Part I - Ford GoBike Dataset Exploration

by Darragh Merrick

Introduction

Dataset Overview and Notes

This data set includes information about individual rides made in a bike-sharing system covering the greater San Francisco Bay area.

Note that this dataset will require some data wrangling in order to make it tidy for analysis. There are multiple cities covered by the linked system, and multiple data files will need to be joined together if a full year's coverage is desired. If you're feeling adventurous, try adding in analysis from other cities, following links from this page.

Example Topics/Questions

When are most trips taken in terms of time of day, day of the week, or month of the year?

How long does the average trip take?

Does the above depend on if a user is a subscriber or customer?

> **Rubric Tip**: Your code should not generate any errors, and should use functions, loops where possible to reduce repetitive code. Prefer to use functions to reuse code statements.

> **Rubric Tip**: Document your approach and findings in markdown cells. Use comments and docstrings in code cells to document the code functionality.
>
> **Rubric Tip**: Markup cells should have headers and text that organize your thoughts, findings, and what you plan on investigating next.

Preliminary Wrangling

In [1]:

```
# import all packages and set plots to be embedded inline
from requests import get
from zipfile import ZipFile
from io import StringIO, BytesIO
import numpy as np
import pandas as pd
import missingno as ms
import matplotlib.pyplot as plt
import seaborn as sb
import geopandas as gpd
import folium
from shapely.geometry import Point, Polygon
import haversine as hs


%matplotlib inline
```

> Load in your dataset and describe its properties through the questions below. Try and motivate your exploration goals through this section.

In [2]:

```
#Download the fordgobike data from the Udacity provided link
url = 'https://video.udacity-data.com/topher/2020/October/5f91cf38_201902-
data_csv = get(url)
data_csv
#<Response [200]> = Success
```

Out[2]: <Response [200]>

In [3]:

```
#Store the downloaded data in a csv file and verify it
df = pd.read_csv(StringIO(data_csv.content.decode('utf-8')))
df.head()
```

Out[3]:

	duration_sec	start_time	end_time	start_station_id	start_station_name	start_station_l
0	52185	2019-02-28 17:32:10.1450	2019-03-01 08:01:55.9750	21.0	Montgomery St BART Station (Market St at 2nd St)	37.
1	42521	2019-02-28 18:53:21.7890	2019-03-01 06:42:03.0560	23.0	The Embarcadero at Steuart St	37.
2	61854	2019-02-28 12:13:13.2180	2019-03-01 05:24:08.1460	86.0	Market St at Dolores St	37.
3	36490	2019-02-28 17:54:26.0100	2019-03-01 04:02:36.8420	375.0	Grove St at Masonic Ave	37.

duration_sec	start_time	end_time	start_station_id	start_station_name	start_station_l

In [4]:
```
#Check the size of the data (rows,columns)
df.shape
```

Out[4]:
```
(183412, 16)
```

In [5]:
```
#Investigate the column info to see are there null values and incorrect da
df.info(show_counts = True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 183412 entries, 0 to 183411
Data columns (total 16 columns):
 #   Column                   Non-Null Count   Dtype  
---  ------                   --------------   -----  
 0   duration_sec             183412 non-null  int64  
 1   start_time               183412 non-null  object 
 2   end_time                 183412 non-null  object 
 3   start_station_id         183215 non-null  float64
 4   start_station_name       183215 non-null  object 
 5   start_station_latitude   183412 non-null  float64
 6   start_station_longitude  183412 non-null  float64
 7   end_station_id           183215 non-null  float64
 8   end_station_name         183215 non-null  object 
 9   end_station_latitude     183412 non-null  float64
 10  end_station_longitude    183412 non-null  float64
 11  bike_id                  183412 non-null  int64  
 12  user_type                183412 non-null  object 
 13  member_birth_year        175147 non-null  float64
 14  member_gender            175147 non-null  object 
 15  bike_share_for_all_trip  183412 non-null  object 
dtypes: float64(7), int64(2), object(7)
memory usage: 22.4+ MB
```

In [6]:
```
#Show the sum of null entries for each column
df.isna().sum()
```

Out[6]:
```
duration_sec                  0
start_time                    0
end_time                      0
start_station_id            197
start_station_name          197
start_station_latitude        0
start_station_longitude       0
end_station_id              197
end_station_name            197
end_station_latitude          0
end_station_longitude         0
bike_id                       0
user_type                     0
member_birth_year          8265
member_gender              8265
bike_share_for_all_trip       0
dtype: int64
```

In [7]:
```
# describe the data to generate descriptive statistics
df.describe()
```

Out[7]:

	duration_sec	start_station_id	start_station_latitude	start_station_longitude	end_station_

	duration_sec	start_station_id	start_station_latitude	start_station_longitude	end_station
count	183412.000000	183215.000000	183412.000000	183412.000000	183215.0000
mean	726.078435	138.590427	37.771223	-122.352664	136.249
std	1794.389780	111.778864	0.099581	0.117097	111.515
min	61.000000	3.000000	37.317298	-122.453704	3.0000
25%	325.000000	47.000000	37.770083	-122.412408	44.0000
50%	514.000000	104.000000	37.780760	-122.398285	100.0000
75%	796.000000	239.000000	37.797280	-122.286533	235.0000

In [8]:

```
#Verify if the data contains duplicate rows
df.duplicated().sum()
```

Out[8]: 0

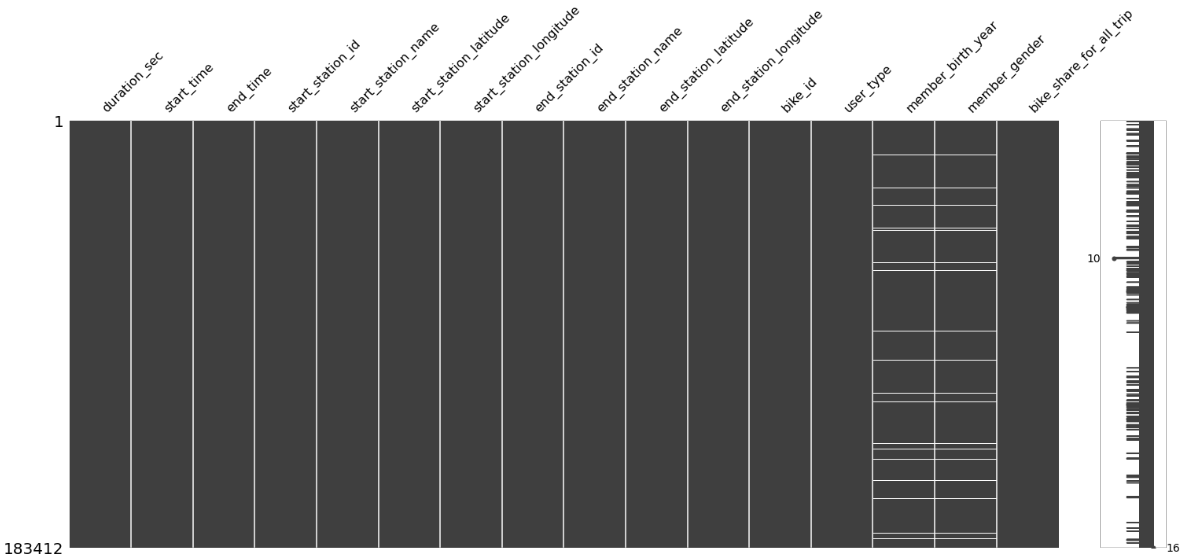
Data Cleaning

Missing data

Visualise missing data in the dataset

In [9]:

```
#Visualize missing data with missingno
ms.matrix(df);
```



There are quality and tidiness issues in the data, that will need to be addressed. The datatypes of multiple columns will need to be changed to gain insights such as:

- start_time object to datetime64
- end_time object to datetime64
- start_station_id float64 to object
- end_station_id float64 to object
- start_station_latitude float64
- start_station_longitude float64
- end_station_latitude float64

- end_station_longitude float64
- bike_id int64 to object
- member_birth_year float64 to int64

There are missing values in:

- start_station_id 197
- start_station_name 197
- end_station_id 197
- end_station_name 197
- member_birth_year 8265
- member_gender 8265

There are also invalid birth year values

There are other elements I could add like distance between start and stop, which would add useful information.

I would like to visualize the start and end locations on a map, which could also add useful information to this data study, even though it's not covered on this course.

In [10]:

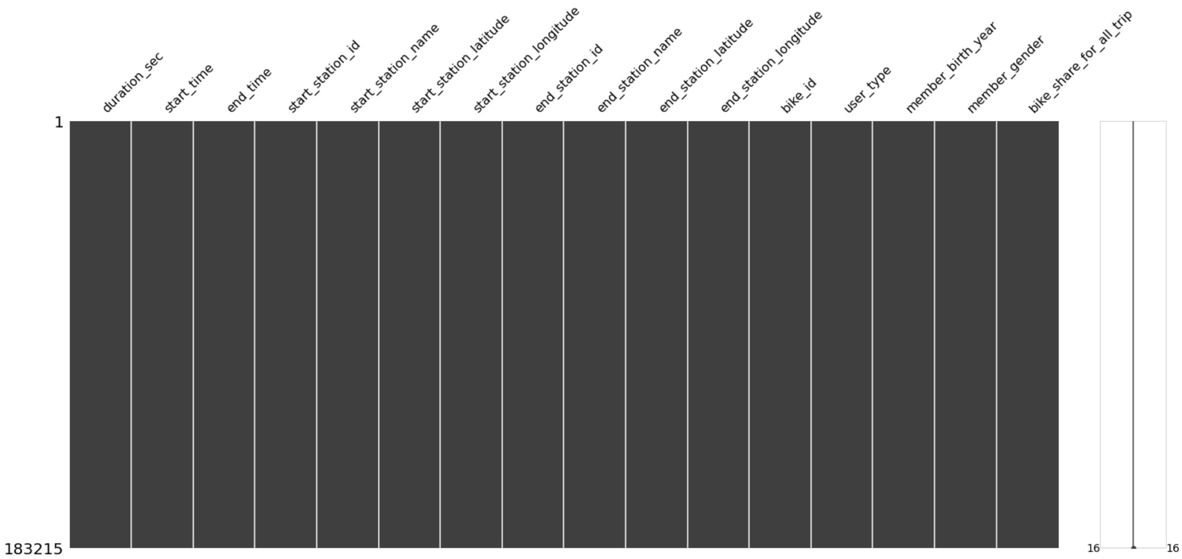
```
#Remove missing values.
df.dropna(subset = ["start_station_id"], inplace = True)
#Replace NULL values with 0
df.member_birth_year.fillna(0, inplace = True)
#Replace NULL values with 'not defined'
df.member_gender.fillna("not defined", inplace = True)
```

In [11]:

```
#Convert column to the correct dtypes
df['start_time'] = pd.to_datetime(df['start_time'], format = "%Y-%m-%d ")
df['end_time'] = pd.to_datetime(df['end_time'], format = "%Y-%m-%d ")
df['duration_sec'] = df['duration_sec'].astype(int)
df['start_station_id'] = df['start_station_id'].astype(str)
df['end_station_id'] = df['end_station_id'].astype(str)
df['start_station_latitude'] = df['start_station_latitude'].astype(float)
df['start_station_longitude'] = df['start_station_longitude'].astype(float
df['end_station_latitude'] = df['end_station_latitude'].astype(float)
df['end_station_longitude'] = df['end_station_longitude'].astype(float)
df['start_time'] = pd.to_datetime(df['start_time'], format = "%Y-%m-%d ")
df['bike_id'] = df['bike_id'].astype(str)
df['member_birth_year'] = df['member_birth_year'].astype(int)
```

In [12]:

```
#Verify data cleaning
ms.matrix(df);
```



What is the structure of your dataset?

> The dataset has 183412 rows and 16 columns.

Exploratory Data Visualizations

In [13]:

```
#Make a new dataframe for geocode data analysis
geo_df = pd.DataFrame()
geo_df['start_lat'] = df['start_station_latitude']
geo_df['start_long'] = df['start_station_longitude']
geo_df['end_lat'] = df['end_station_latitude']
geo_df['end_long'] = df['end_station_longitude']
geo_df['duration_sec'] = df['duration_sec']
geo_df['date'] = df['start_time']
geo_df['member_birth_year'] =  df['member_birth_year']
geo_df['member_gender'] = df['member_gender']
geo_df = geo_df.astype(str)
i=0
rows = geo_df.shape[0]

geo_df.dtypes
geo_df['trips'] = "Start Location: " + ',' + geo_df['start_lat'].map(str) +
#trip_counts shows counts of the most popular journeys
trip_counts = geo_df['trips'].value_counts()
print(trip_counts.head(5))
```

```
Start Location: ,37.77588,-122.39317,End Location: ,37.795392,-122.394203
337
Start Location: ,37.795392,-122.394203,End Location: ,37.80477,-122.403234
314
Start Location: ,37.80889393398715,-122.25646018981932,End Location: ,37.80
90126,-122.2682473    310
Start Location: ,37.80477,-122.403234,End Location: ,37.79413,-122.39443
285
Start Location: ,37.8090126,-122.2682473,End Location: ,37.80889393398715,-
122.25646018981932    284
Name: trips, dtype: int64
```

In [14]:

```
top_trips = trip_counts.head(10)
top_trips.head(5)
```

Out[14]:
```
Start Location: ,37.77588,-122.39317,End Location: ,37.795392,-122.394203
337
Start Location: ,37.795392,-122.394203,End Location: ,37.80477,-122.403234
314
Start Location: ,37.80889393398715,-122.25646018981932,End Location: ,37.80
90126,-122.2682473    310
Start Location: ,37.80477,-122.403234,End Location: ,37.79413,-122.39443
285
Start Location: ,37.8090126,-122.2682473,End Location: ,37.80889393398715,-
122.25646018981932    284
Name: trips, dtype: int64
```

In [15]:
```
#Show most popular journeys as a percentage
trip_counts_percentage = geo_df['trips'].value_counts(normalize=True)
print("Key : {} , Value : {}".format(trip_counts_percentage.index[0],trip_(
```

```
Key : Start Location: ,37.77588,-122.39317,End Location: ,37.795392,-122.39
4203 , Value : 0.0018393690472941625
```

In [16]:
```
print(trip_counts.index[0])
```

```
Start Location: ,37.77588,-122.39317,End Location: ,37.795392,-122.394203
```

In [17]:
```
print(trip_counts.values)
```

```
[337 314 310 ...   1   1   1]
```

In [18]:
```
#Get some metrics from the trip_counts data
print(len(trip_counts))
print(trip_counts.median())
print(trip_counts.min())
print(trip_counts.max())
```

```
23648
3.0
1
337
```

In [19]:
```
#Pie Chart to visualize the top journeys
values = top_trips.values
labels = top_trips.index
explode = (0.2, 0, 0, 0, 0, 0, 0, 0, 0, 0)
plt.pie(values, labels= values,explode=explode,counterclock=False, shadow=T
plt.title('Top 10 Bike Journeys')
plt.legend(labels, loc='center left', bbox_to_anchor=(1.5, 0.5))
plt.show()
```

In [20]:
```
#top 500 is the most the jupyter notebook can handle without crashing or b
n = 500
top_trips = geo_df['trips'].value_counts().index.tolist()[:n]
#print(top_trips)
```

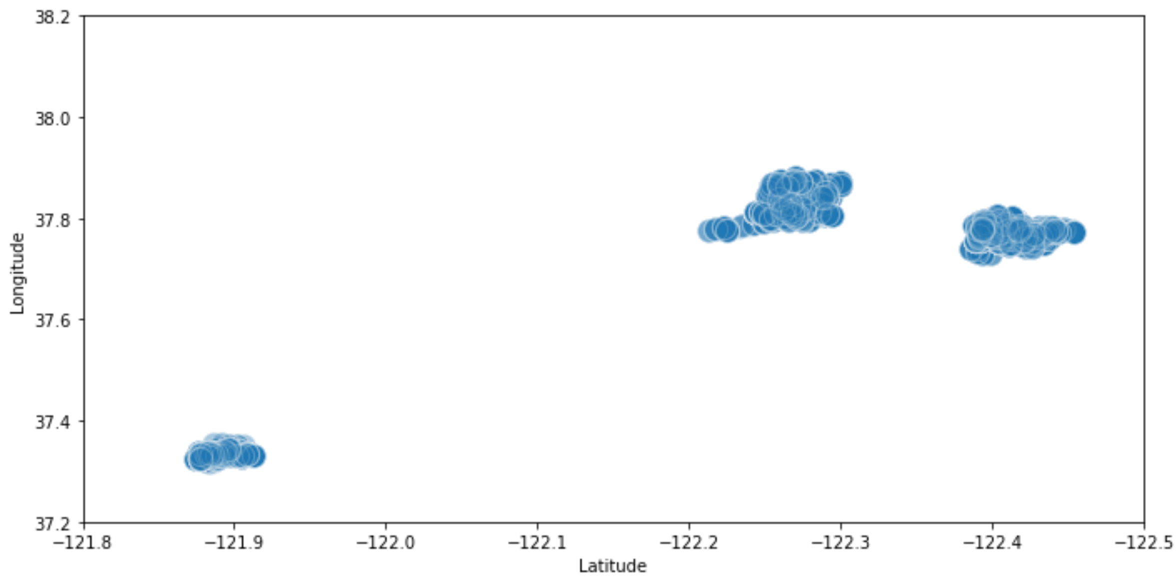
In [21]:
```
print(geo_df.shape[0])
```

183215

In [22]:
```
#Use Folium to plot the journeys on a map
#Green markers are start points, red are finish points
#Hover the marker to get the journey number
my_map = folium.Map(location=(37.7693053,-122.4268256), zoom_start=11);
i=0
limit = len(top_trips)
while i < limit:
    trip = top_trips[i].split(',')
    #print(trip)
    #print('Trip: ',i,'Start LatLong: ',trip[0],trip[1],'End LatLong: ',tr
    folium.Marker(location=(trip[1],trip[2]),popup='Start trip:'+str(i),ic
    folium.Marker(location=(trip[4],trip[5]),popup='End trip:'+str(i),icon=
    i+=1

display(my_map)
```

Leaflet (https://leafletjs.com) | Data by © OpenStreetMap (http://openstreetmap.org), under ODbL (http://www.openstreetmap.org/copyright).

In [23]:
```
#Use Seaborn to plot the jouney in a scatterplot
axes, figure = plt.subplots(figsize = (10,5))
sb.scatterplot(data = df[df.start_station_id.isnull()], x = "end_station_l
sb.scatterplot(data = df.dropna(subset=["end_station_id"]).sample(50000), x
plt.xlim(-121.8,-122.5)
plt.ylim(37.2,38.2)
plt.xlabel("Latitude");
plt.ylabel("Longitude");
plt.tight_layout()
```



Formula to calculate distance between 2 locations

$$d = 2r \arcsin\left(\sqrt{\operatorname{haversin}(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\operatorname{haversin}(\lambda_2 - \lambda_1)}\right)$$
$$= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Distance Based on Latitude and Longitude

Use this calculator to find the shortest distance (great circle/air distance) between two points on the Earth's surface.

Result

The distance between [37.7896254, -122.400811] and [37.794231, -122.402923] is:
0.5440 km or 0.3380 mile

Decimal | Degree-Minute-Second

	Latitude 1	Longitude 1
Point 1:	37.7896254	-122.400811
	Latitude 2	**Longitude 2**
Point 2:	37.794231	-122.402923

Calculate Clear

Tested results against online calculator https://www.calculator.net/distance-calculator.html

In [24]:

```
#https://stackoverflow.com/questions/29545704/fast-haversine-approximation
#this function teakes start Lat Long and Dest Lat long and caculates the d
from math import radians, cos, sin, asin, sqrt
from time import time
#start time
t0 = time()
print('This calculation is slow (minutes), please be patient . . .')
def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    km = 6367 * c
    return round(km,2)

i=0
for index, row in df.iterrows():
    trip = geo_df['trips'].iloc[i].split(',')
    #print(trip[1],trip[2], trip[4], trip[5])
    geo_df.loc[index, 'distances'] = haversine(float(trip[1]), float(trip[
    i+=1

#End time
print("Time taken: {} seconds\nfor {} observations".format(time()-t0, len(
```

```
This calculation is slow (minutes), please be patient . . .
Time taken: 180.83300828933716 seconds
for 183215 observations
```

In [25]:

```
#verify the distance calculations
print(geo_df['distances'].head(5))
```

```
0    0.36
1    0.96
2    2.64
3    0.27
4    2.65
Name: distances, dtype: float64
```

In [26]:

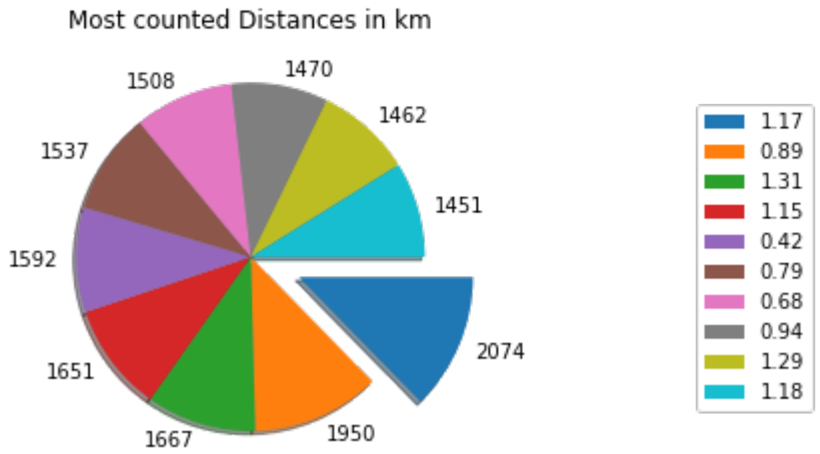
```
#Replace 0 values with NaN, so as not to be counted
geo_df['distances'].replace(0, np.nan, inplace=True)
distance_counts = geo_df['distances'].value_counts(dropna=True)
#Add the distances column to the main dataframe
df['distances'] = geo_df['distances']
```

In [27]:

```
#function to plot a piechart
def pie_Chart(values, labels, title, explode):
    top_dist = distance_counts.head(10)
    explode = explode
    plt.pie(values, labels= values,explode=explode,counterclock=False, shad
    plt.title(title)
    plt.legend(labels, loc='center left', bbox_to_anchor=(1.5, 0.5))
    plt.show()
```

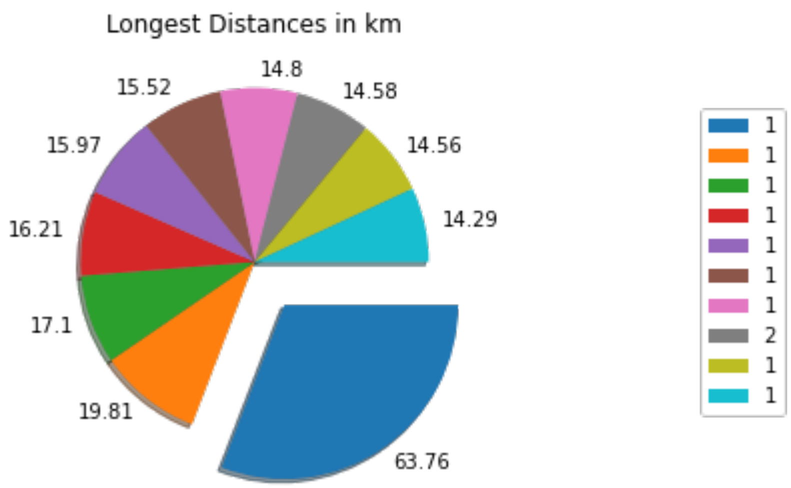
In [28]:

```
#plot piechart of Most counted Distances in km using function
top_dist = distance_counts.head(10)
values = top_dist.values
labels = top_dist.index
title = 'Most counted Distances in km'
explode = (0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0)

pie_Chart(values, labels, title, explode)
```



In [29]:

```
#plot piechart of Longest Distances in km using function
dist = geo_df['distances'].value_counts()
sorted_dist = dist.sort_index(ascending=False)
top_dist = sorted_dist.head(10)
indexs = top_dist.index
values = top_dist.values
title = 'Longest Distances in km'
explode = (0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0)
pie_Chart(indexs, values, title, explode)
```



In [30]:
```
#Convert columns to numerical and date dtypes, so that I can preform mathe
df['distances'] = df['distances'].astype(float)
df['duration_sec'] = df['duration_sec'].astype(int)
df['member_birth_year'] = df['member_birth_year'].astype(float)
df['date'] = pd.to_datetime(geo_df['date'], format = "%Y-%m-%d ")
df.dtypes
```

Out[30]:
```
duration_sec                        int64
start_time                 datetime64[ns]
end_time                   datetime64[ns]
start_station_id                   object
start_station_name                 object
start_station_latitude            float64
start_station_longitude           float64
end_station_id                     object
end_station_name                   object
end_station_latitude              float64
end_station_longitude             float64
bike_id                            object
user_type                          object
member_birth_year                 float64
member_gender                      object
bike_share_for_all_trip            object
distances                         float64
date                       datetime64[ns]
dtype: object
```

In [31]:
```
#What is the longest distance cycled?
df['distances'].max()
```

Out[31]:
```
63.76
```

In [32]:
```
#Calculate age of users by (data_year - birth_year) and add to the main da
df["age"] = df["member_birth_year"].apply(lambda x: 2019 - int(x))
geo_df["age"] = df["age"]
```

In [33]:
```
#cut month_year, store in geo_df, as data isn't neeeded in main data frame
geo_df['month_year'] = pd.to_datetime(df["start_time"]).dt.to_period('M')
```

In [34]:

```
#cut day_month_year.
geo_df['day_month_year'] = pd.to_datetime(df["start_time"]).dt.to_period('I
```

In [35]:

```
#calculate day of the week from datetime
geo_df["dayofweek"] = df["start_time"].apply(lambda x: x.dayofweek)
```

In [36]:

```
#calculate start and end hour of journey.
geo_df["start_hr"] = df["start_time"].apply(lambda x: x.hour)
geo_df["end_hr"] = df["end_time"].apply(lambda x: x.hour)
```

In [37]:

```
#Create Decade age bins
bins = [x for x in range(10,101, 10)]
df["age_bins"] = pd.cut(df.age, bins = bins, precision = 0, include_lowest=
```

In [38]:

```
#verify data
df[["age", "age_bins"]].head()
```

Out[38]:

	age	age_bins
0	35	(30.0, 40.0]
1	2019	NaN
2	47	(40.0, 50.0]
3	30	(20.0, 30.0]
4	45	(40.0, 50.0]

In [39]:

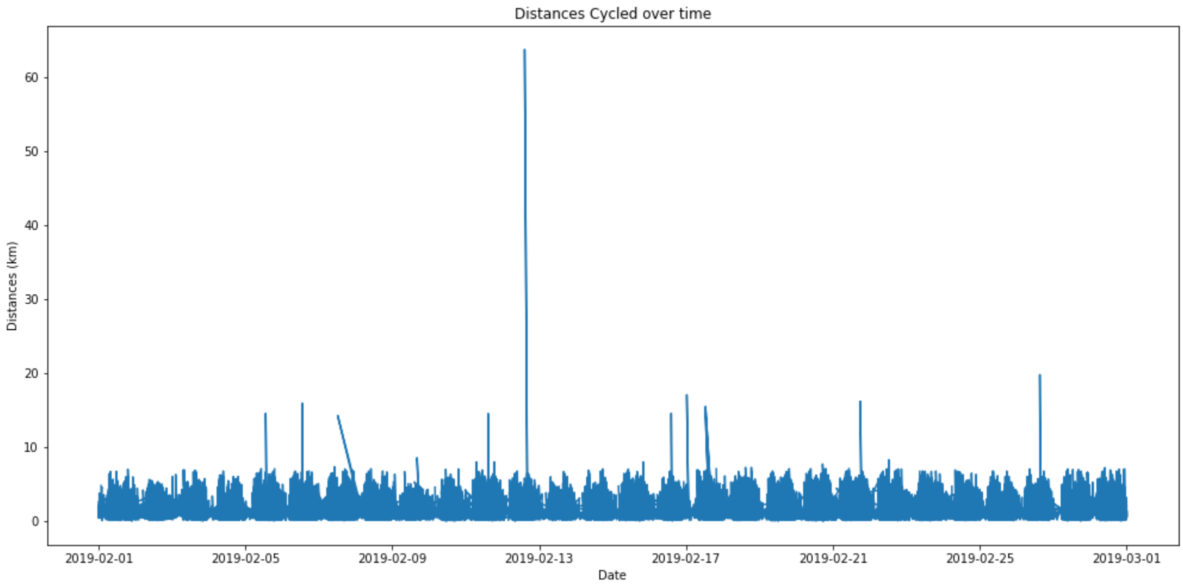
```
#verify data
geo_df[["month_year", "day_month_year","dayofweek","start_hr","end_hr"]].he
```

Out[39]:

	month_year	day_month_year	dayofweek	start_hr	end_hr
0	2019-02	2019-02-28	3	17	8
1	2019-02	2019-02-28	3	18	6
2	2019-02	2019-02-28	3	12	5
3	2019-02	2019-02-28	3	17	4
4	2019-02	2019-02-28	3	23	0

In [67]:

```
# Plotting time vs. Distances

plt.figure(figsize=(17, 8))

plt.xlabel('Date')
plt.ylabel('Distances (km)')

plt.plot(df['date'],df["distances"])
plt.title('Distances Cycled over time');
```



Looking at the 1 duration of 69.47:

start_lat, long	end lat, long	duration	Distance
37.7896254,-122.400811	37.3172979,-121.884995	6945 seconds (1.93 hours)	69.47km

5km every 10 minutes is an estimated average, which would give me 30km over the hour, so this would not be impossible.

From the map below, it looks like someone cycled from San Francisco to San Jose. This would not be impossible, so not going to filter it out as an outlier.

In [68]:

```
#Plot the largest journey to see was it feasible
my_map = folium.Map(location=(37.7693053,-122.4268256), zoom_start=8);
folium.Marker(location=(37.7896254,-122.400811 ),popup='Start of 69.47km t
folium.Marker(location=(37.3172979,-121.884995),popup='End of 69.47km trip
display(my_map)
```

Univariate Exploration

> In this section, investigate distributions of individual variables. If you see unusual points or outliers, take a deeper look to clean things up and prepare yourself to look at relationships between variables.
>
> **Rubric Tip**: The project (Parts I alone) should have at least 15 visualizations distributed over univariate, bivariate, and multivariate plots to explore many relationships in the data set. Use reasoning to justify the flow of the exploration.
>
> **Rubric Tip**: Use the "Question-Visualization-Observations" framework throughout the exploration. This framework involves **asking a question from the data, creating a visualization to find answers, and then recording observations after each visualisation.**

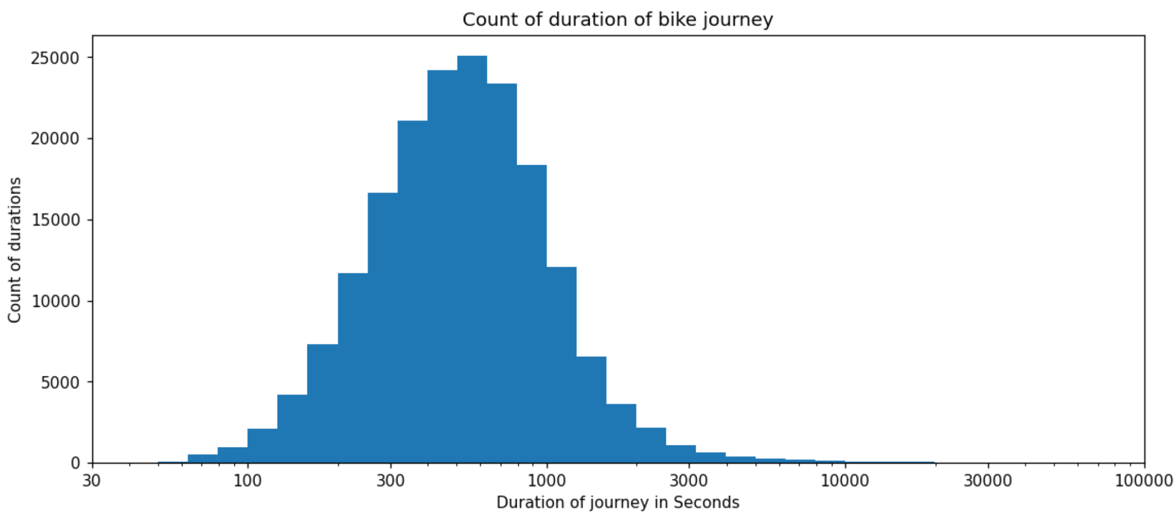
In [69]:

```
#View distances max, min & mean
print(df['distances'].max())
print(df['distances'].min())
print(df['distances'].mean())
```

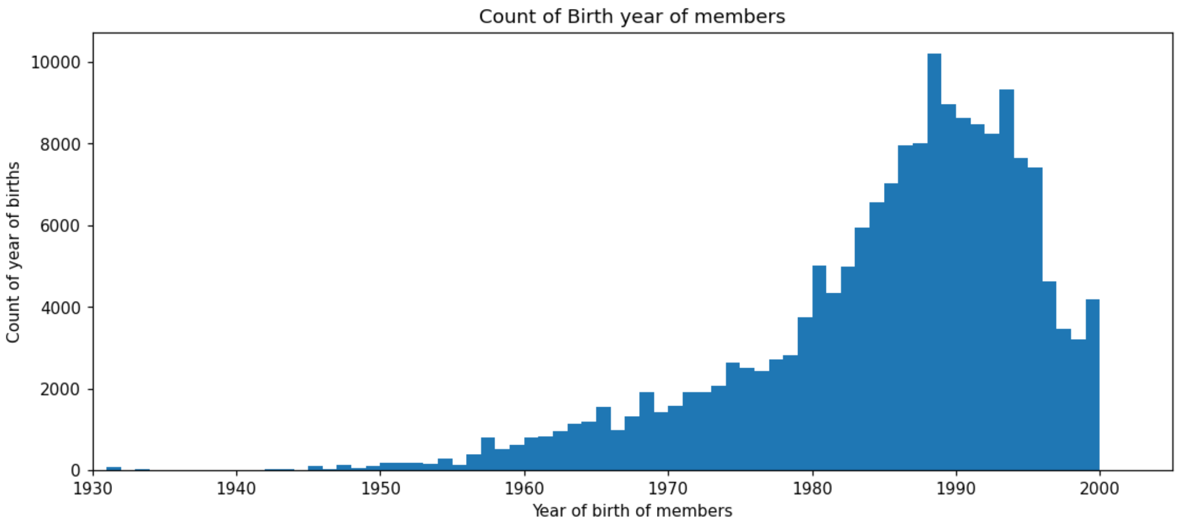
```
63.76
0.02
1.5119350253354733
```

In [70]:

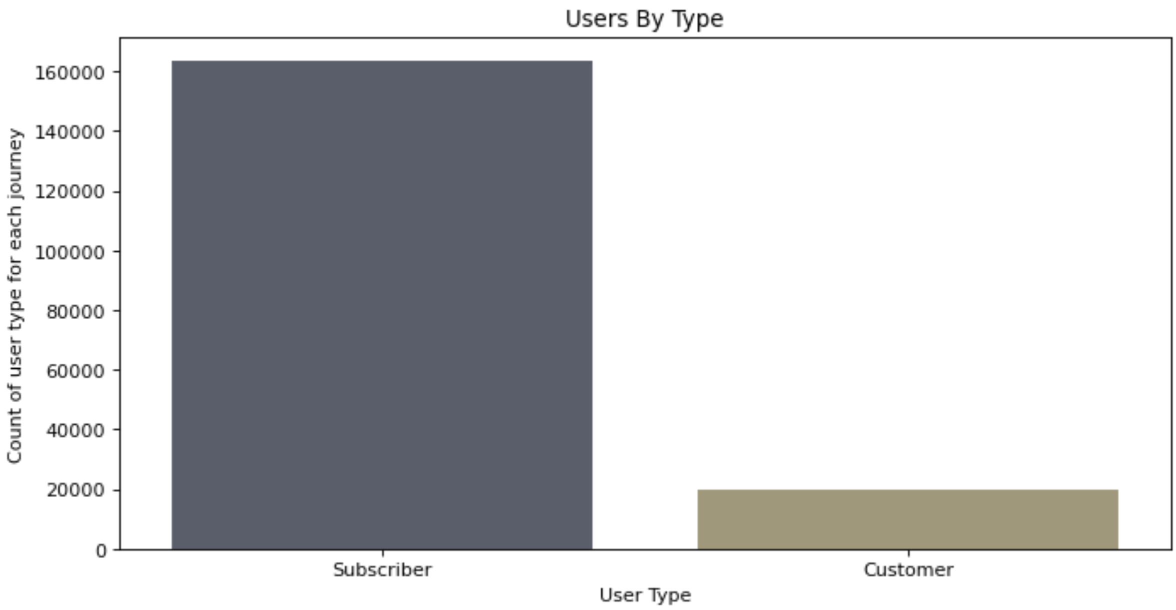
```
#plot count of bike journey durations
#use logarithmic scale
bin_edges = 10 ** np.arange(0, 5, 0.1)
ticks = [30,100,300,1000,3000,10000,30000,100000]
fig, axes = plt.subplots(figsize = (12,5), dpi = 110)
labels = ['{}'.format(v) for v in ticks]
plt.hist(data = df, x ='duration_sec', bins = bin_edges);
plt.xscale("log");
plt.xlim(30, 10000);
plt.xticks(ticks,labels);
plt.title("Count of duration of bike journey");
plt.xlabel("Duration of journey in Seconds");
plt.ylabel("Count of durations");
```



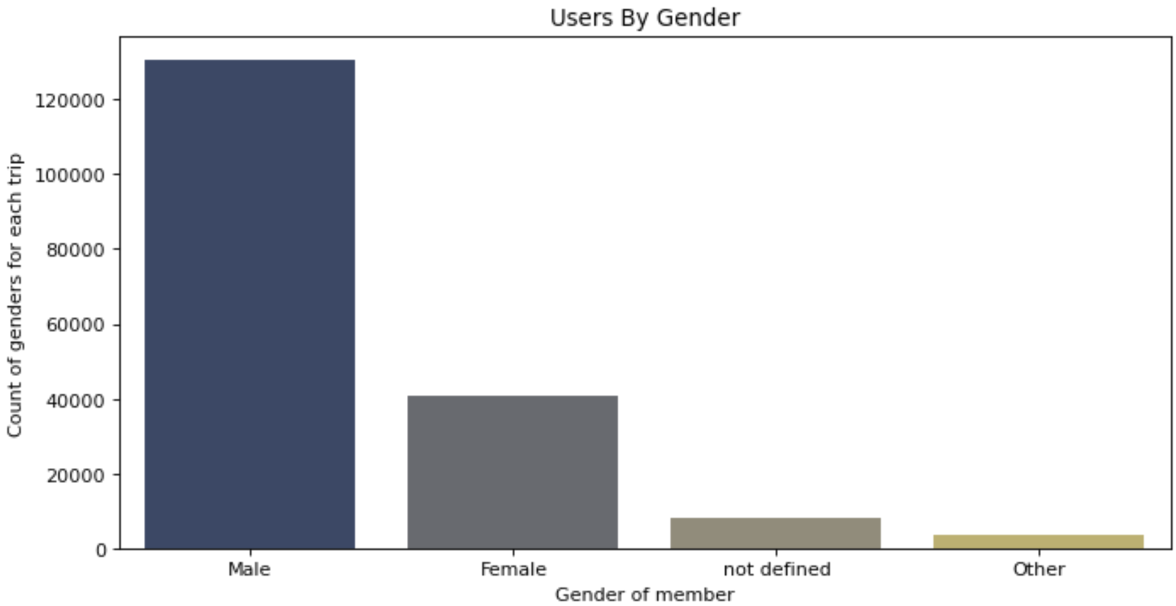
In [71]:
```
#plot count of birth years of members
bin_edges = np.arange(df['member_birth_year'].min(),df['member_birth_year'
fig, axes = plt.subplots(figsize = (12,5), dpi = 110)
plt.hist(data = df, x ='member_birth_year', bins = bin_edges);
plt.xlim(1930, 2005);
plt.title("Count of Birth year of members");
plt.xlabel("Year of birth of members");
plt.ylabel("Count of year of births");
```



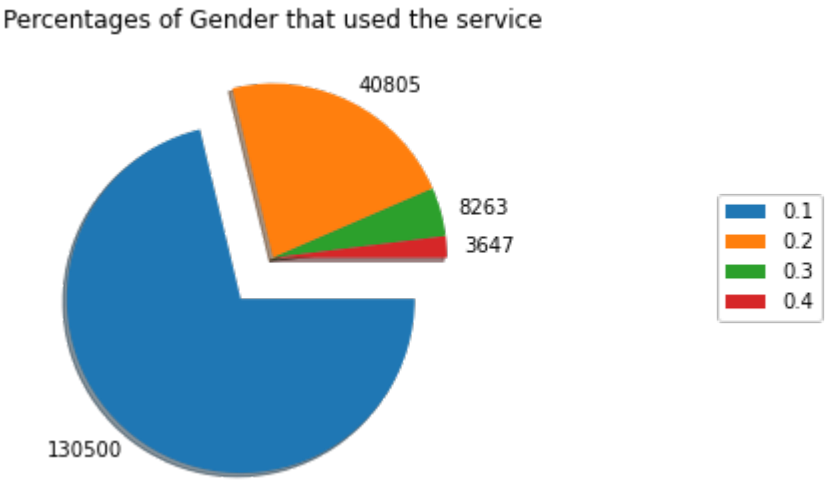
In [79]:
```
#plot count of user types Subscribers Vs Customers
values = df.user_type.value_counts()

fig, ax = plt.subplots(figsize = (10,5), dpi = 80)
sb.countplot(x = "user_type", data = df, order=values.index, palette = "ci

plt.title("Users By Type");
plt.xlabel("User Type");
plt.ylabel("Count of user type for each journey");
```



In [80]:
```
#count trips by gender
fig, ax = plt.subplots(figsize = (10,5), dpi = 80)
sb.countplot(x = "member_gender", data = df,  order=df.member_gender.value_

plt.title("Users By Gender");
plt.xlabel("Gender of member");
plt.ylabel("Count of genders for each trip");
```

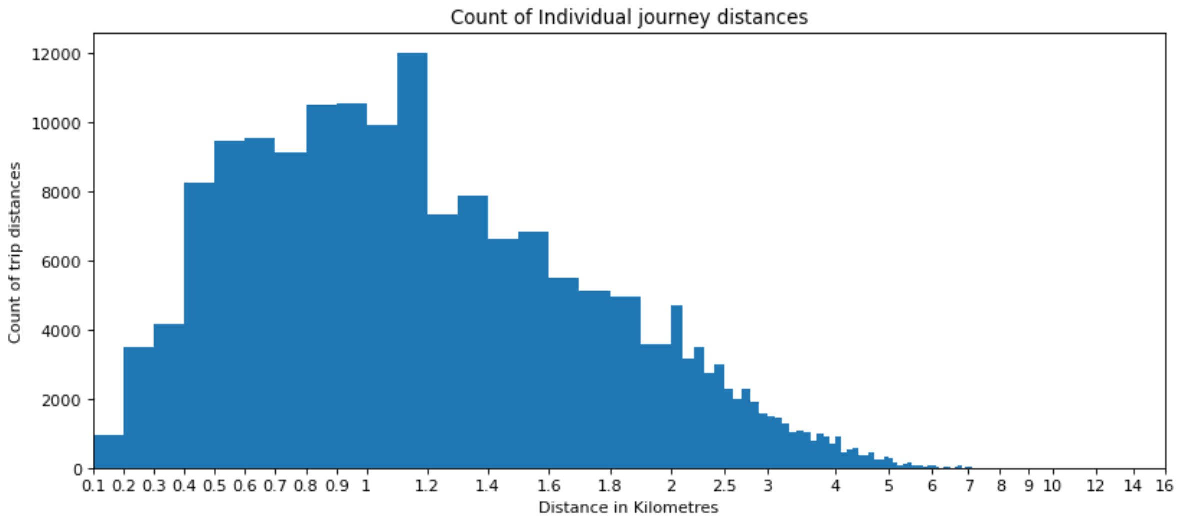


In [81]:
```
#plot piechart of Most counted Distances in km using function

gender = df['member_gender'].value_counts()
sorted_gender = gender.sort_values(ascending=False)
values = sorted_gender.values
indexs = sorted_gender.index
title = 'Percentages of Gender that used the service'
explode = (0.3, 0, 0, 0)

pie_Chart(values, labels, title, explode)
```



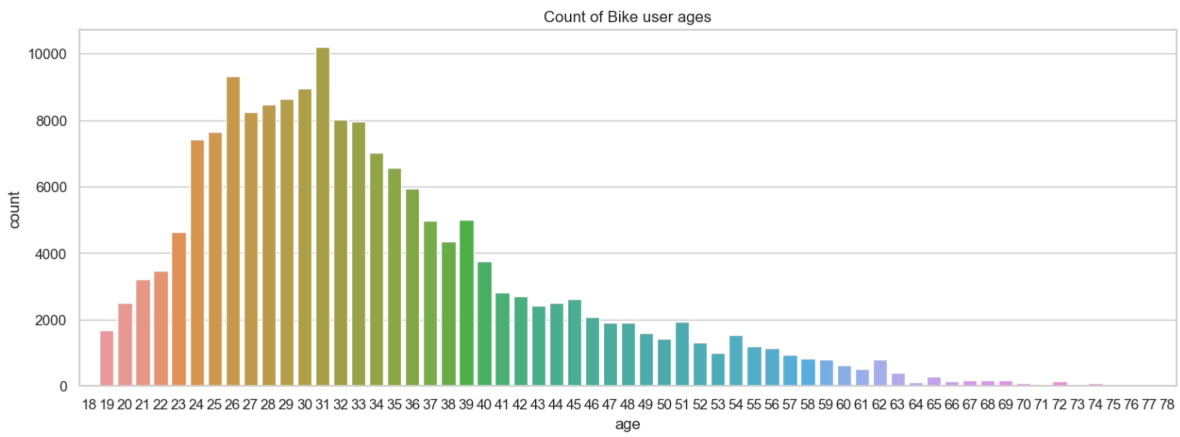
In [82]:

```
#plot histogram to show count of Individual journey distances
#Show the 1st km in 10 divisions for detail
fig, ax = plt.subplots(figsize = (12,5), dpi = 80)
bin_size = 0.1
bin_edges = np.arange(0,geo_df.distances.max()+bin_size,bin_size)
ticks = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1.6,1.8,2,2.5,3,4,5
labels = ['{}'.format(v) for v in ticks]
plt.hist(data = geo_df, x ='distances', bins = bin_edges);
plt.xscale("symlog");
plt.xlim(0.1, 16);
plt.xticks(ticks,labels);
plt.title("Count of Individual journey distances");
plt.xlabel("Distance in Kilometres");
plt.ylabel("Count of trip distances");
```



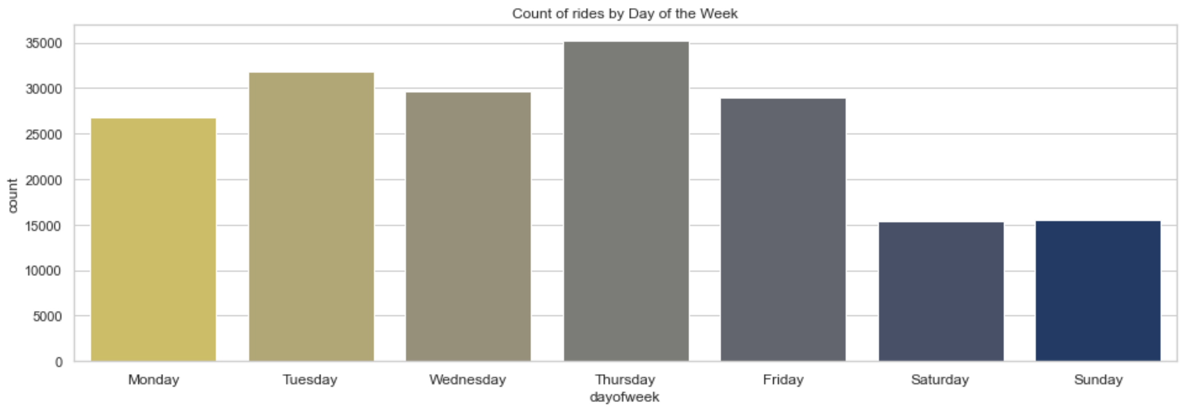
In [147…

```
#plot histogram to show count of ages
fig, ax = plt.subplots(figsize = (15,5), dpi = 100)
color = sb.color_palette("cividis_r")
sb.countplot(x = "age", data = df.query("age < 80").sort_values("age"));
plt.title("Count of Bike user ages");
```



In [146…

```
#Show counts of rides by day of the week
#It can be seen that Thursday is peak and the weekend is quieter
fig, ax = plt.subplots(figsize = (16,5))
sb.countplot(x = "dayofweek", data = geo_df, palette = "cividis_r");
plt.title("Count of rides by Day of the Week");
days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"
plt.xticks(range(len(days)), days, size='medium');
```



What is/are the main feature(s) of interest in your dataset?

> The trip duration and start and end station Lat Longs could generate interesting results. Most popular start stations and end Stations could show interesting trends. Start and end times show year-month-day, so we can find trends of popular times, days, months and seasons. Statistics about gender and age may also show the most popular groups that tend to cycle.

What features in the dataset do you think will help support your investigation into your feature(s) of interest?

> Gender, Age and user type will help profile the customers Start_station_ID and End_Station_id will help find the most popular routes, to assist bike redistribution. Bikes will need to be moved from the most popular destinations to the most popular starting points to keep bikes available at popular starting points. start_time and end_time will help investigate cycle durations and peak times.

> **Rubric Tip**: Visualizations should depict the data appropriately so that the plots are easily interpretable. You should choose an appropriate plot type, data encodings, and formatting as needed. The formatting may include setting/adding the title, labels, legend, and comments. Also, do not overplot or incorrectly plot ordinal data.

Discuss the distribution(s) of your variable(s) of interest. Were there any unusual points? Did you need to perform any transformations?

> In the duration_sec graph, I used a log scale to get an uniform distribution. The

> count of Age and Distances are positively skewed distributions

A distribution is said to be skewed to the right if it has a long tail that trails toward the right side. The skewness value of a positively skewed distribution is greater than zero.

> The count of birth years is a negatively skewed ditribution.

Of the features you investigated, were there any unusual distributions? Did you perform any operations on the data to tidy, adjust, or change the form of the data? If so, why did you do this?

> I didn't find any of the results to be unusual. The most common age groupos were 25 -40 The most common trips were between 0.5 to 2.5 km, although there was one trip from San francisco to San Jose which was 69.47km. I did find it surprising that over 70% of users were male, I would have expected a more even gender usage.

In [87]:

```
#convert distances to float for mathematical calculations
df['distances'] = df['distances'].astype(float)
df.dtypes
```

Out[87]:

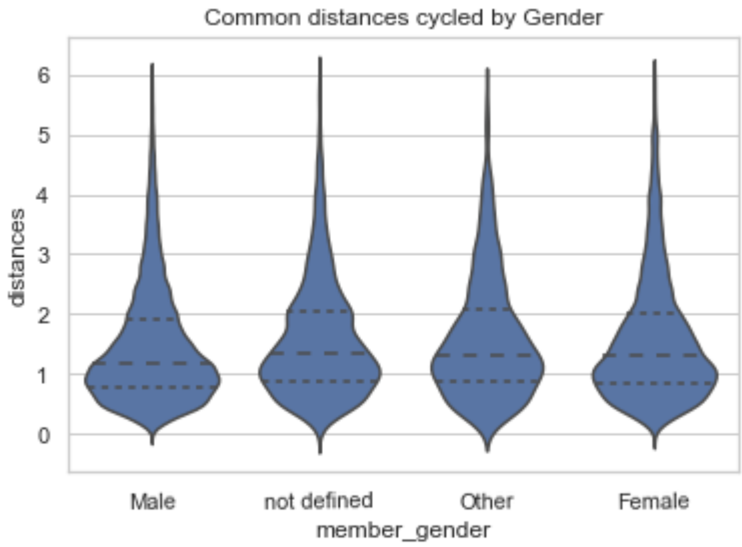
```
duration_sec                          int64
start_time                   datetime64[ns]
end_time                     datetime64[ns]
start_station_id                     object
start_station_name                   object
start_station_latitude              float64
start_station_longitude             float64
end_station_id                       object
end_station_name                     object
end_station_latitude                float64
end_station_longitude               float64
bike_id                              object
user_type                            object
member_birth_year                   float64
member_gender                        object
bike_share_for_all_trip              object
distances                           float64
date                         datetime64[ns]
age                                   int64
age_bins                           category
dtype: object
```

Bivariate Exploration

> In this section, investigate relationships between pairs of variables in your data. Make sure the variables that you cover here have been introduced in some fashion in the previous section (univariate exploration).

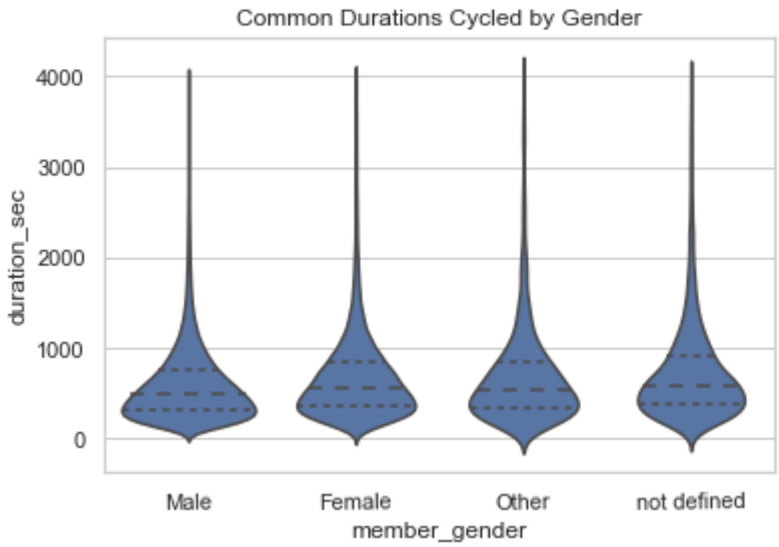
In [145…

```
#show gender count vs distances in violin plot
sb.violinplot(data = df.query("distances <= 6"),
              x = 'member_gender',y='distances',
              color = sb.color_palette()[0],inner='quartile');
plt.xticks(rotation=1);
plt.title("Common distances cycled by Gender");
```



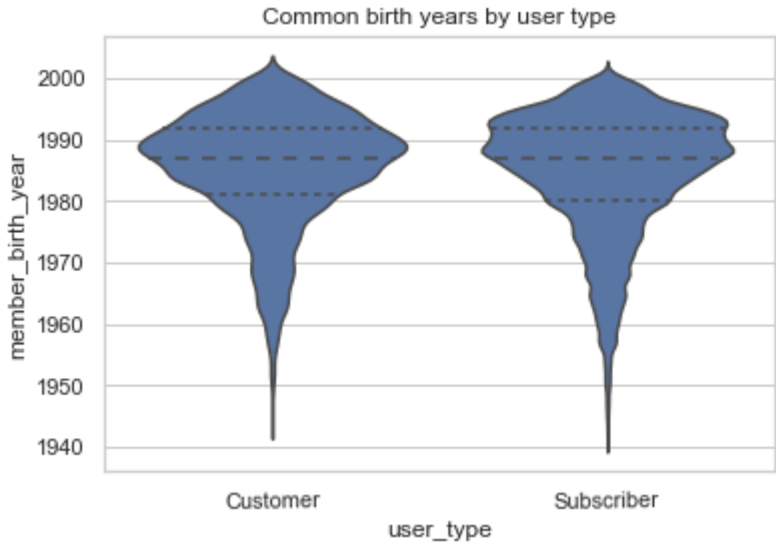
In [144…

```
#show gender count vs distances in violin plot
sb.violinplot(data = df.query("duration_sec <= 4000"),
              x = 'member_gender',y='duration_sec',
              color = sb.color_palette()[0],inner='quartile');
plt.xticks(rotation=1);
plt.title("Common Durations Cycled by Gender");
```



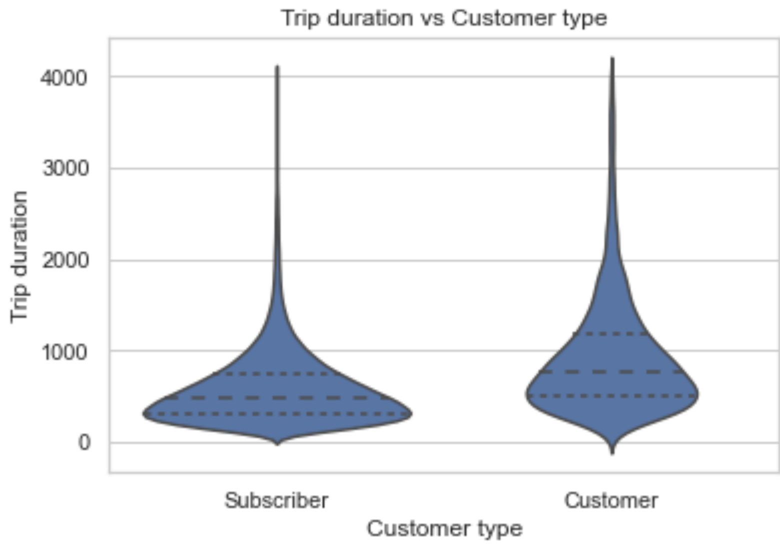
In [143…

```
#show user type vs birth year in violin plot
sb.violinplot(data = df.query("member_birth_year >= 1940"),
              x = 'user_type',y='member_birth_year',
              color = sb.color_palette()[0],inner='quartile');
plt.xticks(rotation=1);
plt.title("Common birth years by user type");
```



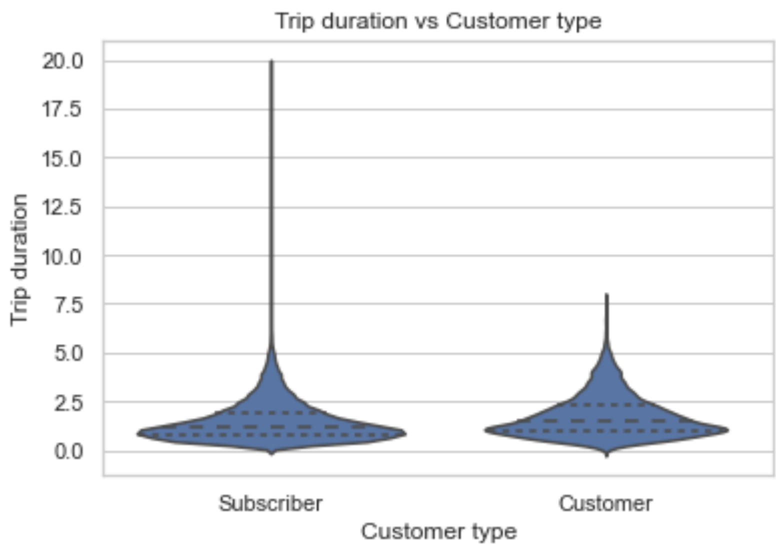
In [142…

```
#show user type vs duration in violin plot
sb.violinplot(data = df.query("duration_sec <= 4039.5"),
              x = 'user_type',y='duration_sec',
              color = sb.color_palette()[0],inner='quartile');
plt.title("Trip duration vs Customer type");
plt.xlabel("Customer type");
plt.ylabel("Trip duration");
```



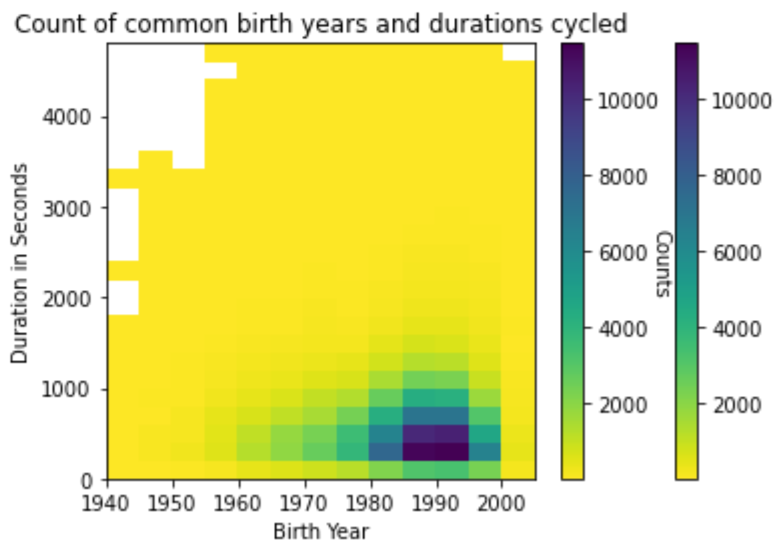
In [141…

```
sb.violinplot(data = df.query("duration_sec <= 4039.5"),x = 'user_type',y=
plt.title("Trip duration vs Customer type");
plt.xlabel("Customer type");
plt.ylabel("Trip duration");
```



In [96]:

```
#plot 2d histogram to show count of births vs duration cycled
xbin = np.arange(df['member_birth_year'].min(), df['member_birth_year'].ma
ybin = np.arange(0, 5000, 200)
plt.hist2d(data = df,x = 'member_birth_year',y='duration_sec',cmin=0.5,cma
plt.xlim(1940, 2005);
plt.colorbar();
plt.colorbar().set_label('Counts', rotation=270);
plt.title("Count of common birth years and durations cycled");
plt.xlabel("Birth Year");
plt.ylabel("Duration in Seconds");
```



Talk about some of the relationships you observed in this part of the investigation. How did the feature(s) of interest vary with other features in the dataset?

- The durations cycled each day do not vary greatly
- The group born between 1985 and 1995 are the most common and cycel for the longest durations.
- Customers tend to cycle longer distances and subscribers tend to cycle shorter distances more often.
- Younger users tend to be subscribers more than customers.
- Durations cycled does not vary greatly between genders.

- Males marginally cycle greater distances

Did you observe any interesting relationships between the other features (not the main feature(s) of interest)?

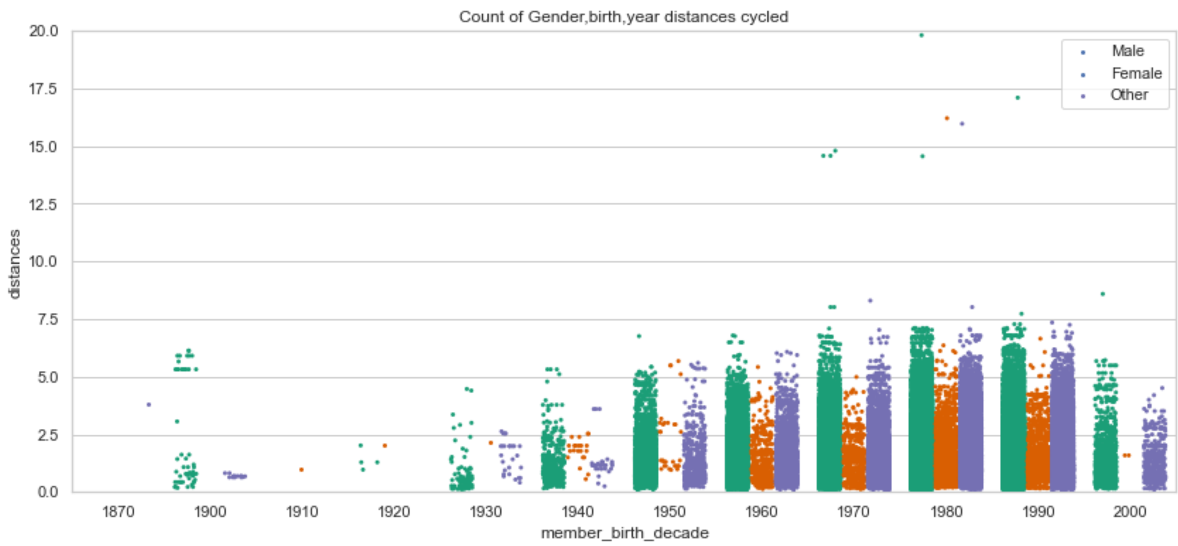
- I found the data to be balanced. The most common users were born around 1985 and 1995 and cycled the farthest. Usage

Multivariate Exploration

> Create plots of three or more variables to investigate your data even further. Make sure that your investigations are justified, and follow from your work in the previous sections.

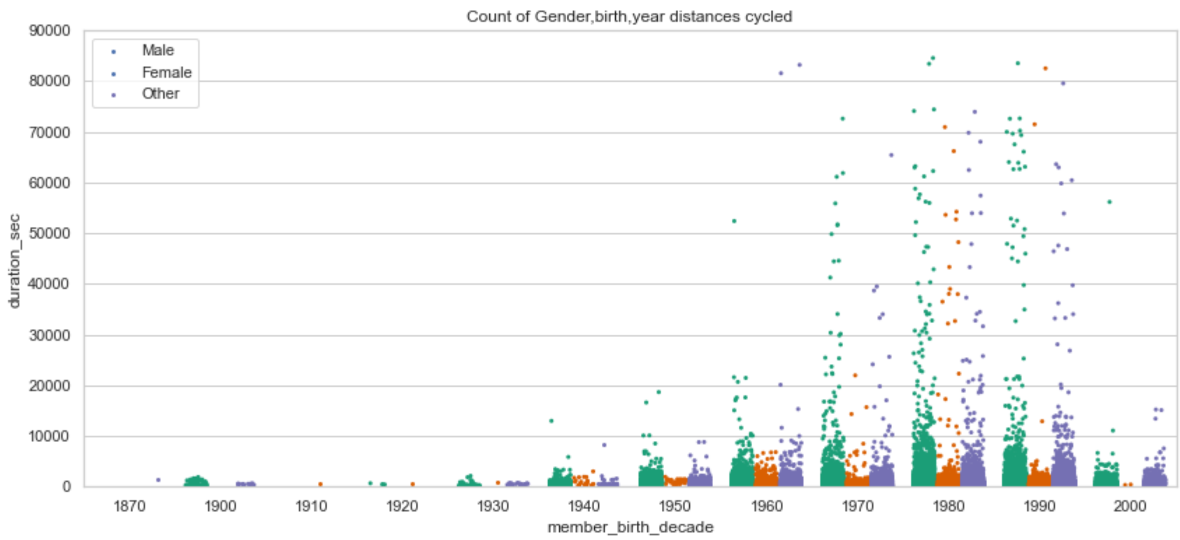
In [150…

```
#set plot dimensions
plt.figure(figsize=[14,6])

#create  a decade variable, using floor division (lowest integer divisor)
df['member_birth_decade'] = ((df['member_birth_year']//10)*10).astype(int)

#use decade
sb.stripplot(data = df.query('member_birth_decade>0'),
             x = 'member_birth_decade', y = 'distances', hue ='member_gend
             jitter = 0.35, dodge = True, palette = 'Dark2', size = 3);

plt.legend(['Male','Female','Other'])
plt.title("Count of Gender,birth,year distances cycled");
plt.ylim((0,20));
```



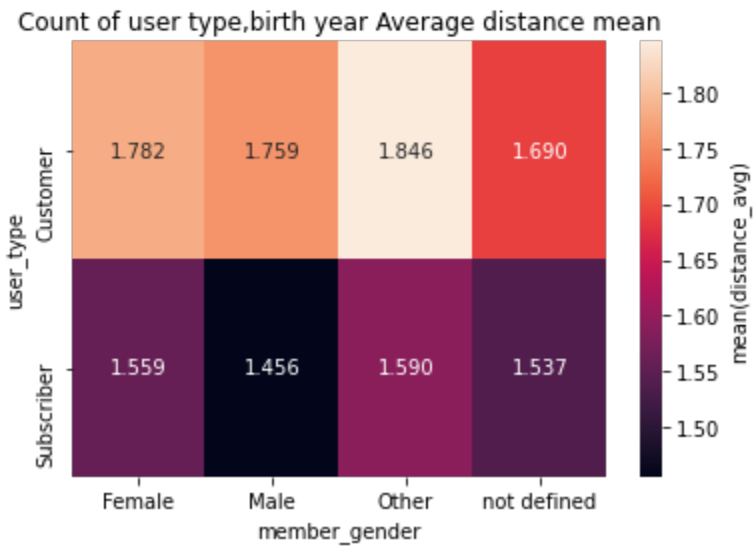
In [151…

```
#set plot dimensions
plt.figure(figsize=[14,6])

#create  a decade variable, using floor division (lowest integer divisor)
df['member_birth_decade'] = ((df['member_birth_year']//10)*10).astype(int)

#use decade
sb.stripplot(data = df.query('member_birth_decade>0'),
             x = 'member_birth_decade', y = 'duration_sec', hue ='member_ge
             jitter = 0.35, dodge = True, palette = 'Dark2', size = 3);

plt.legend(['Male','Female','Other'])
plt.title("Count of Gender,birth,year distances cycled");
plt.ylim((0,90000));
```



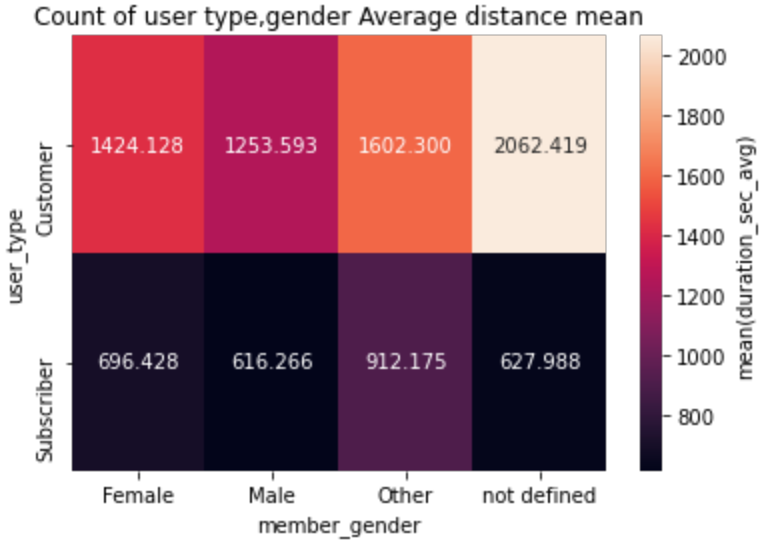
In [102…

```
#plot heatmap with seaborn counts of gender, user type and average distanc
cat_means = df.groupby(['member_gender', 'user_type']).mean()['distances']
cat_means = cat_means.reset_index(name = 'distance_avg')
cat_means = cat_means.pivot(index = 'user_type', columns = 'member_gender'
                            values = 'distance_avg')
sb.heatmap(cat_means, annot = True, fmt = '.3f',
           #set heat map to distance_avg mean
           cbar_kws = {'label' : 'mean(distance_avg)'});
plt.title("Count of user type,birth year Average distance mean");
```



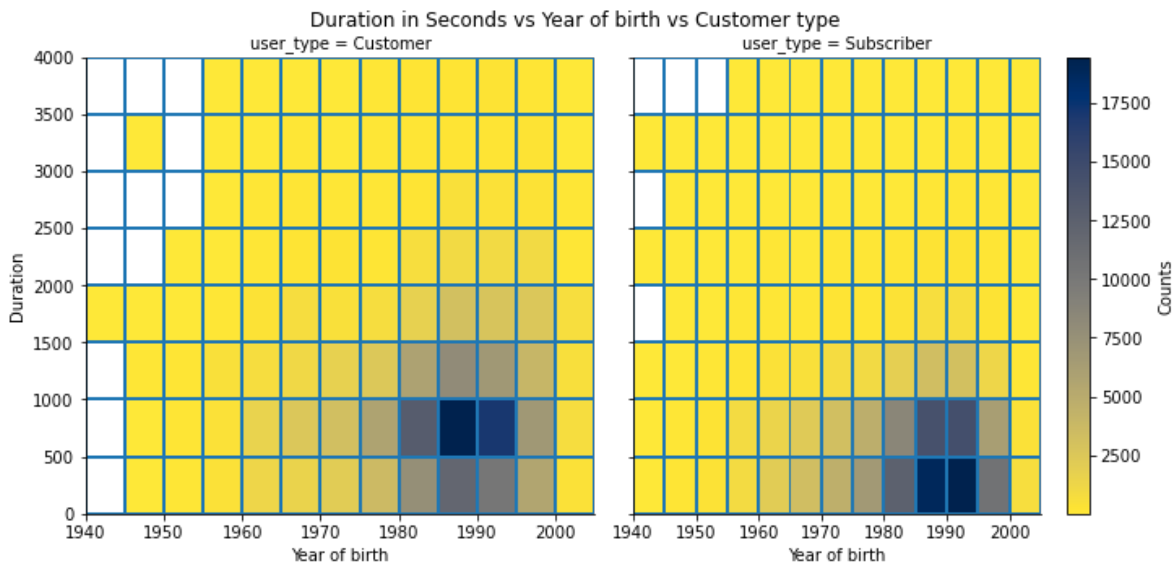
In [103…

```
#plot heatmap with seaborn counts of gender, user type and average duratio
cat_means = df.groupby(['member_gender', 'user_type']).mean()['duration_se
cat_means = cat_means.reset_index(name = 'duration_sec_avg')
cat_means = cat_means.pivot(index = 'user_type', columns = 'member_gender'
                            values = 'duration_sec_avg')
plt.title("Count of user type,gender Average distance mean");
sb.heatmap(cat_means, annot = True, fmt = '.3f',
           #set heat map to duration_avg mean
           cbar_kws = {'label' : 'mean(duration_sec_avg)'});
```



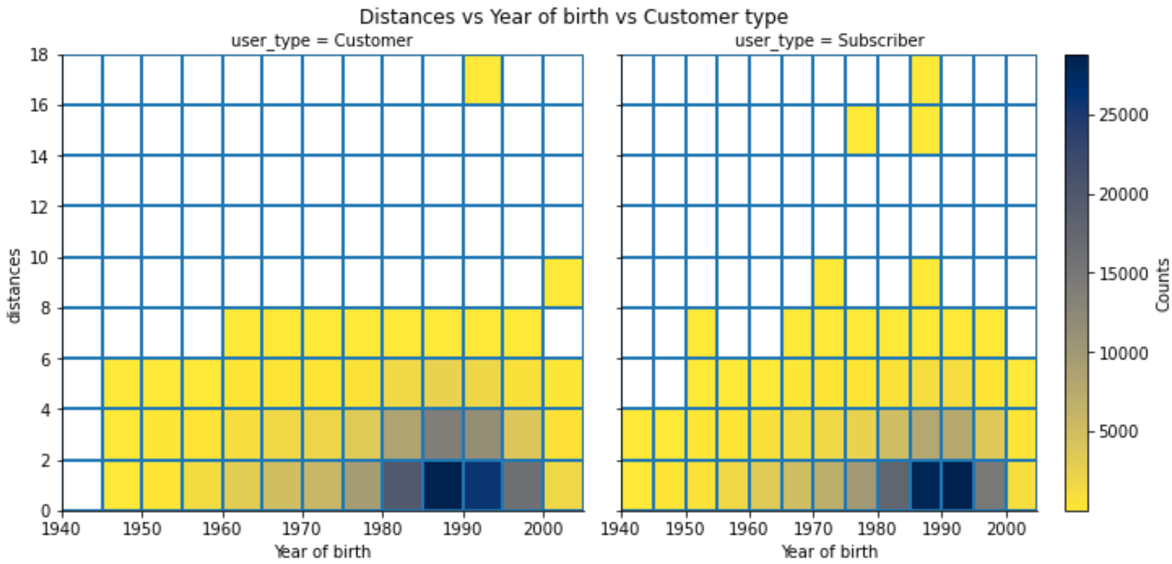
In [106…

```
#plot 2D histogram showing counts of duration in Seconds vs Year of birth
xbin = np.arange(1940, df['member_birth_year'].max()+5, 5)
ybin = np.arange(0, 4500, 500)
#set grid size
grid = sb.FacetGrid(data = df, col = 'user_type',height=5)
grid.map(plt.hist2d, 'member_birth_year','duration_sec',cmin=0.5,cmap = 'c
plt.colorbar().set_label('Counts');

plt.subplots_adjust(top=0.9)
grid.fig.suptitle("Duration in Seconds vs Year of birth vs Customer type")

grid.set_ylabels("Duration");
grid.set_xlabels("Year of birth");
```



In [107…

```
#plot 2D histogram showing counts of distances vs Year of birth vs Custome
xbin = np.arange(1940, df['member_birth_year'].max()+5, 5)
ybin = np.arange(0, 20, 2)
#set grid size
grid = sb.FacetGrid(data = df, col = 'user_type',height=5)
grid.map(plt.hist2d, 'member_birth_year','distances',cmin=0.5,cmap = 'civi
plt.colorbar().set_label('Counts');

plt.subplots_adjust(top=0.9)
grid.fig.suptitle("Distances vs Year of birth vs Customer type");

grid.set_ylabels("distances");
grid.set_xlabels("Year of birth");
```



Talk about some of the relationships you observed in this part of the investigation. Were there features that strengthened each other in terms of looking at your feature(s) of interest?

- Customers cycled over double the average duration than subscribers. The not defined gender cycled for the longest durations, which means either, male or female didn't specify their gender or there were outliers in the data.
- The average distances didn't vary greatly between customers and subscribers, which means there could have been long periods of time where people weren't cycling, because greater cycling durations should equal greater distances cycled.
- distances greater than 12km cycled were only by users born between 1990 and 2000, but these could be outliers.
- Any distance recorded over 10km was by a male user.
- it can be seen that users males cycled the farthest distances, followed by females, then other.

Were there any interesting or surprising interactions between features?

*

In [108…

```
df.to_csv("master_data_frame.csv", sep=',', encoding='utf-8')
```

Conclusions

- This data covers three areas: San Francisco, East Bay and San José
- It can be seen that bike usage is highest on a Thursday and is lowest on the weekends.
- There are alot more male users than other users, for all areas there are always over 64% male
- There are a lot more subscribers than customer using this service
- The average user over all data is most likely between 24 and 35 years old
- People use the bikes more/in higher counts during the week than during the weekend
- The majority of trips are less than 2km long.
- There is an increasing trend of usage
- People start their trips most frequently at 8 and 17 'o clock

During this project I had to clean the data and used folium to plot the Lat Long start and finish locations to visualise the journeys. I analysed the data by plotting exploratory graphs, then used Univariate, Bivariate and Multivariate graphs to further explore the relationships and trends od duration and distances cycled by age, gender and user type. I felt ther was no major surprises from the findings, other than males were the majority of users.

References

- https://towardsdatascience.com/plotting-maps-with-geopandas-428c97295a73
- https://www.datasciencemadesimple.com/pie-chart-in-python-with-legends/
- https://data.sfgov.org/Housing-and-Buildings/Land-Use/us3s-fp9q
- https://colab.research.google.com/drive/1MQQ3wUkMFQHLSJ9L4zjkueqet_PyJuyS#scrollTo=iOvl-TgaBm-G
- https://datascientyst.com/get-most-frequent-values-pandas-dataframe/
- https://colab.research.google.com/drive/1MQQ3wUkMFQHLSJ9L4zjkueqet_PyJuyS#scrollTo=hzbPtRDhfSrq
- https://www.findlatitudeandlongitude.com/
- https://datascientyst.com/plot-latitude-longitude-pandas-dataframe-python/
- https://towardsdatascience.com/calculating-distance-between-two-geolocations-in-python-26ad3afe287b
- https://www.w3resource.com/python-exercises/math/python-math-exercise-27.php