Before we get started, a couple of reminders to keep in mind when using iPython notebooks:

- Remember that you can see from the left side of a code cell when it was last run if there is a number within the brackets.
- When you start a new notebook session, make sure you run all of the cells up to the point where you last left off. Even if the output is still visible from when you ran the cells in your previous session, the kernel starts in a fresh state so you'll need to reload the data, etc. on a new session.
- The previous point is useful to keep in mind if your answers do not match what is expected in the lesson's quizzes. Try reloading the data and run all of the processing steps one by one in order to make sure that you are working with the same variables and data that are at each quiz stage.

Load Data from CSVs

In [1]:
```
import unicodecsv

## Longer version of code (replaced with shorter, equivalent version below)

# enrollments = []
# f = open('enrollments.csv', 'rb')
# reader = unicodecsv.DictReader(f)
# for row in reader:
#     enrollments.append(row)
# f.close()

with open('enrollments.csv', 'rb') as f:
    reader = unicodecsv.DictReader(f)
    enrollments = list(reader)
```

In [2]:
```
#####################################
#                 1                 #
#####################################

## Read in the data from daily_engagement.csv and project_submissions.csv
## and store the results in the below variables.
## Then look at the first row of each table.

engagement_filename = 'C:/Users/Saisandeep/Documents/Udacity Data Analyst/data
 analysis process/daily_engagement.csv'
submissions_filename = 'C:/Users/Saisandeep/Documents/Udacity Data Analyst/dat
a analysis process/project_submissions.csv'

# #
# daily_engagement = []
# f = open(engagement_filename, 'rb')
#reader = unicodecsv.DictReader(f)
# for row in reader:
#     daily_engagement.append(row)
#
# print daily_engagement[0]
#
# f.close()

# project_submissions = []
# f = open(submissions_filename, 'rb')
# reader = unicodecsv.DictReader(f)
# for row in reader:
#      project_submissions.append(row)
#
# print project_submissions[0]
#
# f.close()
#

with open(engagement_filename, 'rb') as f:
    reader = unicodecsv.DictReader(f)
    daily_engagement = list(reader)

with open(submissions_filename, 'rb') as f:
    reader = unicodecsv.DictReader(f)
    project_submissions = list(reader)

print daily_engagement[0]
print project_submissions[0]
```

```
{u'lessons_completed': u'0.0', u'num_courses_visited': u'1.0', u'total_minute
s_visited': u'11.6793745', u'projects_completed': u'0.0', u'acct': u'0', u'ut
c_date': u'2015-01-09'}
{u'lesson_key': u'3176718735', u'processing_state': u'EVALUATED', u'account_k
ey': u'256', u'assigned_rating': u'UNGRADED', u'completion_date': u'2015-01-1
6', u'creation_date': u'2015-01-14'}
```

Fixing Data Types

```
In [3]: from datetime import datetime as dt

        # Takes a date as a string, and returns a Python datetime object.
        # If there is no date given, returns None
        def parse_date(date):
            if date == '':
                return None
            else:
                return dt.strptime(date, '%Y-%m-%d')

        # Takes a string which is either an empty string or represents an integer,
        # and returns an int or None.
        def parse_maybe_int(i):
            if i == '':
                return None
            else:
                return int(i)

        # Clean up the data types in the enrollments table
        for enrollment in enrollments:
            enrollment['cancel_date'] = parse_date(enrollment['cancel_date'])
            enrollment['days_to_cancel'] =
        parse_maybe_int(enrollment['days_to_cancel'])
            enrollment['is_canceled'] = enrollment['is_canceled'] == 'True'
            enrollment['is_udacity'] = enrollment['is_udacity'] == 'True'
            enrollment['join_date'] = parse_date(enrollment['join_date'])

        enrollments[0]
```

```
Out[3]: {u'account_key': u'448',
         u'cancel_date': datetime.datetime(2015, 1, 14, 0, 0),
         u'days_to_cancel': 65,
         u'is_canceled': True,
         u'is_udacity': True,
         u'join_date': datetime.datetime(2014, 11, 10, 0, 0),
         u'status': u'canceled'}
```

```
In [4]: # Clean up the data types in the engagement table
        for engagement_record in daily_engagement:
            engagement_record['lessons_completed'] = int(float(engagement_record['less
        ons_completed']))
            engagement_record['num_courses_visited'] = int(float(engagement_record['nu
        m_courses_visited']))
            engagement_record['projects_completed'] = int(float(engagement_record['pro
        jects_completed']))
            engagement_record['total_minutes_visited'] = float(engagement_record['tota
        l_minutes_visited'])
            engagement_record['utc_date'] = parse_date(engagement_record['utc_date'])

        daily_engagement[0]
```

```
Out[4]: {u'acct': u'0',
         u'lessons_completed': 0,
         u'num_courses_visited': 1,
         u'projects_completed': 0,
         u'total_minutes_visited': 11.6793745,
         u'utc_date': datetime.datetime(2015, 1, 9, 0, 0)}
```

```
In [5]: # Clean up the data types in the submissions table
        for submission in project_submissions:
            submission['completion_date'] = parse_date(submission['completion_date'])
            submission['creation_date'] = parse_date(submission['creation_date'])

        project_submissions[0]
```

```
Out[5]: {u'account_key': u'256',
         u'assigned_rating': u'UNGRADED',
         u'completion_date': datetime.datetime(2015, 1, 16, 0, 0),
         u'creation_date': datetime.datetime(2015, 1, 14, 0, 0),
         u'lesson_key': u'3176718735',
         u'processing_state': u'EVALUATED'}
```

Note when running the above cells that we are actively changing the contents of our data variables. If you try to run these cells multiple times in the same session, an error will occur.

Investigating the Data

```
In [6]: #####################################
        #                 3                 #
        #####################################

        ## Rename the "acct" column in the daily_engagement table to "account_key".

        for engagement in daily_engagement:
            engagement['account_key'] = engagement['acct']
            del[engagement['acct']]
```

In [7]:

```python
def unique_students(data):
    unique_students = set()
    for i in data:
        unique_students.add(i['account_key'])
    return unique_students
```

In [8]:
```
#####################################
#                 2                 #
#####################################

def count_rows(data):
    count = 0
    for row in data:
        count = count + 1
    return count

#print count_rows(enrollments)
#print count_rows(daily_engagement)
#print count_rows(project_submissions)

enrollment_num_rows = len(enrollments)             # Replace this with your co
de

# # enrollment_unique_students = set()
# for enrollment in enrollments:
#     enrollment_unique_students.add(enrollment['account_key'])
#
# enrollment_num_unique_students = len(enrollment_unique_students)
#
enrollment_num_unique_students = len(unique_students(enrollments))

engagement_num_rows = len(daily_engagement)             # Replace this with yo
ur code

# engagement_unique_students = set()
# for engagement in daily_engagement:
#     engagement_unique_students.add(engagement['acct'])

# engagement_num_unique_students = len(engagement_unique_students)  # Replace
 this with your code

engagement_num_unique_students = len(unique_students(daily_engagement))

submission_num_rows = len(project_submissions)             # Replace this with
 your code

# submission_unique_students = set()
# for submission in project_submissions:
#     submission_unique_students.add(submission['account_key'])
#
# submission_num_unique_students = len(submission_unique_students)  # Replace
 this with your code

submission_num_unique_students = len(unique_students(project_submissions))

print enrollment_num_rows
print enrollment_num_unique_students
print engagement_num_rows
print engagement_num_unique_students
print submission_num_rows
print submission_num_unique_students

## Find the total number of rows and the number of unique students (account ke
ys)
## in each table.
```

```
1640
1302
136240
1237
3642
743
```

Problems in the Data

In [9]:
```
print daily_engagement[0]
```

```
{u'lessons_completed': 0, u'num_courses_visited': 1, u'total_minutes_visite
d': 11.6793745, u'projects_completed': 0, 'account_key': u'0', u'utc_date': d
atetime.datetime(2015, 1, 9, 0, 0)}
```

Missing Engagement Records

In [10]:
```
#####################################
#                 4                 #
#####################################

## Find any one student enrollments where the student is missing from the dail
y engagement table.
## Output that enrollment.

for enrollment in enrollments:
    if enrollment['account_key'] not in unique_students(daily_engagement):
        print enrollment
        break
```

```
{u'status': u'canceled', u'is_udacity': False, u'is_canceled': True, u'join_d
ate': datetime.datetime(2014, 11, 12, 0, 0), u'account_key': u'1219', u'cance
l_date': datetime.datetime(2014, 11, 12, 0, 0), u'days_to_cancel': 0}
```

Checking for More Problem Records

In [11]:
```
#####################################
#               5                   #
#####################################

## Find the number of surprising data points (enrollments missing from
## the engagement table) that remain, if any.
sup_students = []
for enrollment in enrollments:
    if enrollment['account_key'] not in unique_students(daily_engagement) and
enrollment['days_to_cancel'] != 0:
        sup_students.append(enrollment['account_key'])
#        print enrollment
print sup_students
print len(sup_students)
```

```
[u'1304', u'1304', u'1101']
3
```

Tracking Down the Remaining Problems

In [12]:
```
# Create a set of the account keys for all Udacity test accounts
udacity_test_accounts = set()
for enrollment in enrollments:
    if enrollment['is_udacity']:
        udacity_test_accounts.add(enrollment['account_key'])
len(udacity_test_accounts)
```

Out[12]:
```
6
```

In [13]:
```
# Given some data with an account_key field, removes any records corresponding
 to Udacity test accounts
def remove_udacity_accounts(data):
    non_udacity_data = []
    for data_point in data:
        if data_point['account_key'] not in udacity_test_accounts:
            non_udacity_data.append(data_point)
    return non_udacity_data
```

In [14]:
```
# Remove Udacity test accounts from all three tables
non_udacity_enrollments = remove_udacity_accounts(enrollments)
non_udacity_engagement = remove_udacity_accounts(daily_engagement)
non_udacity_submissions = remove_udacity_accounts(project_submissions)

print len(non_udacity_enrollments)
print len(non_udacity_engagement)
print len(non_udacity_submissions)
```

```
1622
135656
3634
```

Refining the Question

In [15]:
```
#####################################
#                 6                 #
#####################################

## Create a dictionary named paid_students containing all students who either
## haven't canceled yet or who remained enrolled for more than 7 days. The key
s
## should be account keys, and the values should be the date the student enrol
led.

paid_students = {}

for enrollment in non_udacity_enrollments:
    if enrollment['days_to_cancel'] == None or enrollment['days_to_cancel'] >
7:
        account_key = enrollment['account_key']
        enrollment_date = enrollment['join_date']
        if account_key not in paid_students or paid_students[account_key]<enro
llment_date:
            paid_students[account_key] = enrollment_date

print len(paid_students)
```

```
995
```

Getting Data from First Week

In [16]:
```
# Takes a student's join date and the date of a specific engagement record,
# and returns True if that engagement record happened within one week
# of the student joining.
def within_one_week(join_date, engagement_date):
    time_delta = engagement_date - join_date
    return time_delta.days < 7 and time_delta.days >=0
```

In [17]:
```
def remove_free_trail_cancel(data):
    new_data = []
    for data_point in data:
        if data_point['account_key'] in paid_students:
            new_data.append(data_point)
    return new_data
```

In [18]:
```
paid_enrollments = remove_free_trail_cancel(non_udacity_enrollments)
paid_engagement = remove_free_trail_cancel(non_udacity_engagement)
paid_submissions = remove_free_trail_cancel(non_udacity_submissions)

print len(paid_enrollments)
print len(paid_engagement)
print len(paid_submissions)
```
```
1293
134549
3618
```

In [19]:
```
for engagement in paid_engagement:
    if engagement['num_courses_visited']>0:
        engagement['has_visited'] = 1
    else:
        engagement['has_visited'] = 0
```

In [20]:
```
#####################################
#                 7                 #
#####################################

## Create a list of rows from the engagement table including only rows where
## the student is one of the paid students you just found, and the date is wit
hin
## one week of the student's join date.

paid_engagement_in_first_week = []

for engagement in non_udacity_engagement:
    account_key = engagement['account_key']
    if account_key in paid_students:
        join_date = paid_students[account_key]
        engagement_date = engagement['utc_date']
        if within_one_week(join_date, engagement_date):
            paid_engagement_in_first_week.append(engagement)


for i in paid_engagement_in_first_week:
    print i
    break
len(paid_engagement_in_first_week)
```
```
{u'lessons_completed': 0, u'num_courses_visited': 1, 'has_visited': 1, u'tota
l_minutes_visited': 11.6793745, u'projects_completed': 0, 'account_key':
 u'0', u'utc_date': datetime.datetime(2015, 1, 9, 0, 0)}
```

Out[20]:
```
6919
```

Exploring Student Engagement

```
In [21]: from collections import defaultdict

         # Create a dictionary of engagement grouped by student.
         # The keys are account keys, and the values are lists of engagement records.
         def group_data(data, key_name):
             grouped_data = defaultdict(list)
             for data_point in data:
                 key = data_point[key_name]
                 grouped_data[key].append(data_point)
             return grouped_data

         engagement_by_account = group_data(paid_engagement_in_first_week, 'account_ke
         y')

         for i in engagement_by_account:
             print i
             print engagement_by_account[i]
             break
```

```
1200
[{u'lessons_completed': 1, u'num_courses_visited': 2, 'has_visited': 1, u'tot
al_minutes_visited': 114.853432, u'projects_completed': 0, 'account_key': u'1
200', u'utc_date': datetime.datetime(2015, 3, 4, 0, 0)}, {u'lessons_complete
d': 0, u'num_courses_visited': 1, 'has_visited': 1, u'total_minutes_visited':
43.4168625, u'projects_completed': 0, 'account_key': u'1200', u'utc_date': da
tetime.datetime(2015, 3, 5, 0, 0)}, {u'lessons_completed': 0, u'num_courses_v
isited': 1, 'has_visited': 1, u'total_minutes_visited': 187.776832833, u'proj
ects_completed': 0, 'account_key': u'1200', u'utc_date': datetime.datetime(20
15, 3, 6, 0, 0)}, {u'lessons_completed': 0, u'num_courses_visited': 1, 'has_v
isited': 1, u'total_minutes_visited': 150.081577333, u'projects_completed':
 0, 'account_key': u'1200', u'utc_date': datetime.datetime(2015, 3, 7, 0,
 0)}, {u'lessons_completed': 0, u'num_courses_visited': 1, 'has_visited': 1,
 u'total_minutes_visited': 191.61088, u'projects_completed': 0, 'account_ke
y': u'1200', u'utc_date': datetime.datetime(2015, 3, 8, 0, 0)}, {u'lessons_co
mpleted': 0, u'num_courses_visited': 0, 'has_visited': 0, u'total_minutes_vis
ited': 0.0, u'projects_completed': 0, 'account_key': u'1200', u'utc_date': da
tetime.datetime(2015, 3, 9, 0, 0)}, {u'lessons_completed': 0, u'num_courses_v
isited': 1, 'has_visited': 1, u'total_minutes_visited': 8.83762516667, u'proj
ects_completed': 0, 'account_key': u'1200', u'utc_date': datetime.datetime(20
15, 3, 10, 0, 0)}]
```

In [22]:
```
# Create a dictionary with the total minutes each student spent in the classro
om during the first week.
# The keys are account keys, and the values are numbers (total minutes)
total_minutes_by_account = {}
for account_key, engagement_for_student in engagement_by_account.items():
    total_minutes = 0
    for engagement_record in engagement_for_student:
        total_minutes += engagement_record['total_minutes_visited']
    total_minutes_by_account[account_key] = total_minutes

# for account_key, engagement_for_student in engagement_by_account.items():
#     for engagement_record in engagement_for_student:
#         print account_key
#         print engagement_record
#     break
```

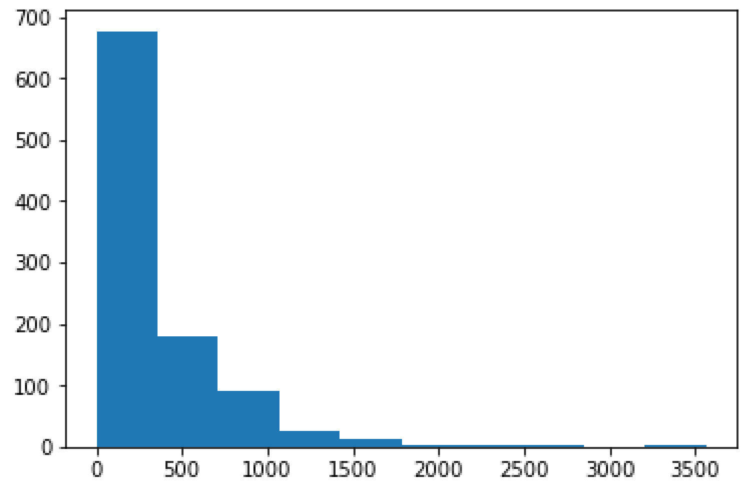
In [23]:
```
def sum_grouped_items(grouped_data, field_name):
    summed_data = {}
    for key, data_points in grouped_data.items():
        total = 0
        for data_point in data_points:
            total += data_point[field_name]
        summed_data[key] = total
    return summed_data

total_minutes_by_account = sum_grouped_items(engagement_by_account, 'total_min
utes_visited')
```

In [24]:
```
for i in total_minutes_by_account:
    print i
    print total_minutes_by_account[i]
    break
```

```
619
1482.90204567
```

```
In [47]: %matplotlib inline
         import matplotlib.pyplot as plt
         import numpy as np

         # Summarize the data about minutes spent in the classroom

         def describe_data(data):
             print 'Mean:', np.mean(data)
             print 'Standard deviation:', np.std(data)
             print 'Minimum:', np.min(data)
             print 'Maximum:', np.max(data)
             plt.hist(data)

         total_minutes = total_minutes_by_account.values()
         describe_data(total_minutes)
```

```
Mean: 306.708326753
Standard deviation: 412.996933409
Minimum: 0.0
Maximum: 3564.7332645
```



Debugging Data Analysis Code

In [26]:
```
#####################################
#               8                   #
#####################################

## Go through a similar process as before to see if there is a problem.
## Locate at least one surprising piece of data, output it, and take a look at
 it.

# for account_key in total_minutes_by_account:
#     if total_minutes_by_account[account_key] > 10080:
#         print account_key, total_minutes_by_account[account_key]
# print engagement_by_account['108']

student_with_max_minutes = None
max_minutes = 0
for student, total_minutes in total_minutes_by_account.items():
    if total_minutes > max_minutes:
        max_minutes = total_minutes
        student_with_max_minutes = student

max_minutes

for engagement in paid_engagement_in_first_week:
    if engagement['account_key'] == student_with_max_minutes:
        print engagement
```

```
{u'lessons_completed': 4, u'num_courses_visited': 4, 'has_visited': 1, u'tota
l_minutes_visited': 850.519339666, u'projects_completed': 0, 'account_key':
 u'163', u'utc_date': datetime.datetime(2015, 7, 9, 0, 0)}
{u'lessons_completed': 6, u'num_courses_visited': 6, 'has_visited': 1, u'tota
l_minutes_visited': 872.633923334, u'projects_completed': 0, 'account_key':
 u'163', u'utc_date': datetime.datetime(2015, 7, 10, 0, 0)}
{u'lessons_completed': 6, u'num_courses_visited': 2, 'has_visited': 1, u'tota
l_minutes_visited': 777.018903666, u'projects_completed': 0, 'account_key':
 u'163', u'utc_date': datetime.datetime(2015, 7, 11, 0, 0)}
{u'lessons_completed': 2, u'num_courses_visited': 1, 'has_visited': 1, u'tota
l_minutes_visited': 294.568774, u'projects_completed': 0, 'account_key': u'16
3', u'utc_date': datetime.datetime(2015, 7, 12, 0, 0)}
{u'lessons_completed': 1, u'num_courses_visited': 3, 'has_visited': 1, u'tota
l_minutes_visited': 471.2139785, u'projects_completed': 0, 'account_key': u'1
63', u'utc_date': datetime.datetime(2015, 7, 13, 0, 0)}
{u'lessons_completed': 1, u'num_courses_visited': 2, 'has_visited': 1, u'tota
l_minutes_visited': 298.778345333, u'projects_completed': 0, 'account_key':
 u'163', u'utc_date': datetime.datetime(2015, 7, 14, 0, 0)}
{u'lessons_completed': 0, u'num_courses_visited': 0, 'has_visited': 0, u'tota
l_minutes_visited': 0.0, u'projects_completed': 0, 'account_key': u'163', u'u
tc_date': datetime.datetime(2015, 7, 15, 0, 0)}
```

Lessons Completed in First Week

In [27]:
```
######################################
#               9                    #
######################################

## Adapt the code above to find the mean, standard deviation, minimum, and max
imum for
## the number of lessons completed by each student during the first week. Try
 creating
## one or more functions to re-use the code above.

def stats_data(data, item):
    total_by_account = {}
    for account_key, engagement_for_student in data.items():
        total = 0
        for engagement in engagement_for_student:
            total += engagement[item]
        total_by_account[account_key] = total

    totals_for_account = total_by_account.values()
    print 'Mean:', np.mean(totals_for_account)
    print 'Standard deviation:', np.std(totals_for_account)
    print 'Minimum:', np.min(totals_for_account)
    print 'Maximum:', np.max(totals_for_account)
```

In [28]:
```
lessons_completed_by_account = sum_grouped_items(engagement_by_account, 'lesso
ns_completed')
describe_data(lessons_completed_by_account.values())
```

```
Mean: 1.63618090452
Standard deviation: 3.00256129983
Minimum: 0
Maximum: 36
```

In [29]:
```
stats_data(engagement_by_account, 'lessons_completed')
```

```
Mean: 1.63618090452
Standard deviation: 3.00256129983
Minimum: 0
Maximum: 36
```

In [30]:
```
def stats_visit(data):
    total_by_account = {}
    for account_key, engagement_for_student in data.items():
        total = 0
        for engagement in engagement_for_student:
            if engagement['num_courses_visited']>0:
                total += 1
        total_by_account[account_key] = total

    totals_for_account = total_by_account.values()
    print 'Mean:', np.mean(totals_for_account)
    print 'Standard deviation:', np.std(totals_for_account)
    print 'Minimum:', np.min(totals_for_account)
    print 'Maximum:', np.max(totals_for_account)
```

Number of Visits in First Week

```
In [31]: ######################################
         #                10                  #
         ######################################

         ## Find the mean, standard deviation, minimum, and maximum for the number of
         ## days each student visits the classroom during the first week.
         stats_visit(engagement_by_account)
         stats_data(engagement_by_account, 'has_visited')
         days_visited_by_account = sum_grouped_items(engagement_by_account, 'has_visite
         d')
         describe_data(days_visited_by_account.values())
```

```
Mean: 2.86733668342
Standard deviation: 2.25519800292
Minimum: 0
Maximum: 7
Mean: 2.86733668342
Standard deviation: 2.25519800292
Minimum: 0
Maximum: 7
Mean: 2.86733668342
Standard deviation: 2.25519800292
Minimum: 0
Maximum: 7
```

Splitting out Passing Students

```
In [32]: #######################################
         #              11                     #
         #######################################

         ## Create two lists of engagement data for paid students in the first week.
         ## The first list should contain data for students who eventually pass the
         ## subway project, and the second list should contain data for students
         ## who do not.

         subway_project_lesson_keys = ['746169184', '3176718735']

         accounts_passing = []
         accounts_not_passing = []

         for record in paid_submissions:
             if record['lesson_key'] == '746169184' or record['lesson_key'] == '3176718
         735':
                 if record['assigned_rating'] == 'PASSED' or record['assigned_rating']
         == 'DISTINCTION':
                     accounts_passing.append(record['account_key'])
                 else:
                     accounts_not_passing.append(record['account_key'])

         passing_engagement = []
         non_passing_engagement = []
         for engagement in paid_engagement_in_first_week:
             if engagement['account_key'] in accounts_passing:
                 passing_engagement.append(engagement)
             else:
                 non_passing_engagement.append(engagement)

         print len(passing_engagement)
         print len(non_passing_engagement)

         4527
         2392
```

```
In [33]: passing_engagement[0]

Out[33]: {'account_key': u'0',
          'has_visited': 1,
          u'lessons_completed': 0,
          u'num_courses_visited': 1,
          u'projects_completed': 0,
          u'total_minutes_visited': 11.6793745,
          u'utc_date': datetime.datetime(2015, 1, 9, 0, 0)}
```

Comparing the Two Student Groups

In [36]:

```
######################################
#                12                  #
######################################

## Compute some metrics you're interested in and see how they differ for
## students who pass the subway project vs. students who don't. A good
## starting point would be the metrics we looked at earlier (minutes spent
## in the classroom, lessons completed, and days visited).

passing_by_account = group_data(passing_engagement, 'account_key')

minutes_spent_passing_account = sum_grouped_items(passing_by_account, 'total_m
inutes_visited')
describe_data(minutes_spent_passing_account.values())

lessons_completed_passing_account = sum_grouped_items(passing_by_account, 'les
sons_completed')
describe_data(lessons_completed_passing_account.values())

days_visited_by_passing_account = sum_grouped_items(passing_by_account, 'has_v
isited')
describe_data(days_visited_by_passing_account.values())

non_passing_by_account = group_data(non_passing_engagement, 'account_key')

minutes_spent_non_passing_account = sum_grouped_items(non_passing_by_account,
'total_minutes_visited')
describe_data(minutes_spent_non_passing_account.values())

lessons_completed_non_passing_account = sum_grouped_items(non_passing_by_accou
nt, 'lessons_completed')
describe_data(lessons_completed_non_passing_account.values())

days_visited_by_non_passing_account =
sum_grouped_items(non_passing_by_account, 'has_visited')
describe_data(days_visited_by_non_passing_account.values())
```

```
Mean: 394.586046484
Standard deviation: 448.499519327
Minimum: 0.0
Maximum: 3564.7332645
Mean: 2.05255023184
Standard deviation: 3.14222705558
Minimum: 0
Maximum: 36
Mean: 3.38485316847
Standard deviation: 2.25882147092
Minimum: 0
Maximum: 7
Mean: 143.326474267
Standard deviation: 269.538619011
Minimum: 0.0
Maximum: 1768.52274933
Mean: 0.862068965517
Standard deviation: 2.54915994183
Minimum: 0
Maximum: 27
Mean: 1.90517241379
Standard deviation: 1.90573144136
Minimum: 0
Maximum: 7
```

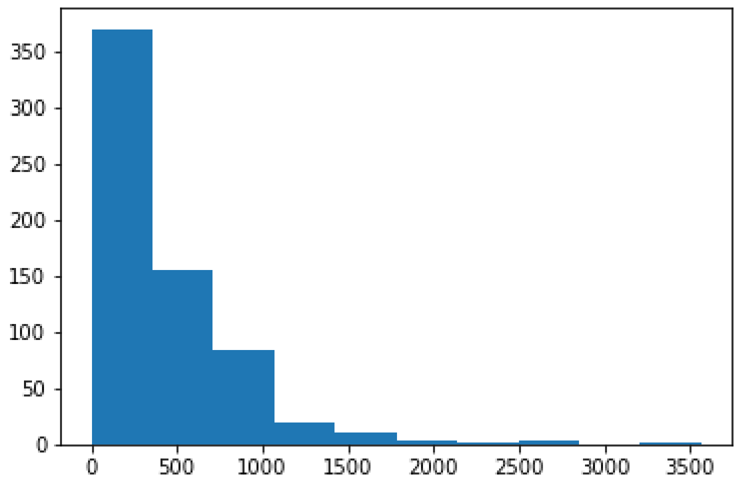
```
In [37]: daily_engagement_by_account = group_data(daily_engagement, 'account_key')

         minutes_spent_daily_account = sum_grouped_items(daily_engagement_by_account,
         'total_minutes_visited')
         describe_data(minutes_spent_daily_account.values())

         lessons_completed_daily_account = sum_grouped_items(daily_engagement_by_accoun
         t, 'lessons_completed')
         describe_data(lessons_completed_daily_account.values())
```

```
Mean: 2704.05890674
Standard deviation: 3144.09358874
Minimum: 0.0
Maximum: 25114.2245857
Mean: 14.1851253032
Standard deviation: 16.0171847464
Minimum: 0
Maximum: 75

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-37-fa3e9d047d0a> in <module>()
      7 describe_data(lessons_completed_daily_account.values())
      8 
----> 9 days_visited_by_daily_account = sum_grouped_items(daily_engagement_by
_account, 'has_visited')
     10 describe_data(days_visited_by_daily_account.values())

<ipython-input-23-b7a55d2b45c6> in sum_grouped_items(grouped_data, field_nam
e)
      4         total = 0
      5         for data_point in data_points:
----> 6             total += data_point[field_name]
      7         summed_data[key] = total
      8     return summed_data

KeyError: 'has_visited'
```

Making Histograms

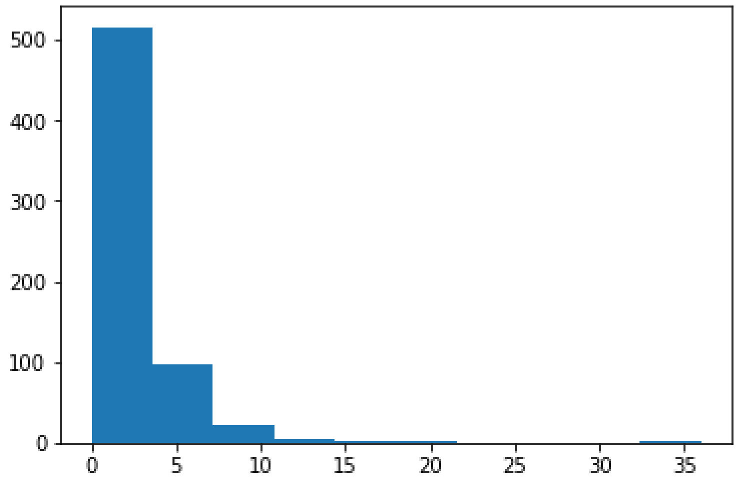
```
In [40]: ######################################
         #                 13                 #
         ######################################

         ## Make histograms of the three metrics we looked at earlier for both
         ## students who passed the subway project and students who didn't. You
         ## might also want to make histograms of any other metrics you examined.

         %matplotlib inline
         import matplotlib.pyplot as plt
         plt.hist(minutes_spent_passing_account.values())
```

```
Out[40]: (array([ 370.,  155.,   83.,   19.,   10.,    4.,    2.,    3.,    0.,
           1.]),
          array([    0.        ,   356.47332645,   712.9466529 ,  1069.41997935,
                  1425.8933058 ,  1782.36663225,  2138.8399587 ,  2495.31328515,
                  2851.7866116 ,  3208.25993805,  3564.7332645 ]),
          <a list of 10 Patch objects>)
```



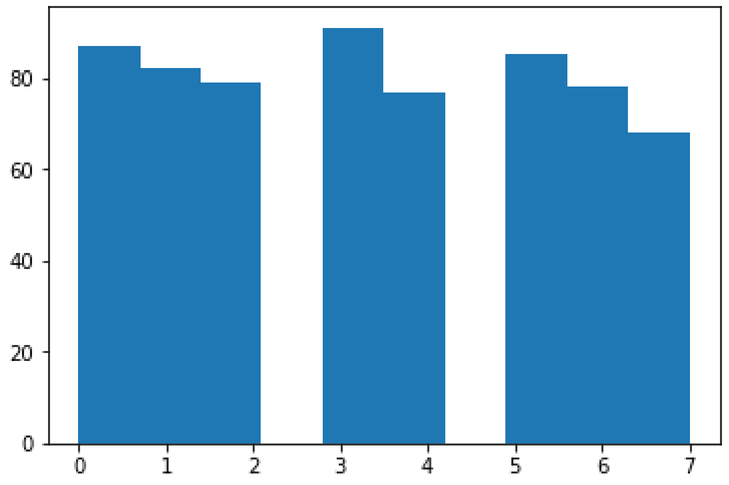
```
In [41]: %matplotlib inline
         import matplotlib.pyplot as plt
         plt.hist(lessons_completed_passing_account.values())
```

```
Out[41]: (array([ 516.,   97.,   23.,    4.,    3.,    3.,    0.,    0.,    0.,
           1.]),
          array([  0. ,   3.6,   7.2,  10.8,  14.4,  18. ,  21.6,  25.2,  28.8,
                 32.4,  36. ]),
          <a list of 10 Patch objects>)
```



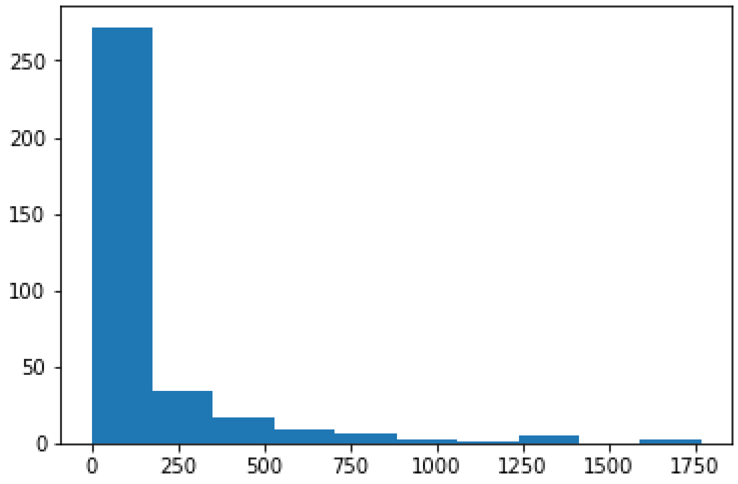
```
In [42]: %matplotlib inline
         import matplotlib.pyplot as plt
         plt.hist(days_visited_by_passing_account.values())
```

```
Out[42]: (array([ 87.,  82.,  79.,   0.,  91.,  77.,   0.,  85.,  78.,  68.]),
          array([ 0. ,  0.7,  1.4,  2.1,  2.8,  3.5,  4.2,  4.9,  5.6,  6.3,  7. ]),
          <a list of 10 Patch objects>)
```



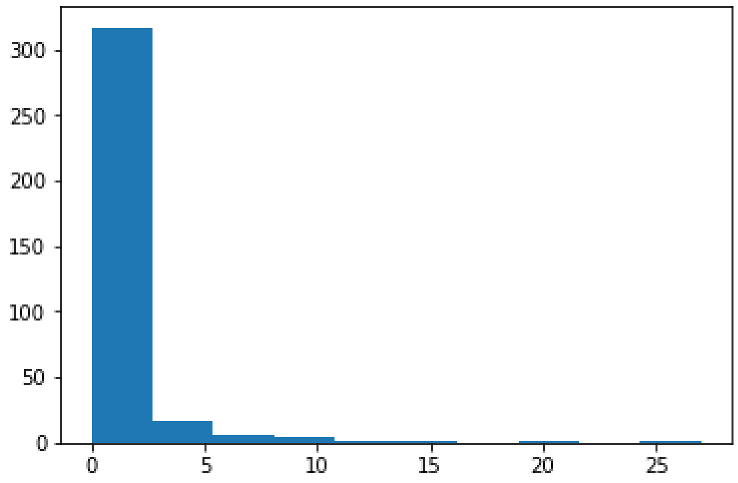
```
In [43]: %matplotlib inline
         import matplotlib.pyplot as plt
         plt.hist(minutes_spent_non_passing_account.values())
```

```
Out[43]: (array([ 272.,   34.,   17.,    9.,    6.,    2.,    1.,    5.,    0.,
           2.]),
          array([    0.        ,   176.85227493,   353.70454987,   530.5568248 ,
                   707.40909973,   884.26137467,  1061.1136496 ,  1237.96592453,
                  1414.81819947,  1591.6704744 ,  1768.52274933]),
          <a list of 10 Patch objects>)
```



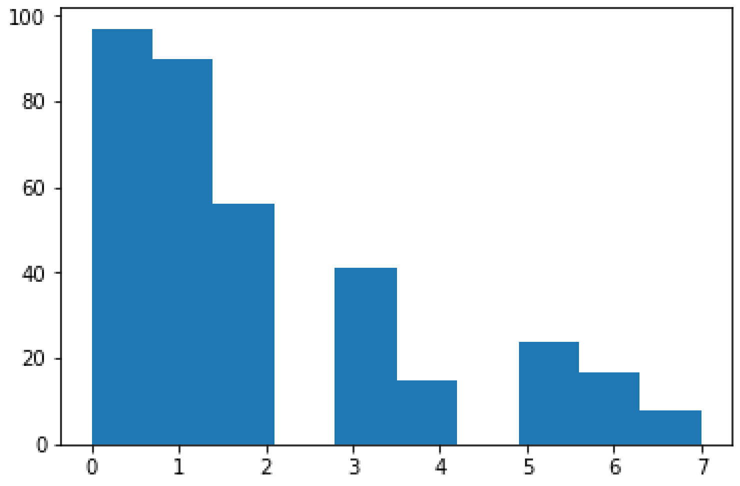
```
In [44]: %matplotlib inline
         import matplotlib.pyplot as plt
         plt.hist(lessons_completed_non_passing_account.values())
```

```
Out[44]: (array([ 317.,   17.,    6.,    4.,    1.,    1.,    0.,    1.,    0.,
           1.]),
          array([  0. ,   2.7,   5.4,   8.1,  10.8,  13.5,  16.2,  18.9,  21.6,
                  24.3,  27. ]),
          <a list of 10 Patch objects>)
```



```
In [45]: %matplotlib inline
         import matplotlib.pyplot as plt
         plt.hist(days_visited_by_non_passing_account.values())
```

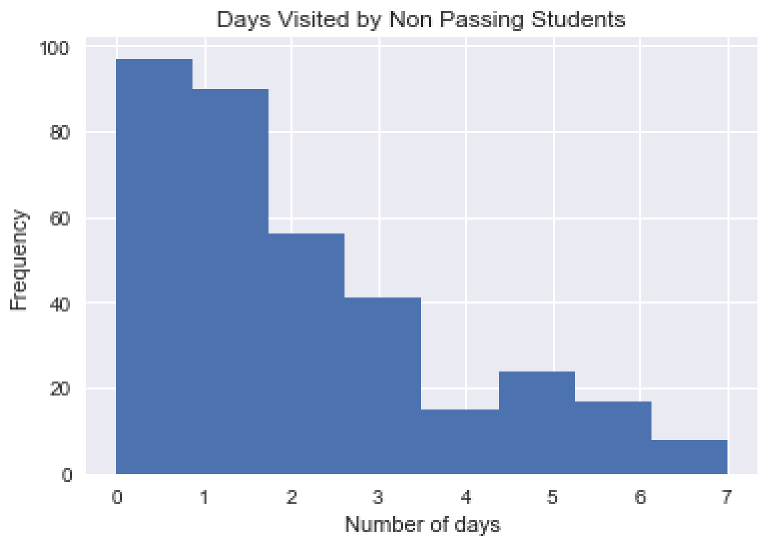
```
Out[45]: (array([ 97.,  90.,  56.,   0.,  41.,  15.,   0.,  24.,  17.,   8.]),
          array([ 0. ,  0.7,  1.4,  2.1,  2.8,  3.5,  4.2,  4.9,  5.6,  6.3,  7. ]),
          <a list of 10 Patch objects>)
```



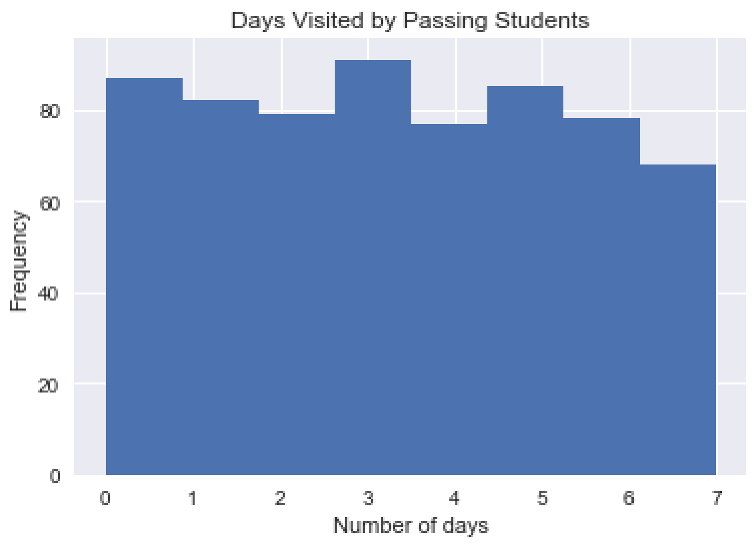
Improving Plots and Sharing Findings

In [53]:
```
######################################
#                 14                 #
######################################

## Make a more polished version of at least one of your visualizations
## from earlier. Try importing the seaborn library to make the visualization
## look better, adding axis labels and a title, and changing one or more
## arguments to the hist() function.

import seaborn as sns
%matplotlib inline
plt.hist(days_visited_by_non_passing_account.values(), bins = 8)
plt.title('Days Visited by Non Passing Students')
plt.xlabel('Number of days')
plt.ylabel('Frequency')
```

Out[53]:
```
<matplotlib.text.Text at 0x1abed9b0>
```



```
In [54]: %matplotlib inline
         plt.hist(days_visited_by_passing_account.values(), bins = 8)
         plt.title('Days Visited by Passing Students')
         plt.xlabel('Number of days')
         plt.ylabel('Frequency')

Out[54]: <matplotlib.text.Text at 0x1b8bef28>
```



```
In [ ]:
```