codecademy

Will D. Leone

Codeflix Churn Rates

Learn SQL from Scratch

June 21, 2018

Will D. Leone

- Overview of Codeflix
  - Launch Date
  - User Segments
  - Minimum Subscription Length
- What is Churn?
  - Churn Formula
  - Date Range for Churn Calculations
- How to Calculate Codeflix's Churn
  - Consolidating User Subscription Status Data
  - Summarizing This Data with Aggregate Functions
  - Calculating Churn by Segment and Month
- Findings and Recommendations
- Appendix: Building the Local Database

All SQL/Python code and code outputs were created by Will D. Leone.

Overview of Codeflix

Will D. Leone

- **Codeflix Launch:** four months ago (December 2016)
- **User segments:** Two (30 and 87), each of which are sourced from distinct channels
- **Minimum Subscription Length:** 1 month.
  - Users cannot start and end their subscriptions in the same month.
  - This means monthly churn calculations will be accurate.

```
SELECT segment
FROM subscriptions
GROUP BY segment;
```

	segment
1	30
2	87

Figure 0: Subscription segments.

```
SELECT subscription_start AS start_date,
subscription_end AS end_date
FROM subscriptions
ORDER BY start_date, end_date;
```

	start_date	end_date
9	1/1/2017	3/18/2017
10	1/1/2017	3/2/2017
11	1/1/2017	3/24/2017
12	1/1/2017	3/28/2017
13	1/10/2017	<null>
14	1/10/2017	<null>
15	1/10/2017	<null>
16	1/10/2017	<null>
17	1/10/2017	<null>
18	1/10/2017	<null>
19	1/10/2017	<null>
20	1/10/2017	2/19/2017

Figure 1: Subscription start and end dates (excerpt).

Overview of Codeflix

Will D. Leone

- **Churn = Attrition / Initial**
  - Attrition is the users who have unsubscribed in a given month.
  - Initial is users who were subscribed at the start of the month.
- **Date Range:** Churn can only be calculated between January and March in 2017.
  - There were no users before December 2016 (Initial = 0).
  - No data is available after March 2017.

```
SELECT '2017-01-01' AS first_day,
    '2017-01-31' AS last_day
UNION
SELECT '2017-02-01' AS first_day,
  '2017-02-28' AS last_day
UNION
SELECT '2017-03-01' AS first_day,
  '2017-03-31' AS last_day;
```

	first_day	last_day
1	1/1/2017	1/31/2017
2	2/1/2017	2/28/2017
3	3/1/2017	3/31/2017

Figure 2: Subscription `months` table. These are the months for which churn will be calculated.

Summarizing Users' Subscription Status

Will D. Leone

- **Goal:** calculate churn by month and by segment.
- **First Step:** consolidate users' subscription status data so that it can be easily counted.

```sql
WITH months AS (
  SELECT '2017-01-01' AS first_day,
    '2017-01-31' AS last_day
  UNION
  SELECT '2017-02-01' AS first_day,
    '2017-02-28' AS last_day
  UNION
  SELECT '2017-03-01' AS first_day,
    '2017-03-31' AS last_day
), cross_join AS (
  SELECT *
  FROM subscriptions
  CROSS JOIN months
)
SELECT cross_join.id AS id,
  cross_join.first_day AS month,
  subscriptions.segment AS segment,
  CASE WHEN cross_join.subscription_start
        < cross_join.first_day
        AND (cross_join.subscription_end
        > cross_join.first_day
        OR cross_join.subscription_end
        IS NULL)
    THEN 1
    ELSE 0
  END AS is_active,
  CASE WHEN (cross_join.subscription_end
        BETWEEN cross_join.first_day
        AND cross_join.last_day)
      THEN 1
      ELSE 0
  END AS is_canceled
FROM cross_join
LEFT JOIN subscriptions
  ON cross_join.id = subscriptions.id;
```

	id	month	segment	is_active	is_canceled
1	1	2017-01-01	87	1	0
2	1	2017-02-01	87	0	1
3	1	2017-03-01	87	0	0
4	2	2017-01-01	87	1	1
5	2	2017-02-01	87	0	0
6	2	2017-03-01	87	0	0
7	3	2017-01-01	87	1	0
8	3	2017-02-01	87	1	0
9	3	2017-03-01	87	1	1
10	4	2017-01-01	87	1	0
11	4	2017-02-01	87	1	1
12	4	2017-03-01	87	0	0
13	5	2017-01-01	87	1	0
14	5	2017-02-01	87	1	0
15	5	2017-03-01	87	1	1
16	6	2017-01-01	87	1	1
17	6	2017-02-01	87	0	0
18	6	2017-03-01	87	0	0
19	7	2017-01-01	87	1	0
20	7	2017-02-01	87	1	1

Figure 3: `status` table. Indicates when the user was active and when they cancelled.

Subscription Status Aggregate Data

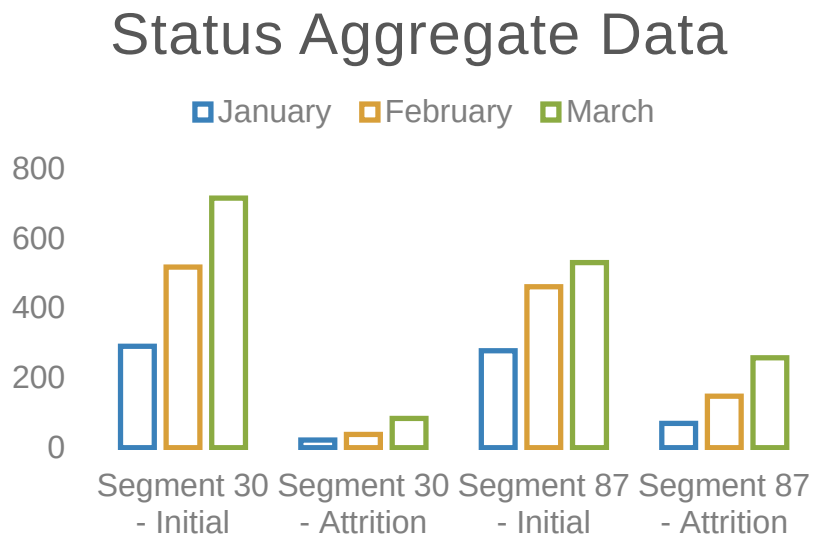
Will D. Leone

- **Goal:** calculate churn by month and by segment.
- **Second Step:** count and group users' subscription statuses by segment and then by month.
- **Initial Analysis:** growth is slowing while attrition is increasing

```
WITH months AS (
  SELECT '2017-01-01' AS first_day,
    '2017-01-31' AS last_day
  UNION
  SELECT '2017-02-01' AS first_day,
    '2017-02-28' AS last_day
  UNION
  SELECT '2017-03-01' AS first_day,
    '2017-03-31' AS last_day
), cross_join AS (
  SELECT *
  FROM subscriptions
  CROSS JOIN months
), status AS (
  SELECT cross_join.id AS id,
    cross_join.first_day AS month,
    subscriptions.segment AS segment,
    CASE WHEN cross_join.subscription_start
         < cross_join.first_day
         AND (cross_join.subscription_end
         > cross_join.first_day
         OR cross_join.subscription_end
         IS NULL)
     THEN 1
     ELSE 0
  END AS is_active,
    CASE WHEN (cross_join.subscription_end
           BETWEEN cross_join.first_day
           AND cross_join.last_day)
         THEN 1
         ELSE 0
    END AS is_canceled
  FROM cross_join
  LEFT JOIN subscriptions
    ON cross_join.id = subscriptions.id
)
SELECT month,
 segment,
 SUM(is_active) AS sum_active,
 SUM(is_canceled) AS sum_canceled
FROM status
GROUP BY segment, month;
```

	month	segment	sum_active	sum_canceled
1	2017-01-01	30	291	22
2	2017-02-01	30	518	38
3	2017-03-01	30	716	84
4	2017-01-01	87	278	70
5	2017-02-01	87	462	148
6	2017-03-01	87	531	258

Figure 4: `status_aggregate` table. Sums and groups the user subscription status data by segment and then by month.



Churn Calculations

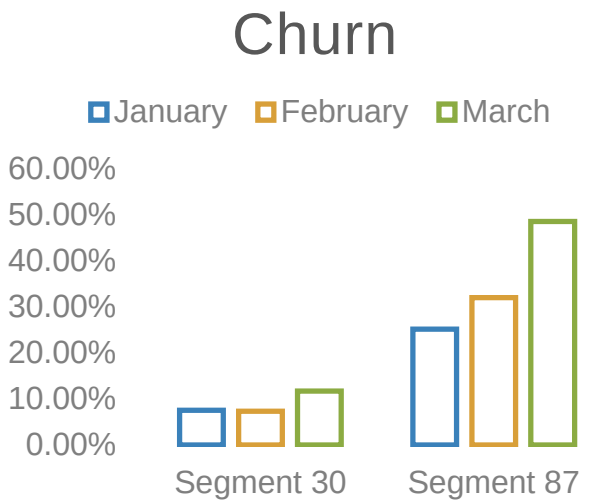
Will D. Leone

- Segment 30 has significantly less churn (7.34% to 11.73% vs. 25.18% to 48.59%).
- Churn sharply increases in March 2017 for both segments.

```sql
WITH months AS (
  SELECT '2017-01-01' AS first_day,
    '2017-01-31' AS last_day
  UNION
  SELECT '2017-02-01' AS first_day,
    '2017-02-28' AS last_day
  UNION
  SELECT '2017-03-01' AS first_day,
    '2017-03-31' AS last_day
), cross_join AS (
  SELECT *
  FROM subscriptions
  CROSS JOIN months
), status AS (
  SELECT cross_join.id AS id,
    cross_join.first_day AS month,
    subscriptions.segment AS segment,
    CASE WHEN cross_join.subscription_start
         < cross_join.first_day
         AND (cross_join.subscription_end
         > cross_join.first_day
         OR cross_join.subscription_end
         IS NULL)
     THEN 1
     ELSE 0
  END AS is_active,
    CASE WHEN (cross_join.subscription_end
           BETWEEN cross_join.first_day
           AND cross_join.last_day)
         THEN 1
         ELSE 0
    END AS is_canceled
  FROM cross_join
  LEFT JOIN subscriptions
    ON cross_join.id = subscriptions.id
), status_aggregate AS (
    SELECT month,
      segment,
      SUM(is_active) AS sum_active,
      SUM(is_canceled) AS sum_canceled
    FROM status
    GROUP BY segment, month
)
SELECT month,
  segment,
  printf("%.2f%%",
    100.0 * sum_canceled/sum_active)
  AS churn
FROM status_aggregate;
```

	month	segment	churn
1	2017-01-01	30	7.56%
2	2017-02-01	30	7.34%
3	2017-03-01	30	11.73%
4	2017-01-01	87	25.18%
5	2017-02-01	87	32.03%
6	2017-03-01	87	48.59%

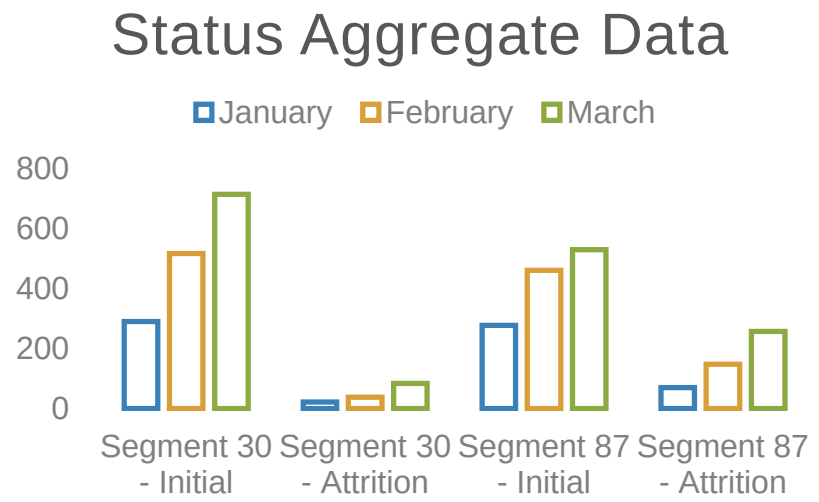
Figure 5: Churn calculations grouped by segment and then by month.

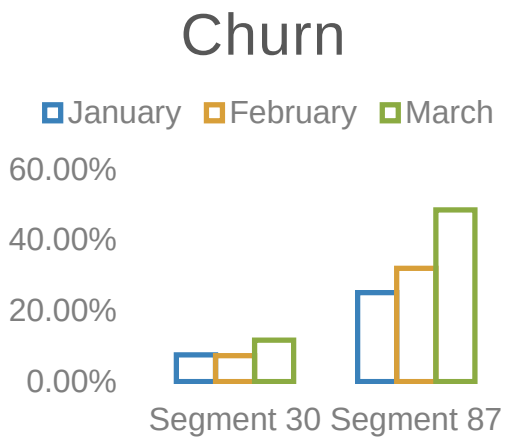


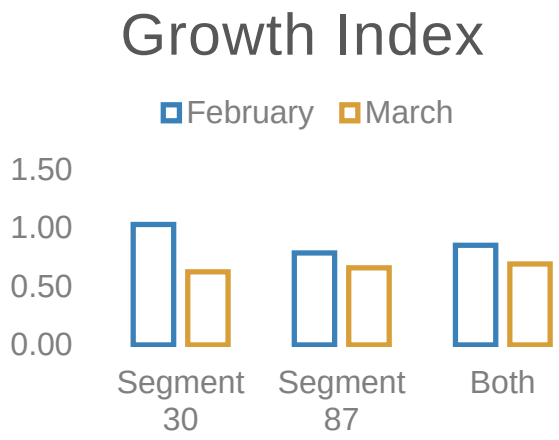
Findings and Recommendations

Will D. Leone

- **Findings**
  - Segment 30:
    - Has significantly less churn.
    - Draws significantly more customers.
  - Both segments:
    - Churn sharply increases between February and March.
    - Growth is slowing while attrition is increasing.
      - Define the *growth index* as (change in Initial)/(change in Attrition) from the prior month to the current month.
      - Segment 30 growth index: 1.03 (February), 0.63 (March)
      - Segment 87 growth index: 0.79 (February), 0.66 (March)
      - Combined growth index: 0.85 (February), 0.69 (March)
- **Recommendations**
  - Determine the cause for March's spike in churn.
  - Re-assess Segment 87 for its effectiveness in attracting customers, especially long-term customers.
  - Review Segment 30 for insight into why it is superior for attracting more customers that are more loyal.







Appendix: Building the Local Database

Will D. Leone

- All code and outputs in this report were run out of a local copy of the database.
  - Data was copied from a `SELECT * FROM subscriptions;` query at Codecademy's website.
  - Copied data was pasted into a CSV file and verified.
  - The Python script shown here ("import_to_mysql3.py") replicates the Codeflix SQLite database ("Codeflix.db").
- See the next slides for the corresponding source code.
- Development Tools
  - Python 3.6.5, sqlite3 (module) 3.23.1
  - IDE: PyCharm 2018.1.4 (Professional Edition).

```
Removed prior version of Codeflix.db.
1380 out of 2000 (69.0%) rows have some null values.
First and last five records:

(1, '2016-12-01', '2017-02-01', 87)
(2, '2016-12-01', '2017-01-24', 87)
(3, '2016-12-01', '2017-03-07', 87)
(4, '2016-12-01', '2017-02-12', 87)
(5, '2016-12-01', '2017-03-09', 87)
(1996, '2017-03-30', None, 30)
(1997, '2017-03-30', None, 30)
(1998, '2017-03-30', None, 30)
(1999, '2017-03-30', None, 30)
(2000, '2017-03-30', None, 30)

2000 total records.

Process finished with exit code 0
```

Figure 6: Output of the Python script used to create the local SQLite database ("Codeflix.db").

Appendix: Building the Local Database

Will D. Leone

```python
import sqlite3
import os


def connect(path, database):
    os.chdir(path)
    return sqlite3.connect(database)


def drop(path, database, table_foo, close=False):
    connection = connect(path, database)
    connection.execute(f"DROP TABLE {table_foo};")
    if close:
        connection.close()
    return


def csv_to_db(path, database, source_csv, close=True):
    connection = connect(path, database)
    connection.execute("CREATE TABLE IF NOT EXISTS subscriptions ( "
                       "  id integer NOT NULL, "
                       "  subscription_start text NOT NULL, "
                       "  subscription_end text, "
                       "  segment integer);")
```

```python
    data = list()
    with open(source_csv, newline='') as csvReader:
        for row in csvReader:
            row = row.split(",")
            for value in row:
                column = row.index(value)
                if not value:
                    if column in [0, 3]:
                        row[column] = 0
                if "/" in value:
                    date = value.split("/")
                    for entry in date:
                        if len(entry) == 1:
                            date[date.index(entry)] = "0" + entry
                    row[column] = date[2] + "-" + date[0] + "-" + date[1]
            data.append([int(row[0]), row[1], row[2], int(row[3])])

    connection.executemany('INSERT INTO subscriptions ( '
                           '  id, '
                           '  subscription_start, '
                           '  subscription_end, '
                           '  segment) '
                           'VALUES (?, ?, ?, ?);', data)
```

Appendix: Building the Local Database

Will D. Leone

```
    for column in ["subscription_end", "segment"]:
        connection.execute(f'UPDATE subscriptions '
                           f'SET {column} = NULL '
                           f'WHERE {column} = "";')
    connection.commit()
    if close:
        connection.close()
    return


def count_rows_with_null(path, database, table_foo):
    connection = connect(path, database)
    total_rows, null_rows = 0, 0
    for row in connection.execute(f'SELECT COUNT(*) '
                                  f'FROM {table_foo};'):
        total_rows = row[0]
    for row in connection.execute(f'SELECT COUNT(*) '
                                  f'FROM {table_foo} '
                                  f'WHERE subscription_end IS NULL;'):
        null_rows = row[0]
    percentage = round(100 * null_rows/total_rows, 2)
    return print(f"{null_rows} out of {total_rows} "
                 f"({percentage}%) rows have some null values.")

```

```
def show_rows(path, database, table_foo):
    connection = connect(path, database)
    data = list()
    try:
        print("First and last five records: \n")
        for row in connection.execute(f'SELECT * '
                                      f'FROM {table_foo};'):
            data.append(row)
    finally:
        for record in data[:5] + data[-5:]:
            print(record)
    print("\n" + str(len(data)) + " total records.")


try:
    drop(os.getcwd(), "Codeflix.db", "subscriptions")
    print("Removed prior version of Codeflix.db.")
finally:
    csv_to_db(os.getcwd(), "Codeflix.db", "Codeflix.csv")
    count_rows_with_null(os.getcwd(), "Codeflix.db", "subscriptions")
    show_rows(os.getcwd(), "Codeflix.db", "subscriptions")

```