Zapier Monthly Active User Analysis

January 20, 2020

0.0.1 Importing modules and packages

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import psycopg2
import prophet
from fbprophet import Prophet
from fbprophet.plot import plot_plotly
import plotly.offline as py
import os
warnings.filterwarnings('ignore')
%matplotlib inline
py.init_notebook_mode()
color = sns.color_palette()
pd.set_option('display.float_format', lambda x: '%.3f' % x)
sns.set(rc={'figure.figsize':(8,5)})
```

0.0.2 Connecting to the Zapier database

- A view called **active_users_analysis** has been created in the **jchumley** schema using the **tasks_used_da table**.
- We'll use the **psycopg** odbc module to connect to the Redshift cluster. Pandas will be used extensively in conjunction with Postgres/Redshift queries.
- The user, password, host, and database parameters are loaded from a .bashrc file.

[2]:
```
port='5439'
user=os.environ['ZAP_USER']
password=os.environ['ZAP_PASS']
host=os.environ['ZAP_HOST']
db=os.environ['ZAP_DB']
conn = psycopg2.connect(user=user, password=password, host=host, port=port,␣
 ↪database=db)
```

0.0.3 Creating dataframe

- Let's create our first dataframe using pandas read_sql method.

[3]:
```
query = """
SELECT * FROM
jchumley.active_users_analysis
"""
df = pd.read_sql(query, conn)
df.head()
```

[3]:
```
         date  user_id  account_id  sum_tasks_used  cum_sum_tasks_used  \
0  2017-01-01        1           1              48                  48
1  2017-01-02        1           1              65                 113
2  2017-01-03        1           1              71                 184
3  2017-01-04        1           1              64                 304
4  2017-01-05        1           1              74                 406

   num_days_since_last_active  active  churned  month_num    month  dow_num  \
0                           0    True    False      1.000  January        0
1                           1    True    False      1.000  January        1
2                           1    True    False      1.000  January        2
3                           1    True    False      1.000  January        3
4                           1    True    False      1.000  January        4

         dow
0     Sunday
1     Monday
2    Tuesday
3  Wednesday
4   Thursday
```

- What is the shape of this dataset?

[4]:
```
df.shape
```

[4]:
```
(10547587, 12)
```

- There are ~10.5m rows and 12 columns.
- We'll also compute the ratio of missing values for each column.

[5]:
```
df_na = (df.isnull().sum() / len(df)) * 100
df_na = df_na.drop(df_na[df_na == 0].index).sort_values(ascending=False)
missing_data = pd.DataFrame({'Missing Ratio' :df_na})
missing_data.head()
```

[5]:
```
Empty DataFrame
Columns: [Missing Ratio]
```

```
Index: []
```

- We don't have null values in this data set.

0.1 Exploratory data analysis

0.1.1 Correlation Matrix

- Let's gain a deeper understanding of the associations between the features in this dataset. We'll use the pandas **corr** method to complete this task.

[6]:
```
df.corr()
```

[6]:

	user_id	account_id	sum_tasks_used
user_id	1.000	0.974	-0.022
account_id	0.974	1.000	-0.019
sum_tasks_used	-0.022	-0.019	1.000
cum_sum_tasks_used	0.984	0.958	-0.022
num_days_since_last_active	0.021	0.020	-0.012
active	-0.008	-0.008	0.004
churned	0.007	0.006	-0.003
month_num	0.108	0.108	0.001
dow_num	0.002	0.002	0.000

	cum_sum_tasks_used	num_days_since_last_active
user_id	0.984	0.021
account_id	0.958	0.020
sum_tasks_used	-0.022	-0.012
cum_sum_tasks_used	1.000	0.023
num_days_since_last_active	0.023	1.000
active	-0.009	-0.760
churned	0.008	0.519
month_num	0.096	0.060
dow_num	0.002	-0.031

	active	churned	month_num	dow_num
user_id	-0.008	0.007	0.108	0.002
account_id	-0.008	0.006	0.108	0.002
sum_tasks_used	0.004	-0.003	0.001	0.000
cum_sum_tasks_used	-0.009	0.008	0.096	0.002
num_days_since_last_active	-0.760	0.519	0.060	-0.031
active	1.000	-0.873	-0.041	-0.000
churned	-0.873	1.000	0.029	0.000
month_num	-0.041	0.029	1.000	0.016
dow_num	-0.000	0.000	0.016	1.000

- For this stage of analysis, the **user_id** and **account_id** columns will be ignored since they are unique identifiers.

- Notice that there is a strong negative association between the features **active** and **churn** which makes sense since they are mutually exclusive.

0.1.2 Active and churned users by month

- Let's take a look at counts of active and churned users.
- We'll run the following aggregate query to view churn and active user counts for each month in the dataset.

[7]:
```
query_month = """
SELECT
  month,
  month_num,
  count(CASE WHEN churned is TRUE then 1 END) as churned_count,
  count(CASE WHEN active is TRUE then 1 END) as active_count,
  ROUND(
    100.0 * count(CASE WHEN churned is TRUE then 1 END)/ count(*),
    2
  ) as churned_ratio,
  ROUND(
    100.0 * count(CASE WHEN active is TRUE then 1 END)/ count(*),
    2
  ) as active_ratio,
  count(*) as total_user_count
FROM
  jchumley.active_users_analysis
GROUP BY
  month,
  month_num
ORDER BY
  month_num
"""
df_month = pd.read_sql(query_month, conn)
```

[8]:
```
df_month
```

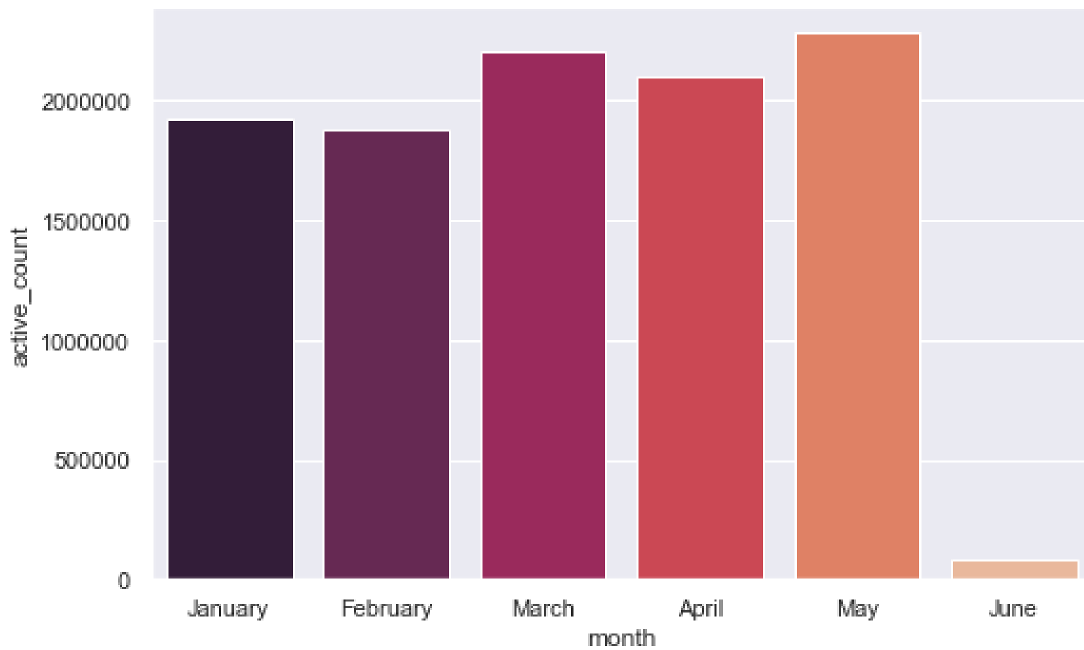
[8]:
```
      month  month_num  churned_count  active_count  churned_ratio  \
0   January      1.000            120       1927467          0.010
1  February      2.000           8256       1881695          0.440
2     March      3.000          12974       2207687          0.580
3     April      4.000          12676       2100295          0.600
4       May      5.000          14207       2282399          0.620
5      June      6.000            964         83651          1.140

   active_ratio  total_user_count
0        99.990           1927587
1        99.560           1889981
2        99.300           2223331
3        99.170           2117882
```

```
4        99.070            2303880
5        98.500              84926
```

[9]:
```python
sns.barplot(x='month', y='active_count', palette="rocket", data=df_month)
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2ce802d0>
```



- We can see that May had the highest count of active users. June has a very low count, perhaps because of data availability. Let's verify.

[10]:
```python
query_months_date_counts = """
SELECT
  month,
  COUNT(
    DISTINCT(date)
  )
FROM
  jchumley.active_users_analysis
GROUP BY
  month
"""
pd.read_sql(query_months_date_counts, conn)
```

[10]:

	month	count
0	January	31
1	June	1

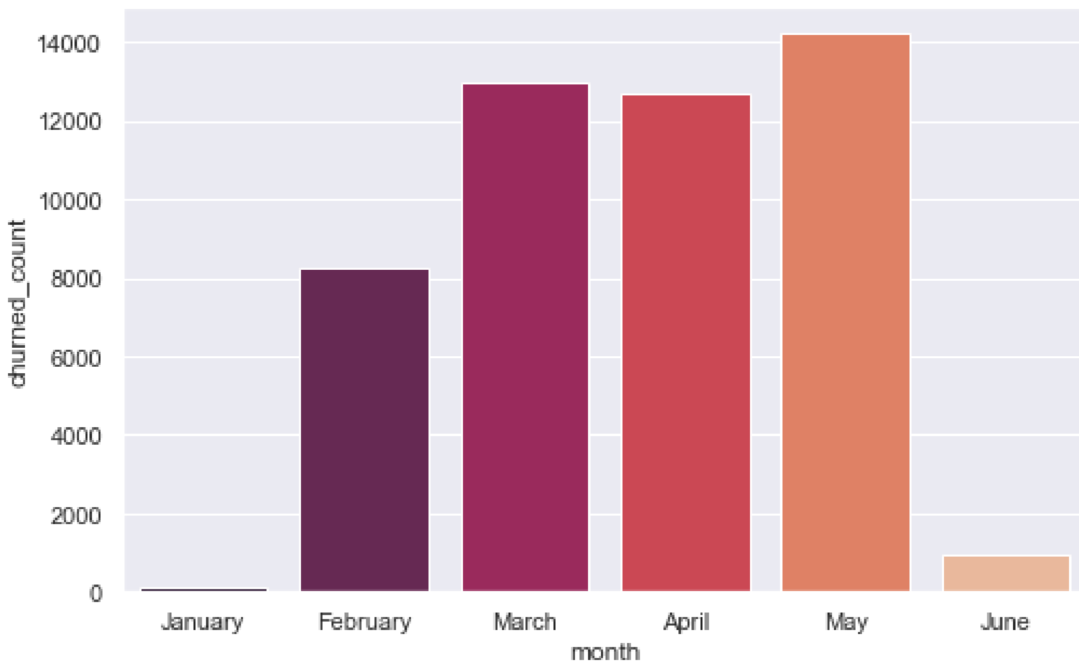
```
2       May     31
3     April     30
4  February     28
5     March     31
```

- We can see that the dataset includes records from the first day of June 2017 but not other days in that month/year.

0.1.3 Churn by month

[11]:
```python
sns.barplot(x='month', y='churned_count', palette="rocket", data=df_month)
```

[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2e1c35d0>



- January's churn count would likely increase if records from 2016 were availabile.
- Potential factors contributing to June's relatively high churn count:
    1. Users either chose to not renew or cancel their subscriptions on the first day of a given month.
    2. All June records are from the first day of the month.

0.1.4 Day of week analysis

- We'll run a query similar to the one used to compute aggregates by month.

[12]:
```
dow_query = """
SELECT
  dow,
  dow_num,
  count(CASE WHEN churned is TRUE then 1 END) as churned_count,
  count(CASE WHEN active is TRUE then 1 END) as active_count,
  ROUND(
    100.0 * count(CASE WHEN churned is TRUE then 1 END)/ count(*),
    2
  ) as churned_ratio,
  ROUND(
    100.0 * count(CASE WHEN active is TRUE then 1 END)/ count(*),
    2
  ) as active_ratio,
  count(*) as total_user_count
FROM
  jchumley.active_users_analysis
GROUP BY
  dow,
  dow_num
ORDER BY
  dow_num
"""
df_dow = pd.read_sql(dow_query, conn)
```

[13]:
```
df_dow
```

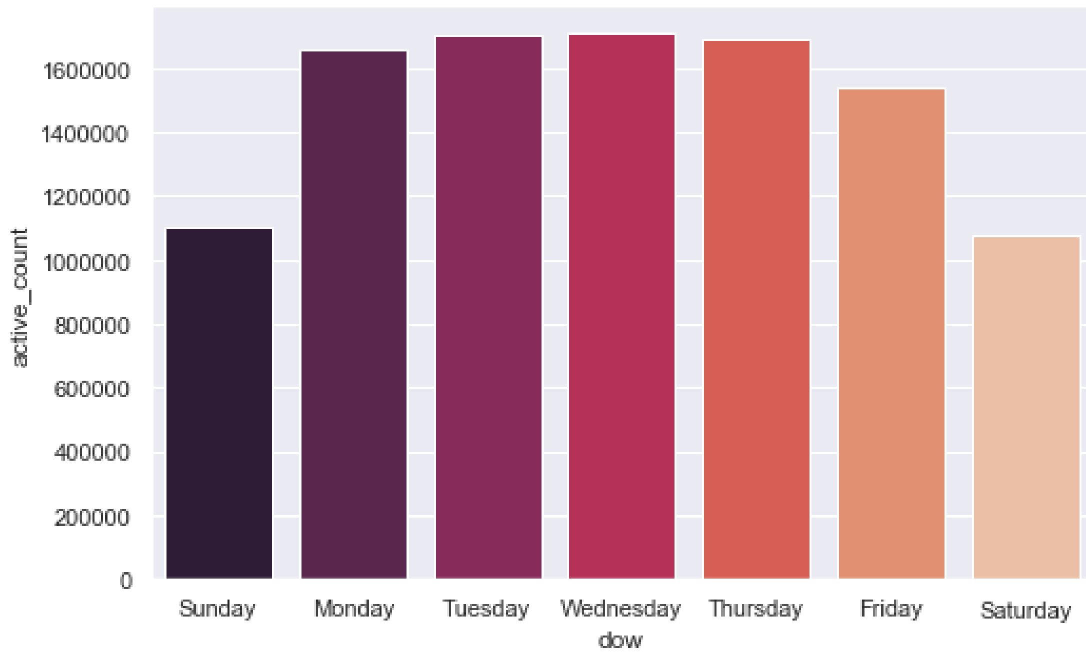
[13]:
```
         dow  dow_num  churned_count  active_count  churned_ratio  \
0     Sunday        0           3619       1102657          0.330
1     Monday        1           8207       1657615          0.490
2    Tuesday        2           8524       1703666          0.500
3  Wednesday        3           9268       1708853          0.540
4   Thursday        4           8773       1689305          0.520
5     Friday        5           6805       1540253          0.440
6   Saturday        6           4001       1080845          0.370

   active_ratio  total_user_count
0        99.570           1107429
1        99.370           1668200
2        99.340           1714992
3        99.300           1720901
4        99.320           1700834
5        99.420           1549246
6        99.530           1085985
```

0.1.5 Counts of active users by day of week

[14]:
```
sns.barplot(x='dow', y='active_count', palette="rocket", data=df_dow)
```

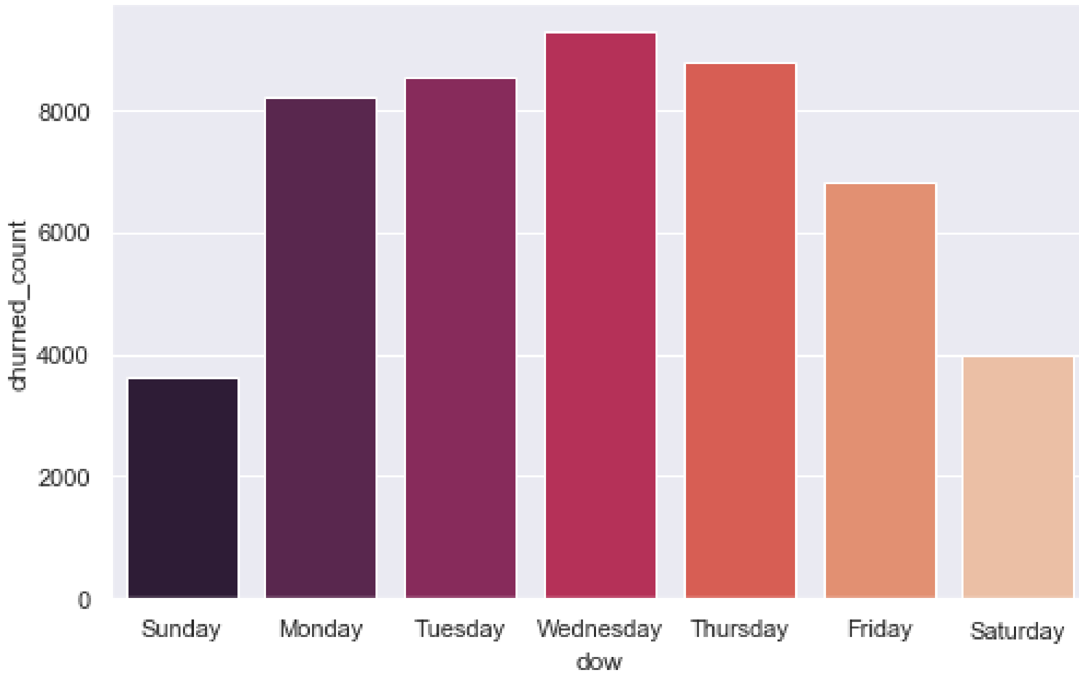
[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2efb8350>



0.1.6 Counts of churned users by day of week

[15]:
```
sns.barplot(x='dow', y='churned_count', palette="rocket", data=df_dow)
```

[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2f803910>



- Since Zapier is a B2B platform, its users are expected to be most active while they are in the office.
- Recall that **churned** and **active** have a very strong negative association (see the correlation matrix above), so it's expected that churn is higher during normal business hours.

0.1.7 Power Users

- Let's see who the top 10 Zapier users based on the **sum_tasks_used** column.

[16]:
```
power_user_query = """WITH total_sum_tasks_used_CTE AS (
  SELECT
    user_id,
    SUM(sum_tasks_used) as total_sum_tasks_used
  FROM
    jchumley.active_users_analysis
  GROUP BY
    user_id
  ORDER BY
    user_id
)
SELECT
  user_id,
  total_sum_tasks_used,
  ROUND(
    100.0 * total_sum_tasks_used /(
```

```
      SELECT
        SUM(total_sum_tasks_used) as total_sum_tasks_used_all_users
      FROM
        total_sum_tasks_used_CTE
    ),
    2
  ) pct_all_sum_tasks_used,
  RANK() OVER(
    ORDER BY
      total_sum_tasks_used DESC
  ) as user_rank_by_sum_tasks_used
FROM
  total_sum_tasks_used_CTE
  LIMIT 10
"""
df_power_users = pd.read_sql(power_user_query, conn)
df_power_users
```

[16]:

```
   user_id  total_sum_tasks_used  pct_all_sum_tasks_used  \
0   541993              15878183                   3.120
1    19415              12574205                   2.470
2   558906               8282192                   1.630
3   645563               6153468                   1.210
4   103042               3872036                   0.760
5  1830742               3305466                   0.650
6   166842               3136333                   0.620
7   289586               2659815                   0.520
8   617995               2335859                   0.460
9   115417               2174002                   0.430

   user_rank_by_sum_tasks_used
0                            1
1                            2
2                            3
3                            4
4                            5
5                            6
6                            7
7                            8
8                            9
9                           10
```

- Would be interesting to dig deeper into metadata on these users (e.g, age, industry, gender, profession, etc.)

0.2 Timeseries forecasting

- We'll use Prophet to forecast active users and churn into the next 365 days. See https://facebook.github.io/prophet/ if you're not familiar with Prophet.
- Let's run another aggregate query against the active_users_analysis view, this time we'll group by date.

[17]:
```
user_aggregates_query = """
SELECT
  date as ds,
  count(CASE WHEN churned is TRUE then 1 END) as churned_count,
  count(CASE WHEN active is TRUE then 1 END) as active_count,
  ROUND(
    100.0 * count(CASE WHEN churned is TRUE then 1 END)/ count(*),
    2
  ) as churned_ratio,
  ROUND(
    100.0 * count(CASE WHEN active is TRUE then 1 END)/ count(*),
    2
  ) as active_ratio,
  count(*) as total_count
FROM
  jchumley.active_users_analysis
GROUP BY
  date
ORDER BY
  date"""
df_ts = pd.read_sql(user_aggregates_query, conn)
df_ts.head()
```

[17]:
```
           ds  churned_count  active_count  churned_ratio  active_ratio  \
0  2017-01-01              0         38903          0.000       100.000
1  2017-01-02              0         53046          0.000       100.000
2  2017-01-03              0         64350          0.000       100.000
3  2017-01-04              0         65926          0.000       100.000
4  2017-01-05              0         66304          0.000       100.000

   total_count
0        38903
1        53046
2        64350
3        65926
4        66304
```

0.2.1 Forecast of Active Users

- Let's create a timeseries forecast of counts of active users into June 2018.

[18]:
```
df_ts_active = df_ts.copy()
df_ts_active[['ds', 'y']] = df_ts[['ds', 'active_count']]
```

Instantiating and Prophet object and making predictions into June 2018.

[19]:
```
m = Prophet()
m.fit(df_ts_active)
active_users_future = m.make_future_dataframe(periods=365)
active_users_future.tail()
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
```

[19]:

	ds
512	2018-05-28
513	2018-05-29
514	2018-05-30
515	2018-05-31
516	2018-06-01

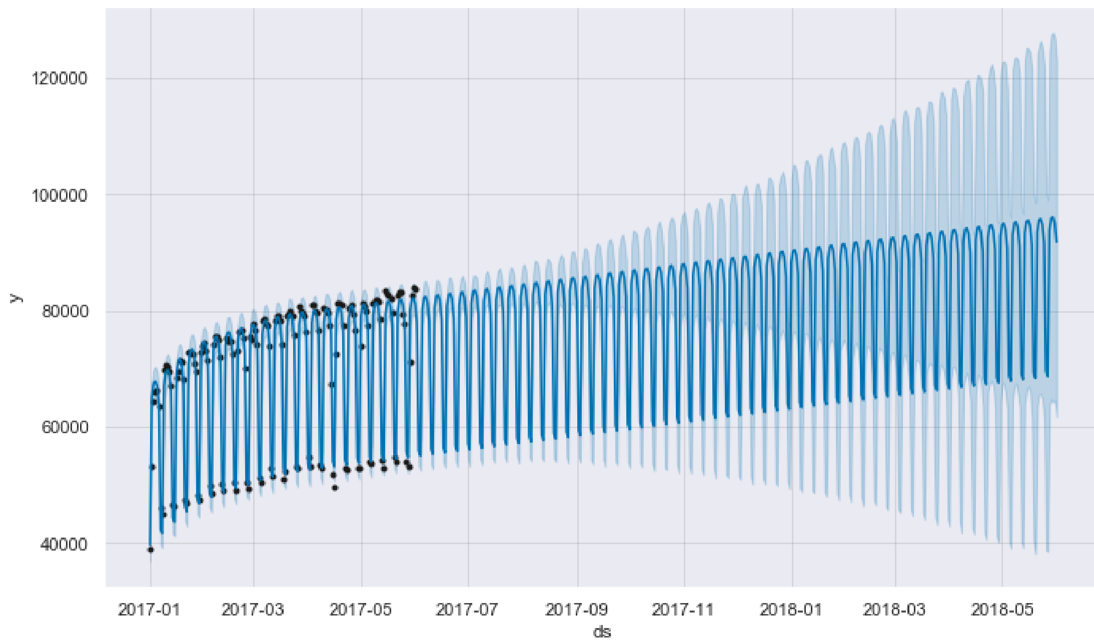
[20]:
```
active_users_forecast = m.predict(active_users_future)
active_users_forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

[20]:

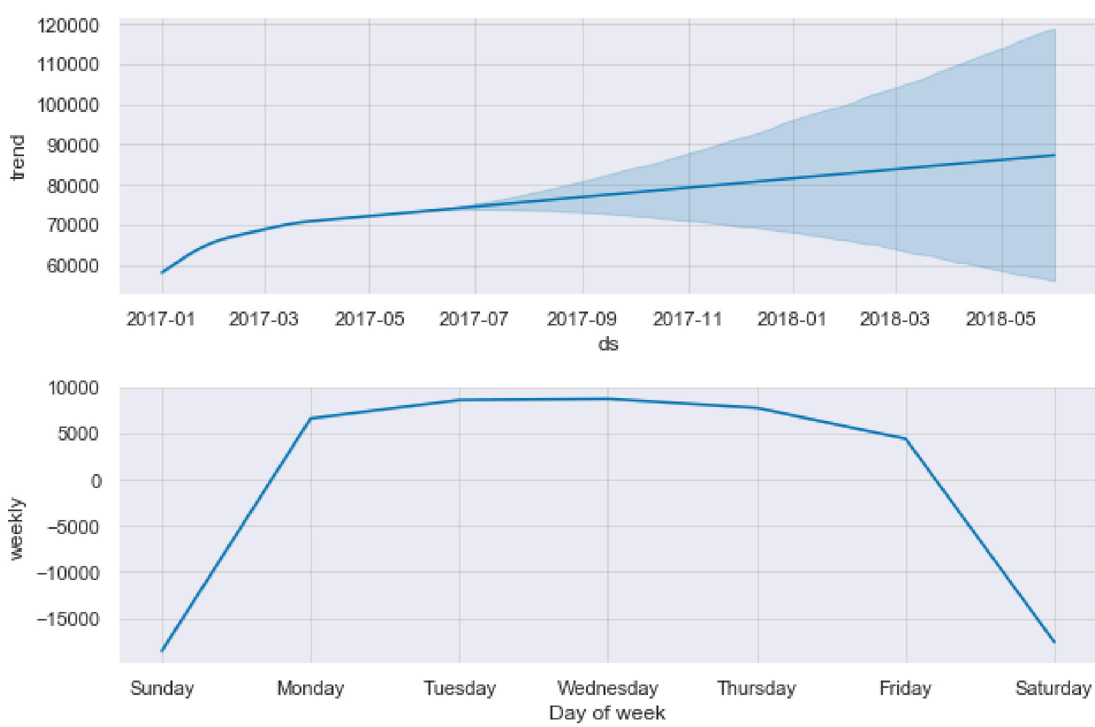
	ds	yhat	yhat_lower	yhat_upper
512	2018-05-28	93785.262	63780.546	124429.538
513	2018-05-29	95813.956	64771.929	127370.172
514	2018-05-30	95986.144	64223.599	127730.788
515	2018-05-31	95036.977	64546.845	126863.978
516	2018-06-01	91740.225	61631.811	122945.292

Visualizing our active users forecast

[22]:
```
fig1_active = m.plot(active_users_forecast)
```



[23]:
```
fig2 = m.plot_components(active_users_forecast)
```



- The model is predicting a steady growth of active users into June 2018.
- Notice that the weekly component aligns with our day of week analysis for active users.

0.2.2 Forecast of Churned Users

Similarly, let's create a timeseries forecast for churn into June 2018.

[24]:
```python
df_ts_churned = df_ts.copy()
df_ts_churned[['ds', 'y']] = df_ts[['ds', 'churned_count']]
df_ts_churned.head()
```

[24]:
```
          ds  churned_count  active_count  churned_ratio  active_ratio  \
0 2017-01-01              0         38903          0.000       100.000
1 2017-01-02              0         53046          0.000       100.000
2 2017-01-03              0         64350          0.000       100.000
3 2017-01-04              0         65926          0.000       100.000
4 2017-01-05              0         66304          0.000       100.000

   total_count  y
0        38903  0
1        53046  0
2        64350  0
3        65926  0
4        66304  0
```

Instantiating and Prophet object and making predictions

[25]:
```python
m2=Prophet()
m2.fit(df_ts_churned)
churned_future = m2.make_future_dataframe(periods=365)
churned_future.tail()
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
```

[25]:
```
            ds
512 2018-05-28
513 2018-05-29
514 2018-05-30
515 2018-05-31
516 2018-06-01
```

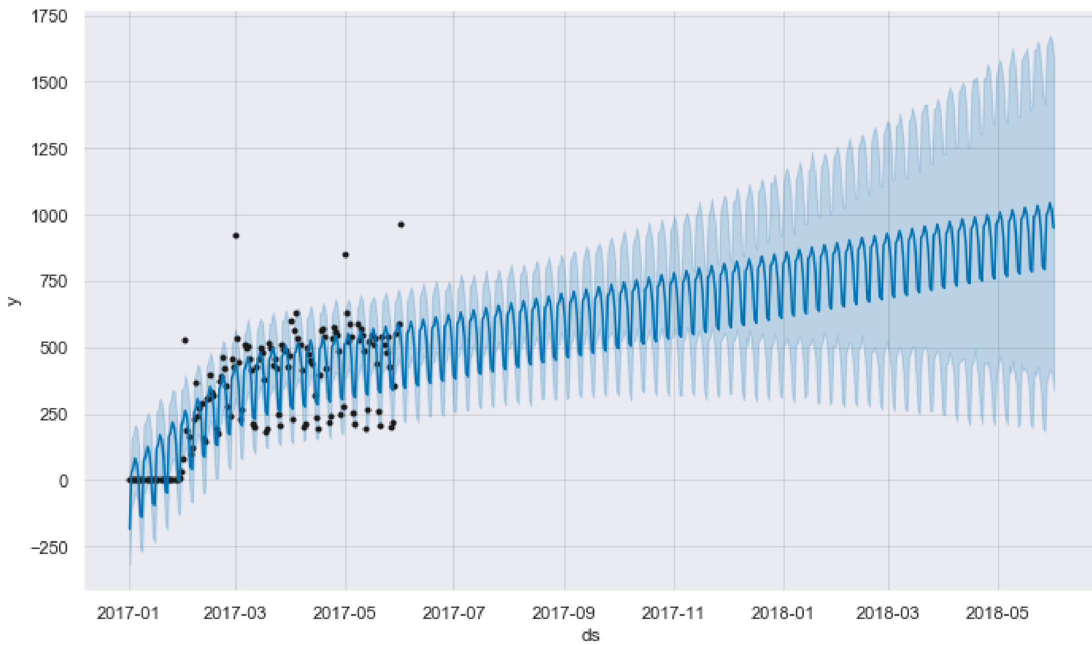
[26]:
```python
churned_user_forecast = m2.predict(churned_future)
churned_user_forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

[26]:
```
            ds     yhat  yhat_lower  yhat_upper
512 2018-05-28  999.565     371.172    1631.167
513 2018-05-29 1011.668     392.231    1652.092
```

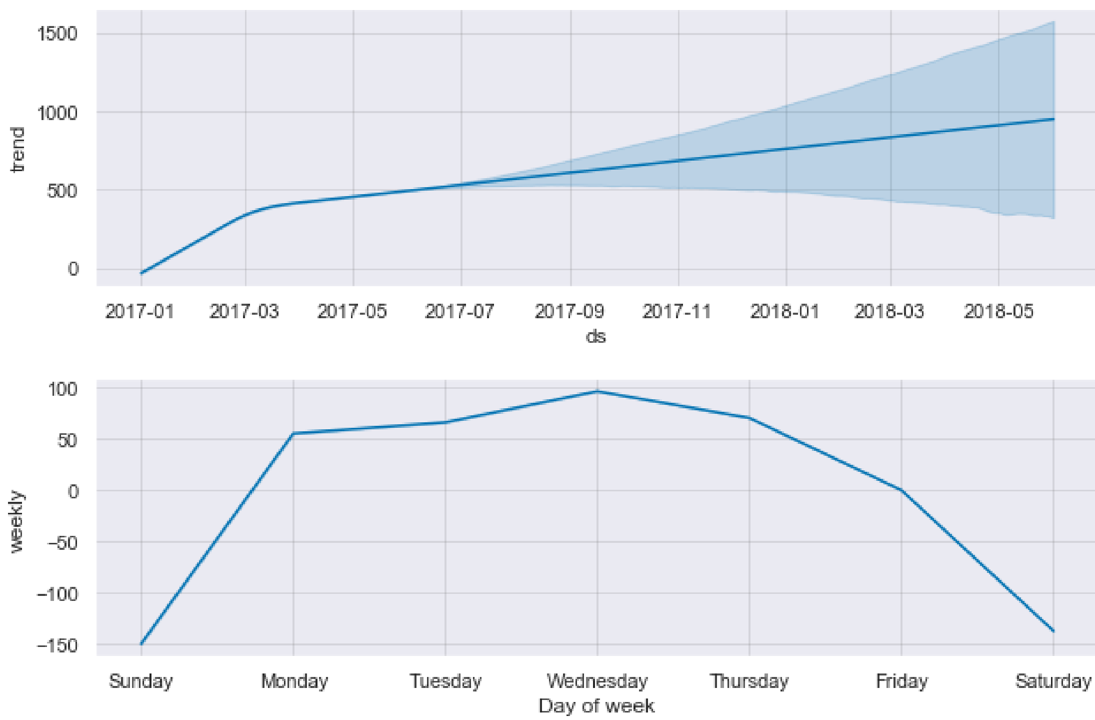
```
514 2018-05-30 1043.238     415.341     1673.703
515 2018-05-31 1018.547     408.432     1653.841
516 2018-06-01  949.444     339.050     1586.101
```

Visualizing out churned users forecast

[31]:
```
fig1_churned = m2.plot(churned_user_forecast)
```



[29]:
```
fig2_churned = m2.plot_components(churned_user_forecast)
```



- Similar to the forecast of active users, churn is predicted to have steady growth into June 2018.
- Unlike the active user plot, we can see outliers in churn at the end of each month. This could be due to a large number of users who are registered under monthly billing subscriptions.

0.2.3 Business recommendations

- Identify new data sources containing Zapier users' metadata (e.g.,registration date, plan type, first usage date, acquisition channel, date of birth, gender, industry, etc.).
- Using additional data source combined with the task_used_da table, develop user metadata features.
- Develop a logistic regression model to generate churn probability scores for each user.
- Perform cohort analysis to group users by shared attributes.
- For users who have churned, send out a survey to understand why they have decided to no longer use the Zapier platform.
- Using the logistic regression model identify high-value power users who are at risks of churn.
- Run A/B testing experiments and compare churn rates between the test and control slices.
- Reach out to users who are on monthly subscription plans prior to the end of the month via email or another communication channel with the goal of customer retention.