```

In [3]:
```
all_files
```

Out[3]:
```
['C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202004-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202005-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202006-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202007-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202008-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202009-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202010-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202011-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202012-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202101-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202102-divvy-tripda
ta.csv',
 'C:\\Users\\mahad\\Downloads\\capstone_project\\project_1\\202103-divvy-tripda
ta.csv']
```

Concatenating all files into one

In [4]:
```
yearly= pd.concat((pd.read_csv(file)
                  for file in all_files),ignore_index=True)
```

Inspecting files for inconsistancy, null value and data typtes.

In [5]:
```
yearly.describe()
```

Out[5]:

	start_lat	start_lng	end_lat	end_lng
count	3.489748e+06	3.489748e+06	3.485010e+06	3.485010e+06
mean	4.190417e+01	-8.764494e+01	4.190444e+01	-8.764522e+01
std	4.364222e-02	2.575969e-02	4.373705e-02	2.589123e-02
min	4.164000e+01	-8.787000e+01	4.154000e+01	-8.807000e+01
25%	4.188224e+01	-8.765888e+01	4.188266e+01	-8.765917e+01
50%	4.190000e+01	-8.764170e+01	4.190068e+01	-8.764275e+01
75%	4.193000e+01	-8.762773e+01	4.193120e+01	-8.762775e+01
max	4.208000e+01	-8.752000e+01	4.216000e+01	-8.744000e+01

In [6]:
```
yearly.info(verbose=True, show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3489748 entries, 0 to 3489747
Data columns (total 13 columns):
 #   Column              Non-Null Count    Dtype  
---  ------              --------------    -----  
 0   ride_id             3489748 non-null  object 
 1   rideable_type       3489748 non-null  object 
 2   started_at          3489748 non-null  object 
 3   ended_at            3489748 non-null  object 
 4   start_station_name  3367573 non-null  object 
 5   start_station_id    3366947 non-null  object 
 6   end_station_name    3346506 non-null  object 
 7   end_station_id      3346045 non-null  object 
 8   start_lat           3489748 non-null  float64
 9   start_lng           3489748 non-null  float64
 10  end_lat             3485010 non-null  float64
 11  end_lng             3485010 non-null  float64
 12  member_casual       3489748 non-null  object 
dtypes: float64(4), object(9)
memory usage: 346.1+ MB
```

descriptions

- we have 3.4M rows and 14 columns.
- as we see datetime is not readable by pandas
- there are missing values in start_station_name, start_station_id, end_station_name , end_station_id, start_lat,start_lng, end_lat, end_lng
- data types are not defined and not useable.

Type *Markdown* and LaTeX: α^2

In [7]:
```
yearly.columns
```

Out[7]:
```
Index(['ride_id', 'rideable_type', 'started_at', 'ended_at',
       'start_station_name', 'start_station_id', 'end_station_name',
       'end_station_id', 'start_lat', 'start_lng', 'end_lat', 'end_lng',
       'member_casual'],
      dtype='object')
```

Manupulating phase

Data Cleaning : Removing Bad Data and formatting

- dates should be made readable hence we will convert it as panda readable datetime format
- Rides with negative ride_length are considered invalid since the trip start time cannot be greater than the trip end time
- The company's website mentions that rides with ride_length less than 60 seconds are invalid as it was potentially false starts or users trying to re-dock a bike to ensure it was secure. Link to the website
- Rides with ride_length greater than 24 hrs are outliers and hence, invalid
- Rides with NA's in end_lat or end_lng are considered invalid as the rides were not ended in the proper way
- Rides with NA's in station names but with end_lat or end_lng are considered valid rides

We will create a new version of the dataframe since data is being removed.

convert date and time *datetime* readable

In [8]:
```
yearly['started_at'] =  pd.to_datetime(yearly['started_at'], format='%Y-%m-%d %H:
```

In [9]:
```
yearly['ended_at'] =  pd.to_datetime(yearly['ended_at'], format='%Y-%m-%d %H:%M:%
```

In [10]:
```
yearly.dtypes
```

Out[10]:
```
ride_id                       object
rideable_type                 object
started_at            datetime64[ns]
ended_at              datetime64[ns]
start_station_name            object
start_station_id              object
end_station_name              object
end_station_id                object
start_lat                    float64
start_lng                    float64
end_lat                      float64
end_lng                      float64
member_casual                 object
dtype: object
```

Calculate the riding time

In [11]:
```
yearly['riding_time'] = (yearly['ended_at'] - yearly['started_at'])/pd.Timedelta(
```

removing outliers

checking the result. we find some unusual vaules so we inspect them. there are some negetive values and some illogically big vaules which caused by corrupted data we need to fillter them. we took maximum value of 24 hour.

In [12]:
```
print (yearly['riding_time'].max())
print (yearly['riding_time'].min())
```

```
58720.03333333333
-29049.966666666667
```

In [13]:
```
yearly.describe()
```

Out[13]:

	start_lat	start_lng	end_lat	end_lng	riding_time
count	3.489748e+06	3.489748e+06	3.485010e+06	3.485010e+06	3.489748e+06
mean	4.190417e+01	-8.764494e+01	4.190444e+01	-8.764522e+01	2.476664e+01
std	4.364222e-02	2.575969e-02	4.373705e-02	2.589123e-02	3.904216e+02
min	4.164000e+01	-8.787000e+01	4.154000e+01	-8.807000e+01	-2.904997e+04
25%	4.188224e+01	-8.765888e+01	4.188266e+01	-8.765917e+01	7.883333e+00
50%	4.190000e+01	-8.764170e+01	4.190068e+01	-8.764275e+01	1.451667e+01
75%	4.193000e+01	-8.762773e+01	4.193120e+01	-8.762775e+01	2.663333e+01
max	4.208000e+01	-8.752000e+01	4.216000e+01	-8.744000e+01	5.872003e+04

In [14]:
```
yearly = yearly[yearly['riding_time'].between(0,1440)]
```

In [15]:
```
print (yearly['riding_time'].max())
print (yearly['riding_time'].min())
```
```
1439.9
0.0
```

Creating weekday for analysis

In [16]:
```
yearly['day_of_week'] = yearly['started_at'].dt.day_name()
```

In [17]:
```
yearly['day_of_week'].unique()
```
Out[17]:
```
array(['Sunday', 'Friday', 'Wednesday', 'Tuesday', 'Saturday', 'Thursday',
       'Monday'], dtype=object)
```

classifying riding hour

In [18]:
```
yearly['riding_hour'] = yearly['started_at'].dt.hour
```

In [19]:
```
yearly['riding_hour'].unique()
```
Out[19]:
```
array([17, 12, 10, 14, 15, 18, 13,  2, 16,  8, 20, 19, 11, 23,  6,  9,  7,
       22, 21,  1,  5,  0,  4,  3], dtype=int64)
```

In [20]:
```
conditions = [
    (yearly['riding_hour'].between(0,4)),
    (yearly['riding_hour'].between(4,12)),
    (yearly['riding_hour'].between(12,16)),
    (yearly['riding_hour'].between(16,24))
]

values = ['night_ride','morning_ride','afternoon_ride', 'evening_ride']

yearly['riding_hour'] = np.select(conditions, values)
```

In [21]:
```
yearly['riding_hour'].unique()
```
Out[21]:
```
array(['evening_ride', 'morning_ride', 'afternoon_ride', 'night_ride'],
      dtype=object)
```

In [22]:
```
yearly.columns
```
Out[22]:
```
Index(['ride_id', 'rideable_type', 'started_at', 'ended_at',
       'start_station_name', 'start_station_id', 'end_station_name',
       'end_station_id', 'start_lat', 'start_lng', 'end_lat', 'end_lng',
       'member_casual', 'riding_time', 'day_of_week', 'riding_hour'],
      dtype='object')
```

In [23]:
```
yearly
```

start_station_name	start_station_id	end_station_name	end_station_id	start_lat	start_lng	end_lat	end_l
Eckhart Park	86	Lincoln Ave & Diversey Pkwy	152.0	41.896400	-87.661000	41.932200	-87.6586
Drake Ave & Fullerton Ave	503	Kosciuszko Park	499.0	41.924400	-87.715400	41.930600	-87.7238
McClurg Ct & Erie St	142	Indiana Ave & Roosevelt Rd	255.0	41.894500	-87.617900	41.867900	-87.6230
California Ave & Division St	216	Wood St & Augusta Blvd	657.0	41.903000	-87.697500	41.899200	-87.6722
Rush St & Hubbard St	125	Sheridan Rd & Lawrence Ave	323.0	41.890200	-87.626200	41.969500	-87.6547
...	...	...	...	...	...	...	

In [24]:
```
yearly.dtypes
```

Out[24]:
```
ride_id                       object
rideable_type                 object
started_at            datetime64[ns]
ended_at              datetime64[ns]
start_station_name            object
start_station_id              object
end_station_name              object
end_station_id                object
start_lat                    float64
start_lng                    float64
end_lat                      float64
end_lng                      float64
member_casual                 object
riding_time                  float64
day_of_week                   object
riding_hour                   object
dtype: object
```

actually we do not need user ID and Station names we can drop those columns.

In [25]:
```
trip_data=yearly.drop(['ride_id','start_station_name','start_station_id','end_sta
                      'end_station_id'], axis=1)
```

In [26]:
```
#yearly.to_csv('tripdata.csv',index= False)
```

In [27]:
```
trip_data.info(verbose=True, show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3476314 entries, 0 to 3489747
Data columns (total 11 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   rideable_type  3476314 non-null  object
 1   started_at     3476314 non-null  datetime64[ns]
 2   ended_at       3476314 non-null  datetime64[ns]
 3   start_lat      3476314 non-null  float64
 4   start_lng      3476314 non-null  float64
 5   end_lat        3472286 non-null  float64
 6   end_lng        3472286 non-null  float64
 7   member_casual  3476314 non-null  object
 8   riding_time    3476314 non-null  float64
 9   day_of_week    3476314 non-null  object
 10  riding_hour    3476314 non-null  object
dtypes: datetime64[ns](2), float64(5), object(4)
memory usage: 318.3+ MB
```

The final usable data

- The data is clean, accurate, consistent and complete.
- Remove Duplicates : Data does not have any duplicate values.
- Check for Outliers : Outliers have been removed. (trip_length).
- Check for Missing Values: Removed invalid data with missing values (station_names, end_lat).
- Check Data Accuracy: After removing bad data, all the remaining data is within its speculated range and hence accurate..

- Check Data Completeness : All matrices are available to answer the Business Question. prepared some calculations for answering question
  - new trip_length
  - riding_hour
  - week_day
- Check Data Consistency: All 12 months of data have consistent format and structure, thus making it easier to combine into 1 single dataset
- Check Data Relevance: The dataset contains ridership data of past 12 months, thus the data is current and not outdated.
- Check Data Formats: The columns have been typecast correctly and have appropriate data formats.
- Date-Time Format Consistency: All throughout the dataset the date and time are in consistent format.
- Column Names: All the column names are clear and meaningful.
- Overall sense of Data: Given the knowledge of the business, the data makes sense.

In [28]:
```
trip_data
```

Out[28]:

	rideable_type	started_at	ended_at	start_lat	start_lng	end_lat	end_lng	mem
0	docked_bike	2020-04-26 17:45:14	2020-04-26 18:12:03	41.896400	-87.661000	41.932200	-87.658600	
1	docked_bike	2020-04-17 17:08:54	2020-04-17 17:17:03	41.924400	-87.715400	41.930600	-87.723800	
2	docked_bike	2020-04-01 17:54:13	2020-04-01 18:08:36	41.894500	-87.617900	41.867900	-87.623000	
3	docked_bike	2020-04-07 12:50:19	2020-04-07 13:02:31	41.903000	-87.697500	41.899200	-87.672200	
4	docked_bike	2020-04-18 10:22:59	2020-04-18 11:15:54	41.890200	-87.626200	41.969500	-87.654700	
...	...	...	...	...	...	...	...	

Insight phase

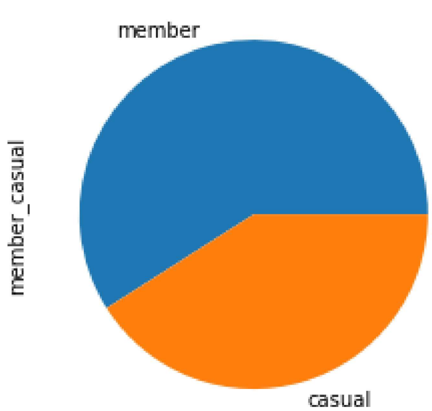
Answering Questions and Visualizing

1. Total member vs casual rider ratio

In [29]:
```
print(trip_data['member_casual'].value_counts())
trip_data['member_casual'].value_counts().plot(kind='pie')
```

```
member    2051734
casual    1424580
Name: member_casual, dtype: int64
```

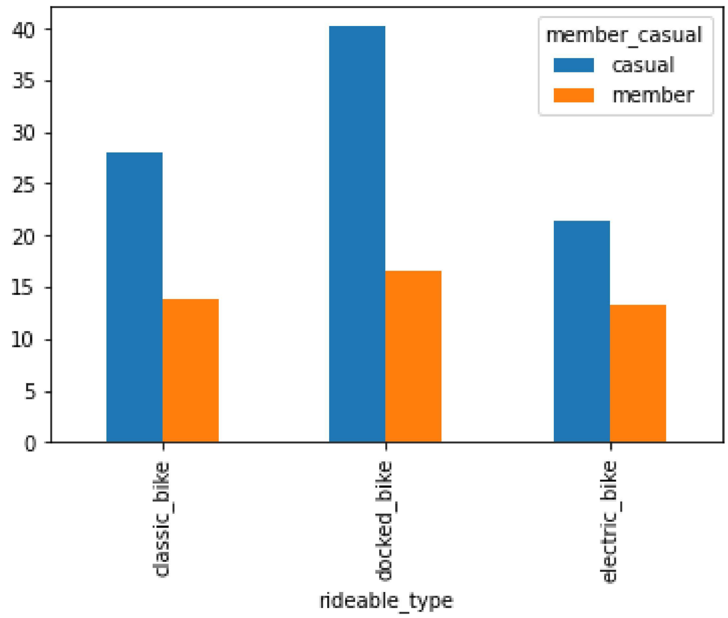
Out[29]: <AxesSubplot:ylabel='member_casual'>



In [40]:
```
print(trip_data.pivot_table(index='rideable_type',values='riding_time',columns='r
trip_data.pivot_table(index='rideable_type',values='riding_time',columns='member_
```

```
member_casual    casual    member
rideable_type
classic_bike      70666    248982
docked_bike     1111001   1434468
electric_bike    242913    368284
```

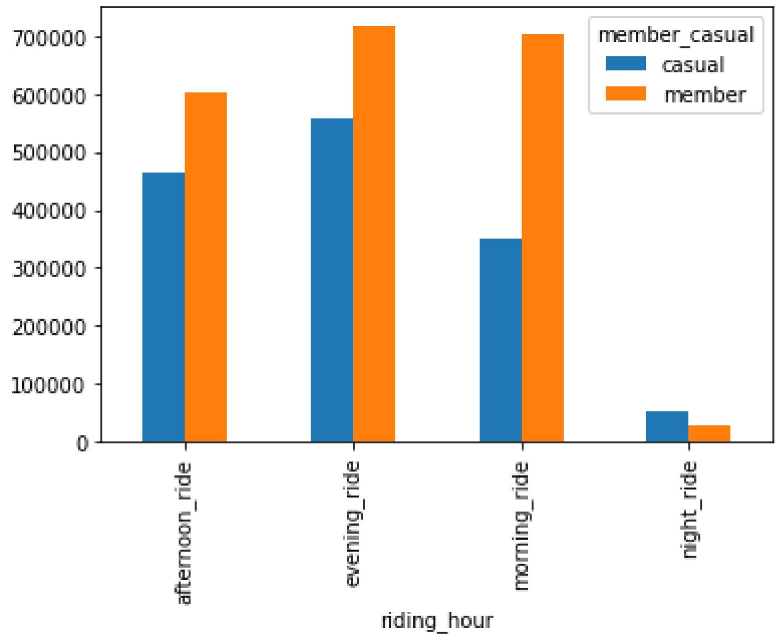
Out[40]: <AxesSubplot:xlabel='rideable_type'>



In [59]:
```
print(yearly.pivot_table(index='riding_hour',values='ride_id',columns='member_cas
yearly.pivot_table(index='riding_hour',values='ride_id',columns='member_casual',a
```

```
member_casual   casual  member
riding_hour
afternoon_ride  465613  602665
evening_ride    556849  717646
morning_ride    350616  702939
night_ride       51502   28484
```

Out[59]: <AxesSubplot:xlabel='riding_hour'>

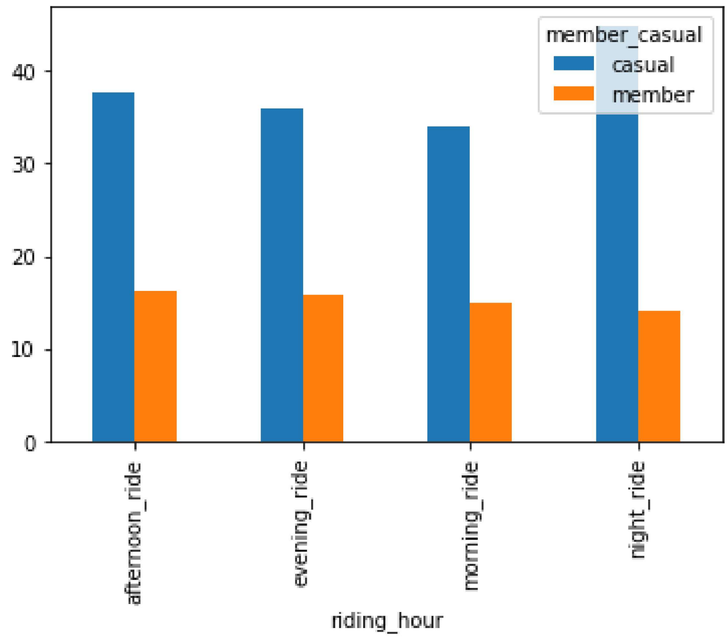


2. Riding time of the day

In [41]:
```
print(trip_data.pivot_table(index='riding_hour',values='riding_time',columns='mem
trip_data.pivot_table(index='riding_hour',values='riding_time',columns='member_ca
```

```
member_casual      casual     member
riding_hour
afternoon_ride  37.691010  16.234165
evening_ride    35.934929  15.849243
morning_ride    34.078059  14.881799
night_ride      44.841748  14.057713
```

Out[41]: <AxesSubplot:xlabel='riding_hour'>

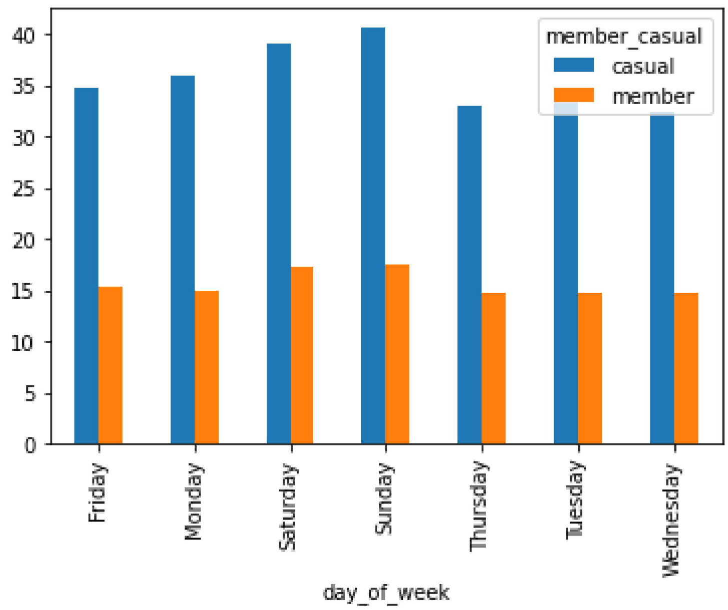


3. Avg weekly riding time

In [32]:
```
print(trip_data.pivot_table(index='day_of_week',values='riding_time',columns='mem
trip_data.pivot_table(index='day_of_week',values='riding_time',columns='member_ca
```

```
member_casual      casual     member
day_of_week
Friday          34.776969  15.384440
Monday          35.890543  14.830327
Saturday        39.040948  17.266623
Sunday          40.618858  17.396474
Thursday        32.991772  14.809095
Tuesday         33.549535  14.715884
Wednesday       32.411688  14.806827
```

Out[32]: <AxesSubplot:xlabel='day_of_week'>

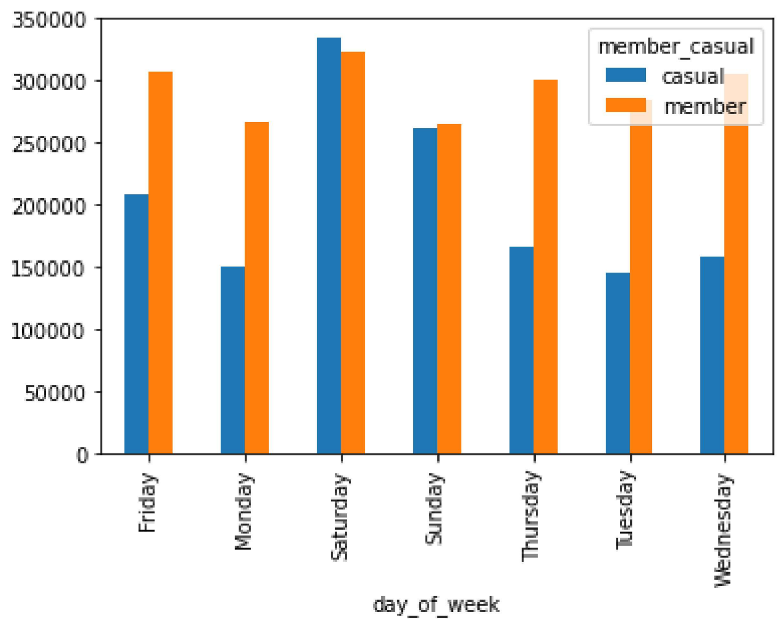


Weekly ride count

In [33]:
```
print(trip_data.pivot_table(index='day_of_week',values='riding_time',columns='mem
trip_data.pivot_table(index='day_of_week',values='riding_time',columns='member_ca
```

```
member_casual  casual  member
day_of_week
Friday         208207  306333
Monday         150843  267292
Saturday       334514  323072
Sunday         261756  265255
Thursday       166092  300407
Tuesday        145038  284325
Wednesday      158130  305050
```

Out[33]: <AxesSubplot:xlabel='day_of_week'>

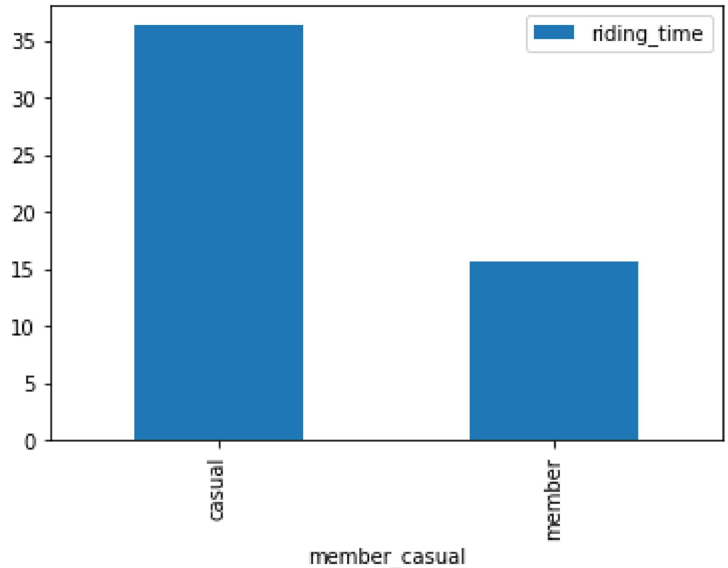


4. Average riding time member vs casual

In [34]:
```
print(trip_data.pivot_table(index='member_casual',values='riding_time'))
trip_data.pivot_table(index='member_casual',values='riding_time').plot(kind='bar'
```

```
               riding_time
member_casual
casual           36.373883
member           15.605983
```

Out[34]: <AxesSubplot:xlabel='member_casual'>



In [35]:
```
#trip_data['date']=trip_data['started_at'].dt.to_period('M')
yearly['date']=trip_data['started_at'].dt.month_name()

#sales_df['Month'] = sales_df['Date'].dt.month_name(locale='English')
```

In [36]:
```
yearly['date'].unique()
```

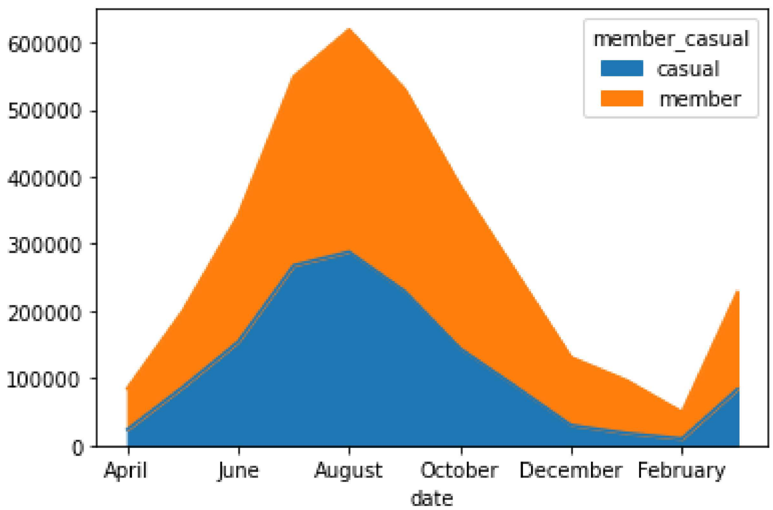
Out[36]:
```
array(['April', 'May', 'June', 'July', 'August', 'September', 'October',
       'November', 'December', 'January', 'February', 'March'],
      dtype=object)
```

5. User in every month of the year

In [37]:
```
print(yearly.pivot_table(index='date',sort=False,values= 'ride_id',columns='membe
yearly.pivot_table(index='date',sort=False,values= 'ride_id',columns='member_casu
#plt.rcParams["figure.figsize"] = (10, 5)
```

```
member_casual  casual  member
date
April           23507   61095
May             86666  113236
June           154216  187963
July           268021  281003
August         288183  330914
September      229800  300715
October        144368  242169
November        87820  170920
December        29956  101130
January         18090   78705
February        10073   39432
March           83880  144452
```

Out[37]: <AxesSubplot:xlabel='date'>



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Dashboard and Report avilable in tableau public

https://public.tableau.com/app/profile/nayem.hasan
(https://public.tableau.com/app/profile/nayem.hasan)

In [ ]: