Using BigQuery to perform basic data analytics

Here's a sample notebook of executing SQL commands in order to analyze some data, along with some basic visualization.

We'll explore some data sets and reproduce how we might write queries for certain business problems.

Setup

In [1]:

```
# relevant installs
# !pip install google-cloud
# !pip install --upgrade google-cloud-bigquery[pandas]
# !pip install google-cloud-storage
```

In [2]:

```
%load_ext google.cloud.bigquery
```

In [3]:

```
SERVICE_ACCOUNT= 'bq_jupyter'
JSON_FILE_NAME = '../credentials/ds-portfolio-a04fdb631b73.json'
GCP_PROJECT_ID = 'ds-portfolio'
```

In [4]:

```
import subprocess
import sys
import logging
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

logger = logging.Logger('catch_all')
```

```
def run_command(parameters):
    try:
        # """Prints and runs a command."""
        return subprocess.check_output(parameters)
    except BaseException as e:
       logger.error(e)
       logger.error('ERROR: Looking in jupyter console for more information')
```

In [5]:

```
%matplotlib inline
```

Queries

We'll be using the San Francisco Bikeshares dataset, which contains information around trips for the bikeshare program in San Fr

In [6]:

```
from google.cloud import bigquery

client = bigquery.Client.from_service_account_json(JSON_FILE_NAME)

def query_to_df(query):
    # transfers query results to pandas dataframe for easy manipulating
    return(client.query(query).result().to_dataframe())

def get_schema(table):
    # retreives the schema as a printed object
    return(client.get_table(table).schema)
```

There's 4 different tables in this database. As a first step, we should look at the schema of all of these tables and see where we m

The tables are...

- bikeshare_regions
- bikeshare_station_info
- bikeshare_station_status
- bikeshare_trips

In [7]:

```
#bikeshare_regions
table = 'bigquery-public-data.san_francisco_bikeshare.bikeshare_regions'
get_schema(table)
```

Out[7]:

```
[SchemaField('region_id', 'INTEGER', 'REQUIRED', 'Unique identifier for the region', ()),
 SchemaField('name', 'STRING', 'REQUIRED', 'Public name for this region', ())]
```

In [8]:

```
#bikeshare_station_info
table = 'bigquery-public-data.san_francisco_bikeshare.bikeshare_station_info'
get_schema(table)
```

Out[8]:

```
[SchemaField('station_id', 'INTEGER', 'REQUIRED', 'Unique identifier of a station.', ()),
 SchemaField('name', 'STRING', 'REQUIRED', 'Public name of the station', ()),
 SchemaField('short_name', 'STRING', 'NULLABLE', 'Short name or other type of identifier, as used by the data p
 SchemaField('lat', 'FLOAT', 'REQUIRED', 'The latitude of station. The field value must be a valid WGS 84 lati
imal_degrees', ()),
 SchemaField('lon', 'FLOAT', 'REQUIRED', 'The longitude of station. The field value must be a valid WGS 84 long
ecimal_degrees', ()),
 SchemaField('region_id', 'INTEGER', 'NULLABLE', 'ID of the region where station is located', ()),
 SchemaField('rental_methods', 'STRING', 'NULLABLE', 'Array of enumerables containing the payment methods accep
PASS APPLEPAY ANDROIDPAY TRANSITCARD ACCOUNTNUMBER PHONE This list is intended to be as comprehensive at the ti
 SchemaField('capacity', 'INTEGER', 'NULLABLE', 'Number of total docking points installed at this station, both
 SchemaField('external_id', 'STRING', 'NULLABLE', '', ()),
 SchemaField('rental_url', 'STRING', 'NULLABLE', '', ()),
 SchemaField('eightd_has_key_dispenser', 'BOOLEAN', 'NULLABLE', '', ()),
 SchemaField('has_kiosk', 'BOOLEAN', 'NULLABLE', '', ()),
 SchemaField('station_geom', 'GEOGRAPHY', 'NULLABLE', '', ())]
```

In [9]:

```
#bikeshare_station_status
table = 'bigquery-public-data.san_francisco_bikeshare.bikeshare_station_status'
get_schema(table)
```

Out[9]:

```
[SchemaField('station_id', 'INTEGER', 'REQUIRED', 'Unique identifier of a station', ()),
 SchemaField('num_bikes_available', 'INTEGER', 'REQUIRED', 'Number of bikes available for rental', ()),
 SchemaField('num_bikes_disabled', 'INTEGER', 'NULLABLE', 'Number of disabled bikes at the station. Vendors who
ion_information), num_bikes_disabled and num_docks_disabled. If station capacity is published then broken docks
```

```
SchemaField('num_docks_available', 'INTEGER', 'REQUIRED', 'Number of docks accepting bike returns', ()),
SchemaField('num_docks_disabled', 'INTEGER', 'NULLABLE', 'Number of empty but disabled dock points at the stat
SchemaField('is_installed', 'BOOLEAN', 'REQUIRED', '1/0 boolean - is the station currently on the street', ())
SchemaField('is_renting', 'BOOLEAN', 'REQUIRED', '1/0 boolean - is the station currently renting bikes (even i
SchemaField('is_returning', 'BOOLEAN', 'REQUIRED', '1/0 boolean - is the station accepting bike returns (if a
SchemaField('last_reported', 'INTEGER', 'REQUIRED', 'Integer POSIX timestamp indicating the last time this sta
SchemaField('num_ebikes_available', 'INTEGER', 'NULLABLE', '', ()),
SchemaField('eightd_has_available_keys', 'BOOLEAN', 'NULLABLE', '', ())]
```

In [10]:

```
#bikeshare_trips
table = 'bigquery-public-data.san_francisco_bikeshare.bikeshare_trips'
get_schema(table)
```

Out[10]:

```
[SchemaField('trip_id', 'INTEGER', 'REQUIRED', 'Numeric ID of bike trip', ()),
 SchemaField('duration_sec', 'INTEGER', 'NULLABLE', 'Time of trip in seconds', ()),
 SchemaField('start_date', 'TIMESTAMP', 'NULLABLE', 'Start date of trip with date and time, in PST', ()),
 SchemaField('start_station_name', 'STRING', 'NULLABLE', 'Station name of start station', ()),
 SchemaField('start_station_id', 'INTEGER', 'NULLABLE', 'Numeric reference for start station', ()),
 SchemaField('end_date', 'TIMESTAMP', 'NULLABLE', 'End date of trip with date and time, in PST', ()),
 SchemaField('end_station_name', 'STRING', 'NULLABLE', 'Station name for end station', ()),
 SchemaField('end_station_id', 'INTEGER', 'NULLABLE', 'Numeric reference for end station', ()),
 SchemaField('bike_number', 'INTEGER', 'NULLABLE', 'ID of bike used', ()),
 SchemaField('zip_code', 'STRING', 'NULLABLE', 'Home zip code of subscriber (customers can choose to manually e
 SchemaField('subscriber_type', 'STRING', 'NULLABLE', 'Subscriber = annual or 30-day member; Customer = 24-hour
 SchemaField('c_subscription_type', 'STRING', 'NULLABLE', '', ()),
 SchemaField('start_station_latitude', 'FLOAT', 'NULLABLE', '', ()),
 SchemaField('start_station_longitude', 'FLOAT', 'NULLABLE', '', ()),
 SchemaField('end_station_latitude', 'FLOAT', 'NULLABLE', '', ()),
 SchemaField('end_station_longitude', 'FLOAT', 'NULLABLE', '', ()),
 SchemaField('member_birth_year', 'INTEGER', 'NULLABLE', '', ()),
 SchemaField('member_gender', 'STRING', 'NULLABLE', '', ()),
 SchemaField('bike_share_for_all_trip', 'STRING', 'NULLABLE', '', ()),
 SchemaField('start_station_geom', 'GEOGRAPHY', 'NULLABLE', '', ()),
 SchemaField('end_station_geom', 'GEOGRAPHY', 'NULLABLE', '', ())]
```

So when we take a look at the schemas, we can see that each table gives us some different information. A few things that jump o

- We get some interesting information from the `station_info` table regarding payment types. It could be interesting to loo
- The `bikeshare_trips` table will give us information around ride-by-ride stats and has unique identifiers around custome

- We have additional information for members, but not for customers
- This will allow us to take a look at where popular routes might be

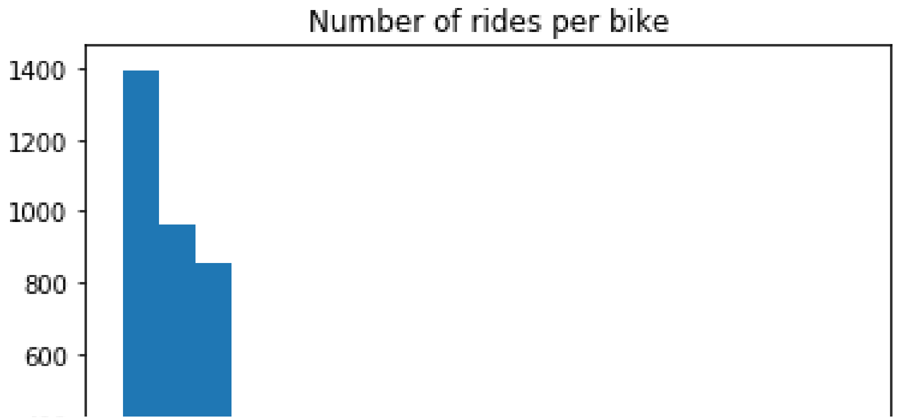
At this point, we can probably start looking at doing some queries for some explorative work, and seeing where we might be ab

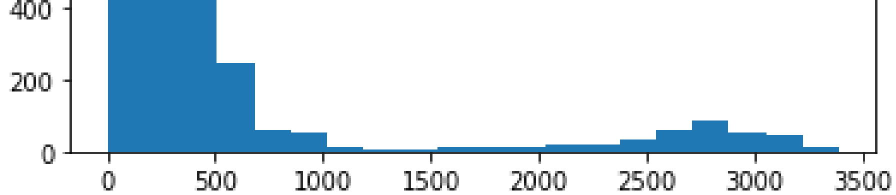
Let's start out by looking how many rides each of the bikes in our dataset have on them. This might give us an idea how much w

In [11]:

```
# q1
# which bikes have been used the most?
QUERY = (
    """
    SELECT
        COUNT(trip_id) AS num_trips, bike_number
    FROM
      `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
    GROUP BY
      bike_number
    ORDER BY
      num_trips DESC
    """)


ret_df = query_to_df(QUERY)
plt.hist(ret_df.num_trips, bins = 20)
plt.title("Number of rides per bike")
```

Out[11]: `Text(0.5, 1.0, 'Number of rides per bike')`





So here we see that the distribution is not normal, and looks like there's two different fundamental groups that we're dealing wit
2750. It might be interesting to look at the differences between these two groups of bikes- maybe they tend to be found on diff

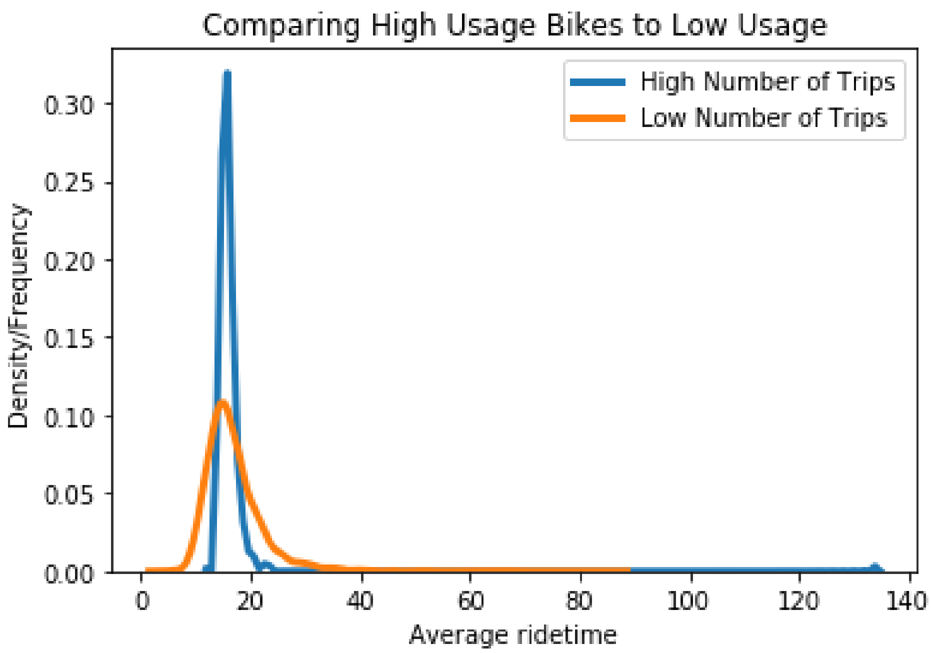
We'll start off by looking at the differences between the average ride time between the many-rides group and the few-rides grou

In [12]:

```
# q2
# compare average ride times for bikes above/below 1500 bikes

# high rides query
q2_a = (
    """
    SELECT
        avg(duration_sec)/60 AS avg_trip_length_min,
        COUNT(trip_id) AS num_trips,
        bike_number
    FROM
        `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
    GROUP BY
        bike_number
    HAVING
        num_trips >= 1500
    """)

ret_df_a = query_to_df(q2_a)

# low rides query
q2_b = (
    """
    SELECT
        avg(duration_sec)/60 AS avg_trip_length_min,
        COUNT(trip_id) AS num_trips,
        bike_number
    FROM
```

```
    FROM
        `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
    GROUP BY
        bike_number
    HAVING
        num_trips < 1500
    """)

ret_df_b = query_to_df(q2_b)

# plot them on the same plot with density lines rather than histograms
sns.distplot(ret_df_a['avg_trip_length_min'], hist = False, kde = True,
             kde_kws = {'linewidth': 3},
             label = "High Number of Trips")
sns.distplot(ret_df_b['avg_trip_length_min'], hist = False, kde = True,
             kde_kws = {'linewidth': 3},
             label = "Low Number of Trips")
plt.legend(prop={'size':10})
plt.title("Comparing High Usage Bikes to Low Usage")
plt.xlabel("Average ridetime")
plt.ylabel("Density/Frequency")
```

Out[12]: `Text(0, 0.5, 'Density/Frequency')`



So we can see that our bikes with lower number of trips have a higher variance around the average trip length, whereas the high high-traffic routes (that presumably are around 17 minutes or so).

Note that I used a density plot instead of comparing histograms. From a data visualization perspective, we want these plotted or plot which comes across much cleaner with the same kind of takeaway as the histogram.

In the query, we also use `HAVING` instead of `WHERE` since the condition is applied after our grouping aggregation.

Interestingly, the overall mean of both of these appear to be the same. We'll calculate some basic statics below to confirm.

In [13]:

```
# high usage
ret_df_a.describe()
```

Out[13]:

	avg_trip_length_min	num_trips	bike_number
count	358.000000	358.000000	358.000000
mean	16.171550	2665.849162	438.709497
std	6.458857	409.004550	139.299144
min	13.069868	1517.000000	16.000000
25%	14.866111	2469.750000	349.250000
50%	15.545687	2754.000000	447.500000
75%	16.481781	2947.000000	545.500000
max	133.671334	3394.000000	878.000000

In [14]:

```
# low usage
ret_df_b.describe()
```

Out[14]:

	avg_trip_length_min	num_trips	bike_number
count	3594.000000	3594.000000	3594.000000

mean	16.913051	276.306344	2152.675849
std	5.593865	205.036652	1090.074206
min	3.858333	1.000000	9.000000
25%	13.594331	114.250000	1276.250000
50%	15.837713	230.500000	2175.500000
75%	18.869190	416.000000	3076.750000
max	86.211910	1471.000000	4073.000000

A few more things to note...

- We have a lot more bikes in the low usage group compared to high usage group, by about 9x
- The means are pretty close, but the standard deviations are less similar. The higher usage has a higher variance but lower m
- We could run a t-test to see if the means are equal... with such large sample sizes we will likely come to the conclusion that t

Let's take a look into the routes used and see if this explains the differences.

In [15]:

```
# routes for high usage
# q3
# maybe there's a more effecient way, but this works
q3_a = (
    """
    SELECT
        SUM(num_trips) as trips,
        start_station_id,
        end_station_id
    FROM
        (SELECT
            COUNT(trip_id) AS num_trips,
            start_station_id,
            end_station_id,
            bike_number
         FROM
```

```
            `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
        WHERE
            bike_number IN
                (SELECT bike_number
                 FROM
                    (SELECT
                     COUNT(trip_id) AS num_trips,
                        bike_number
                     FROM
                        `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
                     GROUP BY
                        bike_number
                     HAVING
                        num_trips >= 1500
                    )
                )
        GROUP BY
            start_station_id, end_station_id, bike_number
        )
    GROUP BY
        start_station_id, end_station_id
    ORDER BY
        trips DESC
    LIMIT 25
    """)

high_vol_routes = query_to_df(q3_a)
high_vol_routes
```

Out[15]:

	trips	start_station_id	end_station_id
0	8749	50	60
1	8168	69	65
2	7281	61	50
3	6601	50	61
4	6568	65	69
5	6557	60	74

6	6065	51	70
7	5930	70	50
8	5790	74	61
9	5714	74	70
10	5597	55	70
11	5159	50	70
12	5113	65	70
13	5086	64	77
14	4977	70	55
15	4921	67	69
16	4887	74	60
17	4804	77	64
18	4530	60	50
19	4318	69	39
20	4241	39	69
21	4238	69	57
22	4231	70	51
23	4150	70	74
24	4111	63	70

In [16]:

```
# routes for low usage
# q3
# maybe there's a more effecient way, but this works
q3_b = (
    """
```

```
    SELECT
        SUM(num_trips) as trips,
        start_station_id,
        end_station_id
    FROM
        (SELECT
            COUNT(trip_id) AS num_trips,
            start_station_id,
            end_station_id,
            bike_number
         FROM
            `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
         WHERE
            bike_number IN
                (SELECT bike_number
                 FROM
                    (SELECT
                     COUNT(trip_id) AS num_trips,
                        bike_number
                     FROM
                        `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
                     GROUP BY
                        bike_number
                     HAVING
                        num_trips < 1500
                    )
                )
         GROUP BY
            start_station_id, end_station_id, bike_number
        )
    GROUP BY
        start_station_id, end_station_id
    ORDER BY
        trips DESC
    LIMIT 25
    """)

low_vol_routes = query_to_df(q3_b)
low_vol_routes
```

Out[16]:

	trips	start_station_id	end_station_id
0	4930	15	6
1	3758	28	27
2	3444	27	28
3	3129	4	2
4	3096	2	4
5	2872	6	16
6	2716	81	15
7	2469	32	28
8	2468	6	15
9	2277	15	81
10	2216	28	32
11	2127	16	6
12	2098	182	196
13	1870	6	6
14	1841	196	182
15	1825	58	67
16	1734	195	182
17	1677	29	31
18	1671	31	29
19	1659	22	30
20	1627	2	6
21	1606	17	27
22	1604	30	28

23	1571	23	30
24	1562	45	67

Let's do some pandas trickery now that we've gotten the data from our database as a comparison.

In [17]:
```
high_vol_routes['route_coding'] = high_vol_routes.start_station_id.astype(str) + \
    "_" + high_vol_routes.end_station_id.astype(str)
low_vol_routes['route_coding'] = low_vol_routes.start_station_id.astype(str) + \
    "_" + low_vol_routes.end_station_id.astype(str)

high_vol_routes['in_low'] = high_vol_routes['route_coding'].\
    isin({'route_coding': low_vol_routes.route_coding.values.tolist()})
high_vol_routes
```

Out[17]:

	trips	start_station_id	end_station_id	route_coding	in_low
0	8749	50	60	50_60	False
1	8168	69	65	69_65	False
2	7281	61	50	61_50	False
3	6601	50	61	50_61	False
4	6568	65	69	65_69	False
5	6557	60	74	60_74	False
6	6065	51	70	51_70	False
7	5930	70	50	70_50	False
8	5790	74	61	74_61	False
9	5714	74	70	74_70	False
10	5597	55	70	55_70	False
11	5159	50	70	50_70	False

12	5113	65	70	65_70	False
13	5086	64	77	64_77	False
14	4977	70	55	70_55	False
15	4921	67	69	67_69	False
16	4887	74	60	74_60	False
17	4804	77	64	77_64	False
18	4530	60	50	60_50	False
19	4318	69	39	69_39	False
20	4241	39	69	39_69	False
21	4238	69	57	69_57	False
22	4231	70	51	70_51	False
23	4150	70	74	70_74	False
24	4111	63	70	63_70	False

Interestingly, none of the top 25 routes for the high volume bikes are in the top 25 low volume bike routes. While we could drill routes for the higher volume bikes and lower volume bikes. This might be an interesting business result if the company is experi for more equal wear. Of course, more detailed analysis on a station-by-station level rather than route-level would be warranted i

Let's pivot a little more to looking at some of our customers and subscribers.

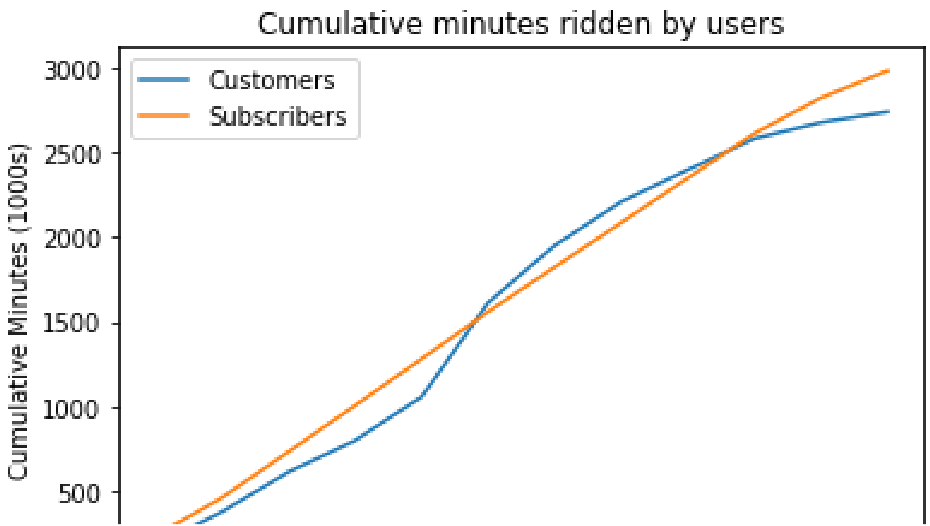
I want to take a look at the cumulative minutes spent on bike rides by our subscribers vs customers on a month-by-month basis

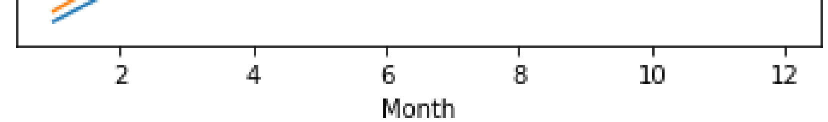
In [18]:

```
# q4
q4 = '''
SELECT
  SUM(customer_minutes_sum) OVER (ORDER BY end_month ROWS UNBOUNDED PRECEDING)/1000 as cumulative_minutes_cust
  SUM(subscriber_minutes_sum) OVER (ORDER BY end_month ROWS UNBOUNDED PRECEDING)/1000 as cumulative_minutes_su
  end_year,
  end_month
```

```
FROM
  (
  SELECT
    SUM(CASE WHEN subscriber_type = 'Customer' THEN duration_sec/60 ELSE NULL END) AS customer_minutes_sum,
    SUM(CASE WHEN subscriber_type = 'Subscriber' THEN duration_sec/60 ELSE NULL END) AS subscriber_minutes_sum
    EXTRACT(YEAR FROM end_date) AS end_year,
    EXTRACT(MONTH FROM end_date) AS end_month
  FROM
    `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
  GROUP BY
    end_year, end_month
  HAVING
    end_year = 2015
  )
ORDER BY
  end_year, end_month
'''
df4 = query_to_df(q4)
plt.plot(df4.end_month, df4.cumulative_minutes_cust, label = "Customers")
plt.plot(df4.end_month, df4.cumulative_minutes_sub, label = "Subscribers")
plt.title("Cumulative minutes ridden by users")
plt.xlabel("Month")
plt.ylabel("Cumulative Minutes (1000s)")
plt.legend()
```

Out[18]: <matplotlib.legend.Legend at 0x1b2edfb76d8>





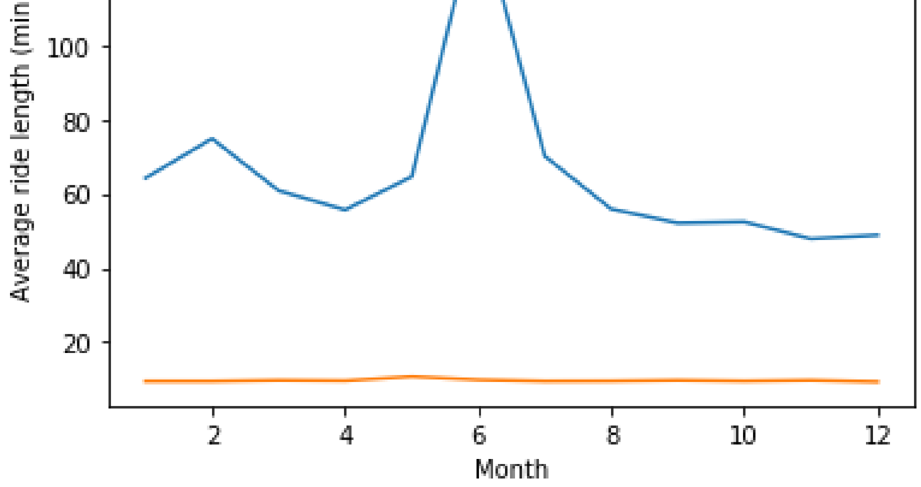
We see a bit of an interesting phenomena here. Our subscribers, AKA our people that pay for longer-term memberships, are usir
(either a 3 day membership or single day) really use them a lot more in the summer, months 6 - 8. Overall, the subscribers will sp

Let's change that query up slightly and look how the average ride length changes over months.

In [19]:

```
q5 = '''
  SELECT
    AVG(CASE WHEN subscriber_type = 'Customer' THEN duration_sec/60 ELSE NULL END) AS customer_minutes_avg,
    AVG(CASE WHEN subscriber_type = 'Subscriber' THEN duration_sec/60 ELSE NULL END) AS subscriber_minutes_avg
    EXTRACT(YEAR FROM end_date) AS end_year,
    EXTRACT(MONTH FROM end_date) AS end_month
  FROM
    `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
  GROUP BY
    end_year, end_month
  HAVING
    end_year = 2015
  ORDER BY
    end_year, end_month
'''
df5 = query_to_df(q5)
plt.plot(df5.end_month, df5.customer_minutes_avg, label = "Customers")
plt.plot(df5.end_month, df5.subscriber_minutes_avg, label = "Subscribers")
plt.title("Average minutes ridden per trip")
plt.xlabel("Month")
plt.ylabel("Average ride length (min)")
plt.legend()
```

Out[19]: <matplotlib.legend.Legend at 0x1b2ee025940>



This picture very clearly shows the phenomena that we showed with the other query- the average ride length skyrockets over the most part, or at least keeping their habits very consistent. It's also interesting to note that the average ride length is much longe more than subscribers. Given the fact that the average is so much lower and the previous chart looks the way it did, we can infer

To utilize data from multiple tables, we'll take a look at the origin stations popular with customers and subscribers and see if the

In [20]:

```
q6_cust = '''
SELECT
  SUM(CASE WHEN trips.subscriber_type = 'Customer' THEN trips.trip_id/1000000 ELSE NULL END) AS cust_trips_mil
  info.station_id AS station,
  info.capacity AS cap
FROM
  `bigquery-public-data.san_francisco_bikeshare.bikeshare_station_info` AS info
  INNER JOIN
  `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips` AS trips
  ON info.station_id = trips.start_station_id
GROUP BY
  station, cap
ORDER BY
  cust_trips_mil DESC
LIMIT
  25
'''
df6_cust = query_to_df(q6_cust)
df6_cust.head()
```

Out[20]:

	cust_trips_mil	station	cap
0	1.561808e+15	6	23
1	1.394113e+15	15	38
2	8.276351e+14	3	35
3	7.868649e+14	60	31
4	7.781771e+14	70	31

In [21]:

```
q6_sub = '''
SELECT
  SUM(CASE WHEN trips.subscriber_type = 'Subscriber' THEN trips.trip_id/1000000 ELSE NULL END) AS sub_trips_mi
  info.station_id AS station,
  info.capacity AS cap
FROM
  `bigquery-public-data.san_francisco_bikeshare.bikeshare_station_info` AS info
  INNER JOIN
  `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips` AS trips
  ON info.station_id = trips.start_station_id
GROUP BY
  station, cap
ORDER BY
  sub_trips_mil DESC
LIMIT
  25
'''
df6_sub = query_to_df(q6_sub)
df6_sub.head()
```

Out[21]:

	sub_trips_mil	station	cap
0	3.945897e+15	70	31
1	3.811147e+15	30	19
2	3.587979e+15	58	31

3	3.343885e+15	81	35
4	3.157456e+15	15	38

Right away, we can see that the most frequent stations to start a trip for both subscribers and customers include station 70, indic

We'll take a look at the mean and standard deviation around each capacity in the 25 stations for both sides.

In [22]:

```
mean_cust = np.mean(df6_cust.cap)
sd_cust = np.std(df6_cust.cap)
print("Mean of top 25 customer stations capacity:")
print(mean_cust)
print("Standard Deviation of top 25 customer stations capacity:")
print(sd_cust)
```

```
Mean of top 25 customer stations capacity:
29.2
Standard Deviation of top 25 customer stations capacity:
6.5482822174979605
```

In [23]:

```
mean_sub = np.mean(df6_sub.cap)
sd_sub = np.std(df6_sub.cap)
print("Mean of top 25 subscriber stations capacity:")
print(mean_sub)
print("Standard Deviation of top 25 subscriber stations capacity:")
print(sd_sub)
```

```
Mean of top 25 subscriber stations capacity:
29.32
Standard Deviation of top 25 subscriber stations capacity:
6.024748957425528
```

We don't really see a big difference here. We might look into doing some kind of weighted comparison approach and/or hypoth

In [ ]: