Python for Data Science
Day 2

By Craig Sakuma

Schedule - Day 2

Time	Topic
10:00 – 11:00	Review Python Fundamentals
11:00 – 12:30	Numpy
12:30 – 1:30	Lunch
1:30 – 4:00	Pandas
4:00 – 5:00	Twitter API

Review Exercises

Open the Review Python Fundamentals ipython notebook

Complete the exercises with a partner

Intro to Numpy

- Numpy is a package for scientific computing
- Arrays are the data structure used in numpy
- Arrays can be multi-dimensional
- Format for data used by scikit-learn
- Contains wide variety of advanced math functions

Creating Arrays

Import the numpy package

```
import numpy as np
```

Make an array from a list

```
np.array([1, 2, 3])
```

Make an array from a list of lists

```
np.array([[1, 2], [3, 4]])
```

Create Arrays with Functions

Make an array using a function

```
np.zeros((2, 3))
```

```
np.ones((3, 2))
```

Make an array by starting at 0, incrementing by 2 and ending before 10

```
np.arange(0,10,2)
```

Creating Advanced Arrays

Make an array of four evenly spaced numbers including both endpoints

```
np.linspace(0, 1, 4)
```

Make an array of random numbers (using a normal distribution)

```
mean = 10
stdev = 3
number_values = 20
np.random.normal(mean, stdev, number_values)
```

Indexing Arrays

Index arrays

```
x = np.arange(4)
print x[0]
print x[-1]
```

Reshape an array

```
x = np.arange(6).reshape(2, 3)
print x
```

Get the last row

```
print x[-1]
```

Indexing Arrays

Get the last element of the last row

```
print x[-1, -1]
```

Get elements like a list

```
x = np.arange(10)
print x
print x[:3]
print x[1:4]
print x[-3:]
```

Indexing Arrays

Get all elements in reverse order

```
print x[::-1]
```

Get every second element or third element

```
print x[::2]
print x[::3]
```

Create a 5x5 matrix

```
x = np.arange(25).reshape(5, 5)
print x
```

Indexing Arrays

Get the first 2 columns and the first 2 rows

```
print x[:2, :2]
```

Get every second column of every second row

```
print x[::2, ::2]
```

Generate six random integers 0 -9 inclusive

```
x = np.random.randint(0, 10, 6)
print x
```

Indexing Arrays

Get the first number and the last two numbers

```
indices = [0, -2, -1]
print x[indices]
```

Create two arrays

```
arrayLength = 5
x1 = np.arange(arrayLength)
x2 = np.arange(arrayLength) * -1
print x1
print x2
```

Indexing Arrays

Create a random shuffled range of numbers

```
indices = np.random.permutation(arrayLength)
print indices
```

Use indices for re-ordering arrays

```
print x1[indices]
print x2[indices]
```

Indexing Arrays

Create array of boolean values

```
x = np.arange(5)
print x
print x > 2
```

Use boolean values to index arrays

```
print x[x > 2]
print x[np.array([False,False,False,True,True])]
```

Indexing Arrays

Create 4 dimensional array

```
x = np.arange(16).reshape(2, 2, 2, 2)
print x
```

Index the array (try turning comments on/off)

```
print x[0]
# print x[0,0]
# print x[0,0,0]
# print x[0,0,0,0]
# print x[0,1,1,0]
```

Array Math

Try adding to a list

```
x = [1, 2, 3, 4]
print x + 10
```

Try again with an array

```
x = np.array([1,2,3,4])
print x + 10
```

Flatten an Array

Create a 2 dimensional array

```
y = np.arange(9).reshape(3,3)
print y
```

Use ravel to flatten an array

```
y.ravel()
```

Exercises

- Create 1-dimensional array from 0 to 26 and assign it to x
- Reverse the order of x and assign it to x_reverse
- Convert x to a 3-dimensional array and assign it to x_3d
- How would you index only the value 12 from x_3d?
- Convert x_3d to a 1-dimensional array
- Create a random shuffled array of values from 0-19 and assign it to y
- Create a boolean array for values in y that are greater than 10
- Create a random sample of 20 data points from a normal distribution with mean of 10 and standard deviation 5

Intro to Pandas

- Primary objects in Pandas are DataFrames
- DataFrames are like tables
  - Contain rows and columns of data
  - Columns have names
  - Rows have index values
- Pandas has easy functions for importing and exporting data
  - CSV files
  - Excel spreadsheets
  - SQL queries

Indexing in Pandas

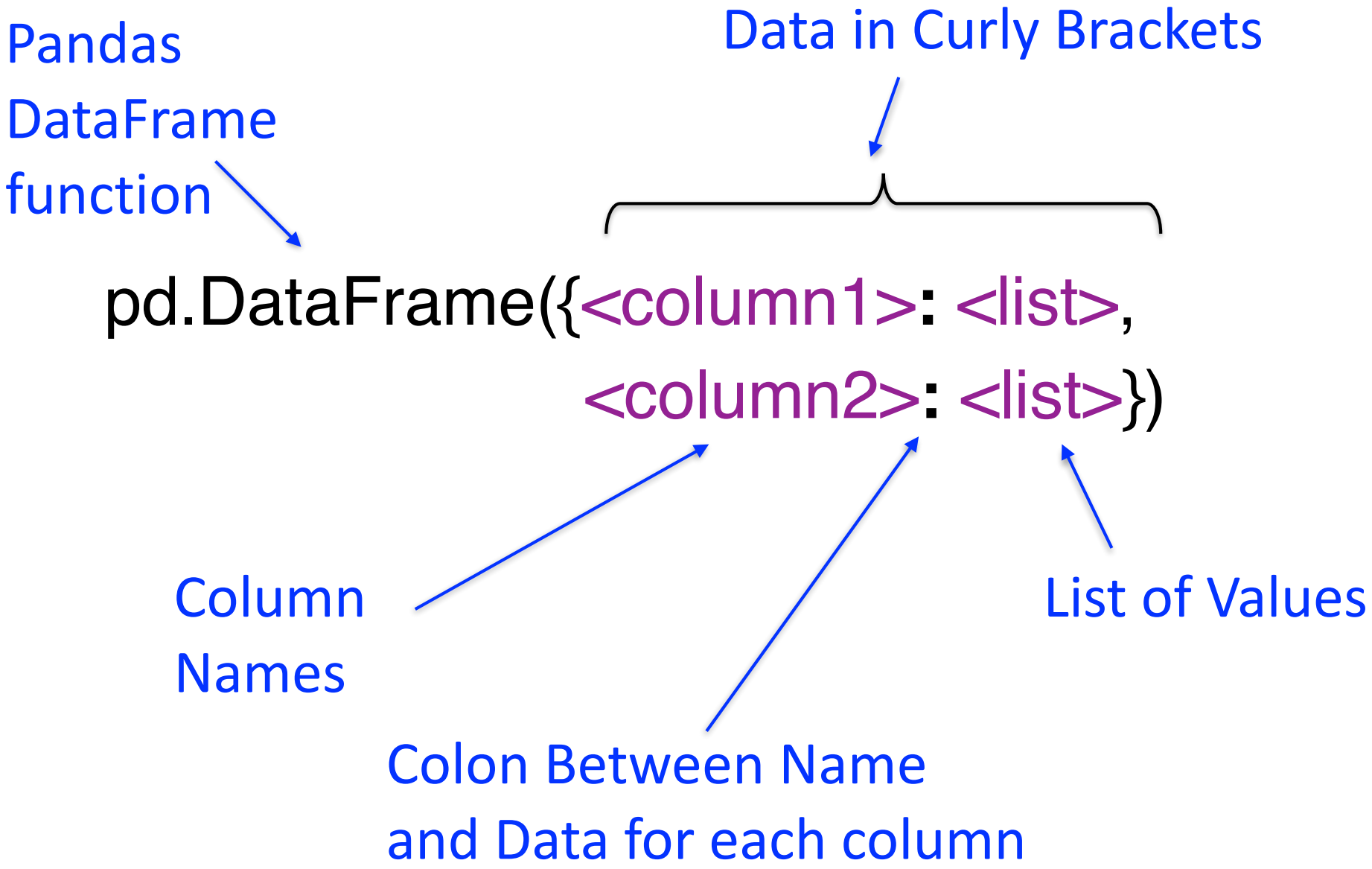
There are a couple ways to index DataFrames

- Column names and row index
- Relative position (similar to Excel)

Indexing can be a large source of confusion and frustration

However, we'll go over some examples that will help avoid a lot of that pain

Create DataFrame Syntax



Create a DataFrame

Build simple DataFrame

```
import pandas as pd
df = pd.DataFrame({'name':['Bob', 'Jen', 'Tim'],
                   'age':[20, 30, 40],
                   'pet':['cat', 'dog', 'bird']})
print df
```

View the column names and index values

```
print df.columns
print df.index
```

Indexing DataFrames

Select a column by name in 2 different ways

```
print df['name']
print df.name
```

Select multiple columns

```
print df[['name','pet']]
```

Select a row by index

```
print df.ix[0]
```

Pandas Documentation

Let's look up the sort function

- Go to pandas.pydata.org
- Select the documentation for the version you have (0.18.1)
- Use search to find 'sort values'

Pandas - Sort Values

pandas.DataFrame.sort_values

`DataFrame.sort_values(by, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')`

Sort by the values along either axis

New in version 0.17.0.

Parameters:

by *: string name or list of names which refer to the axis items*

axis *: index, columns to direct sorting*

ascending *: bool or list of bool*

Sort ascending vs. descending. Specify list for multiple sort orders. If this is a list of bools, must match the length of the by.

inplace *: bool*

if True, perform operation in-place

kind *: {quicksort, mergesort, heapsort}*

Choice of sorting algorithm. See also ndarray.np.sort for more information. *mergesort* is the only stable algorithm. For DataFrames, this option is only applied when sorting on a single column or label.

na_position *: {'first', 'last'}*

first puts NaNs at the beginning, *last* puts NaNs at the end

Returns:

sorted_obj *: DataFrame*

Row Index Stays with Data

Sort the data by name

```
df.sort_values('pet',inplace=True)
```

View the index after the sort

```
print df
```

The index is a fixed reference that is assigned when you create a DataFrame

Indexing by Relative Position

Panda's has another index method - **.iloc**

- Syntax:

  <DataFrame>.iloc[<row>,<column>]

- Row and column are the relative position of the data cells you want
- To select multiple rows or columns, use a colon to separate the start and end values
- Colon with no value returns all rows or columns

Don’t Confuse ix and iloc

Difference between ix and iloc

```
print df.ix[0]
print df.iloc[0]
```

Use iloc to select all rows of a column

```
print df.iloc[:,2]
```

Use iloc to select the last row

```
print df.iloc[-1,:]
```

Exercises

- Create a DataFrame
- What is the name of the first column (by relative position)?
- Sort the DataFrame by city in ascending order.
- Which customer is in the last row (by relative position)?

Bonus:

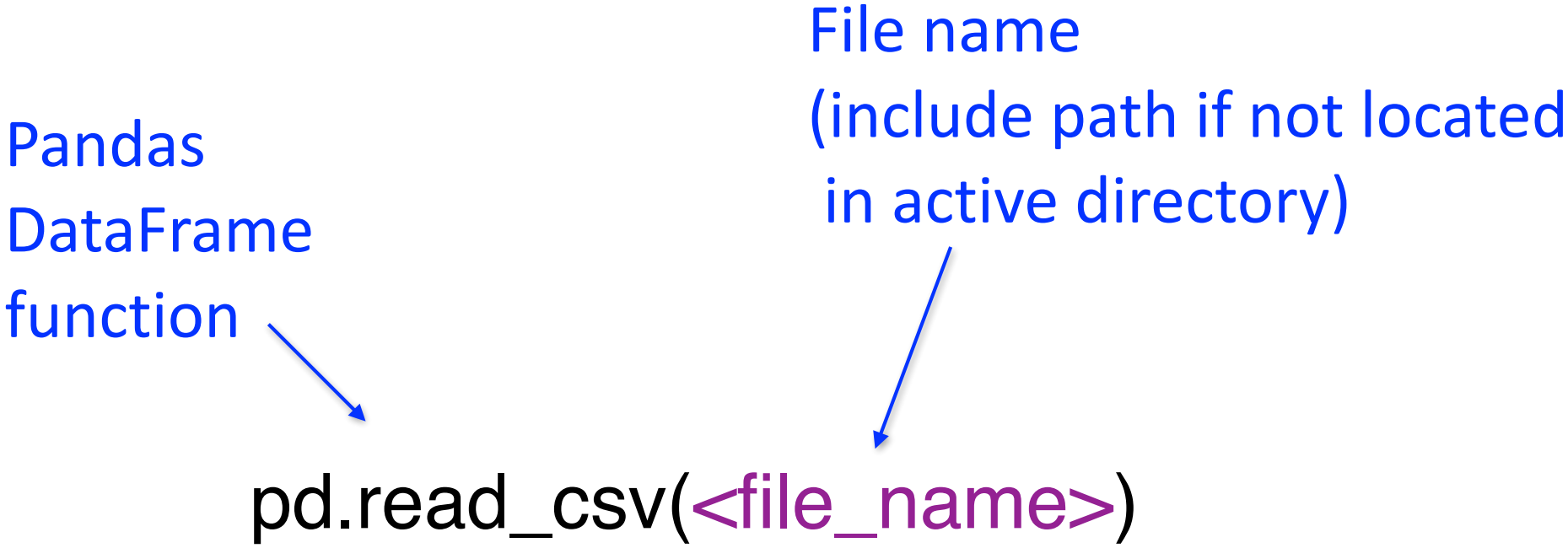
- Change the order of the columns so that customer is the first column
- Rename a column

Importing and Exporting Data

- Pandas has easy to use functions for importing and exporting different data types:
  - CSV Files
  - Excel Worksheets
  - Queries from Databases

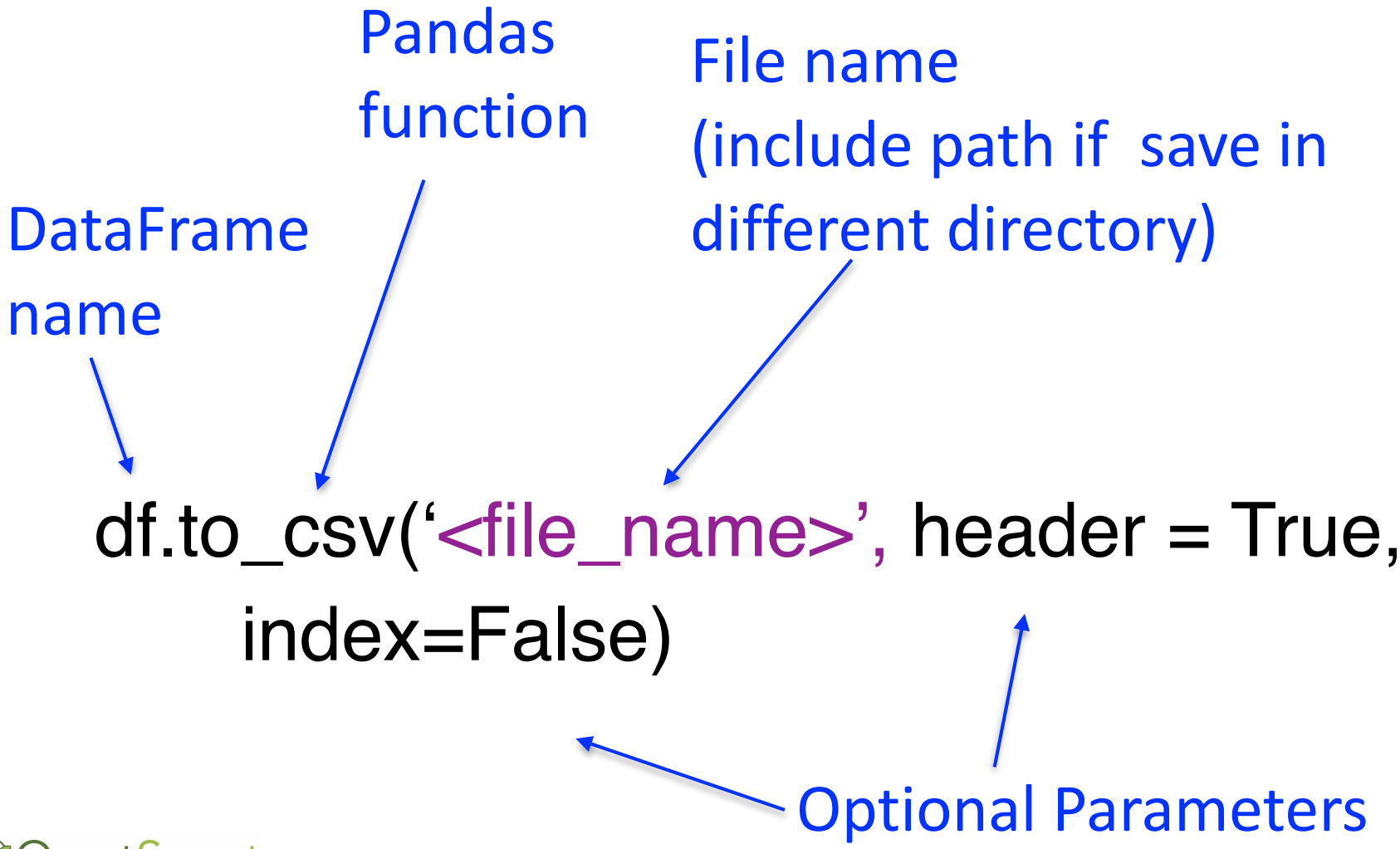
Reading CSV Files

- Syntax:



Writing CSV Files

- Syntax:



Read and Write CSV Files

Open the test_pandas.csv file as a dataframe

```
data = pd.read_csv('test_pandas.csv')
print data
```

Write the data to a csv file without the headers

```
data.to_csv('test_pandas_no_header.csv',
        header=False, index=True)
```

Read and Write CSV Files

Open the test_pandas_no_header.csv file

```
data_no_header =
   pd.read_csv( 'test_pandas_no_header.csv')

print data_no_header
```

Read and Write CSV Files

Try opening the file with header set to None

```
data_no_header =
   pd.read_csv( 'test_pandas_no_header.csv',
                header=None)

print data_no_header
```

Read Excel Files

Reading Excel files is similar. Just add worksheet you want to read.

```
data = pd.read_excel('test_pandas.xlsm', 'Sheet1')

print data
```

Write Excel Files

For writing to an Excel file, you need to use a writer object. This enables you to save multiple dataframes as separate sheets in the same file.

```
writer = pd.ExcelWriter('test_sheets.xlsx')
data.to_excel(writer,'Original')
data.to_excel(writer,'Copy')
writer.save()
```

Connecting to SQL Database

To read and write to SQL you need to establish a connection to a database. Creating connections are different for each type of database, but here is an example for sqlite.

```
import sqlite3
conn = sqlite3.connect('test_pandas.db')
```

Create DataFrame from Query

Use the connection for importing a query as a DataFrame

```
sql_query = "SELECT * FROM test"
data = pd.read_sql(sql_query,conn)
print data
```

Create New DataFrame

Create new DataFrame to load into SQL Database

```
new_data = pd.DataFrame({
        'id':[6,7],
        'city':['Dallas', 'Philadelphia'],
        'mascot':['Cowboys','Eagles']
        })
new_data = new_data[['id','city','mascot']]
print new_data
```

pandas.DataFrame.to_sql

`DataFrame.to_sql`(*name*, *con*, *flavor='sqlite'*, *schema=None*, *if_exists='fail'*, *index=True*, *index_label=None*, *chunksize=None*, *dtype=None*)

Write records stored in a DataFrame to a SQL database.

Parameters :

name *: string*

Name of SQL table

con *: SQLAlchemy engine or DBAPI2 connection (legacy mode)*

Using SQLAlchemy makes it possible to use any DB supported by that library. If a DBAPI2 object, only sqlite3 is supported.

flavor *: {'sqlite', 'mysql'}, default 'sqlite'*

The flavor of SQL to use. Ignored when using SQLAlchemy engine. 'mysql' is deprecated and will be removed in future versions, but it will be further supported through SQLAlchemy engines.

schema *: string, default None*

Specify the schema (if database flavor supports this). If None, use default schema.

if_exists *: {'fail', 'replace', 'append'}, default 'fail'*

- fail: If table exists, do nothing.
- replace: If table exists, drop it, recreate it, and insert data.
- append: If table exists, insert data. Create if does not exist.

index *: boolean, default True*

Write DataFrame index as a column.

Load DataFrame into SQL

Create new table in SQL

```
new_data.to_sql('new_table',conn)
```

Append to existing table in SQL

```
new_data.to_sql('test',conn,if_exists='append',
    index=False)
```

Don't forget to close connection

```
conn.close()
```

Exploring Titanic Data

- Import data from CSV Files
- Investigate data
  - View samples of the data
  - Evaluate summary statistics
- Filter and slice the data for analysis

Load and Investigate the File

Load the train.csv file as DataFrame

```
data = pd.read_csv('train.csv')
```

View the first 5 rows of the data set

```
data.head()
```

Analyze the Data

View the last 10 rows of the data set

```
data.tail(10)
```

Investigate the data types in the DataFrame

```
data.info()
```

Get some summary statistics

```
data.describe()
```

Filter the Data

Filter the data for men

```
data[data.Sex=='male']
```

Filter just the ages of the men

```
data.Age[data.Sex==‘male']
```

Count the men and women

```
print data.Sex[data.Sex=='male'].count()
print data.Sex[data.Sex=='female'].count()
```

Filter with Multiple Criteria

Calc the survival rate for adult men (age>=18)

```
data.Survived[(data.Sex==‘male')&
                    (data.Age>=18)].mean()
```

What was the survival rate for women and children?

```
data.Survived[(data.Sex==‘female’)|
                    (data.Age<18)].mean()
```

Group By

Compare the survival rates by sex

```
data.groupby(‘Sex’)[‘Survived’].mean()
```

Create a DataFrame with groupby and view the index

```
new = data.groupby(‘Sex')['Survived','Age'].mean()
print new.index
```

Reset Index

Reset index for new DataFrame

```
new.reset_index(inplace=True)
```

View the index and the DataFrame

```
print new.index
print new
```

Exercises

- What was the average age of the survivors?
- What was the survival rate of the children (age less than 18) and seniors (age greater than 60)?
- Group by Pclass and investigate average survival rate, age and fare
- Create an Excel file with the names and ages of the survivors on one tab and the names and ages of the deceased in another tab
- Investigate the data and discover an interesting insight

APIs

- Application Programming Interface
- Use URL requests to communicate information
- JSON (Javascript Object Notation) is the common format for results
- URL requests contain
  - API specific URL
  - key for security/identifying users
  - parameters for the request

Twitter API

- Python package for making API requests to Twitter
- Requires authentication with requests
- Collect data from Twitter programmatically

Package Management

- Although Anaconda provides a majority of the analytics packages, you'll eventually need to install additional packages
- Common package management tasks
  - Update anaconda
  - Check which packages are already installed
  - Install new package
  - Upgrade new package

Updating Anaconda

- Go to your command line / terminal
- Update conda first:

  conda update conda
- Next update anaconda:

  conda update anaconda

Warning: Do not upgrade a package within Anaconda with anything other than Conda

PIP

- PIP is the software designed for installing python packages
- View all installed packages with version numbers:

  pip freeze
- Install a package:

  pip install [package name]
- Upgrade a package

  pip install - - upgrade [package name]

Let's Install Some Packages

- Go to your command line / terminal
- Type:

  pip install twitter

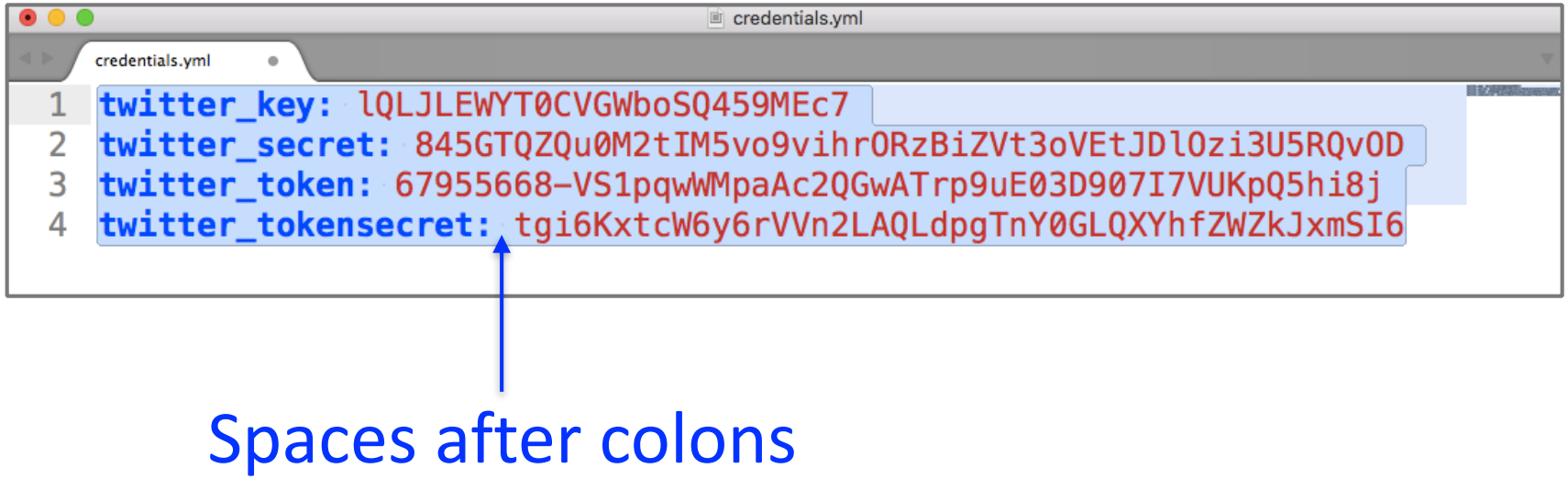
  pip install seaborn
- Check installation using pip freeze

YAML Files for Credentials

- YAML - Yet Another Markup Language
- Syntax is similar to dictionaries
- Useful for Saving Passwords and Keys because you can write code without revealing sensitive information

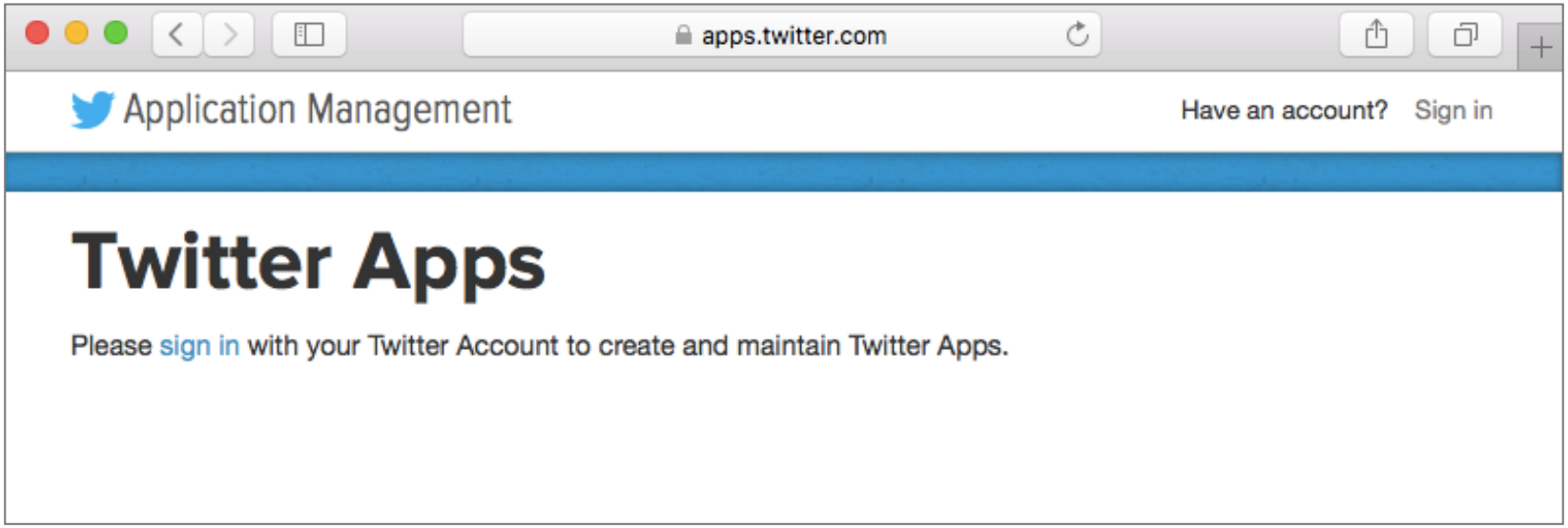
Create Credentials File

- Go to Sublime Text and create a file and save as credentials.yml in same folder as your ipython notebooks
- You'll save your credentials in this file:

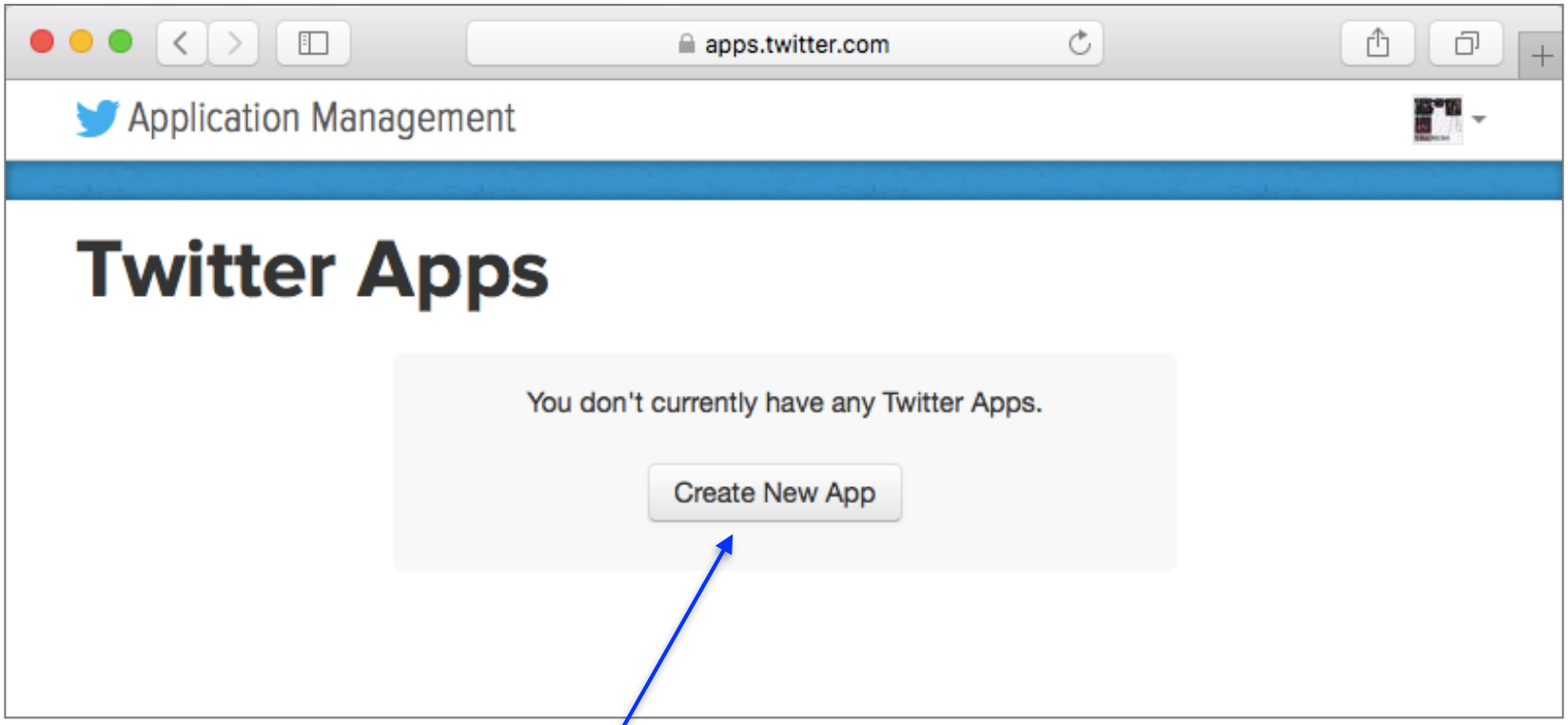


Set Up Twitter Credentials

Open your browser and navigate to apps.twitter.com and login to your Twitter account



Set Up Twitter Credentials

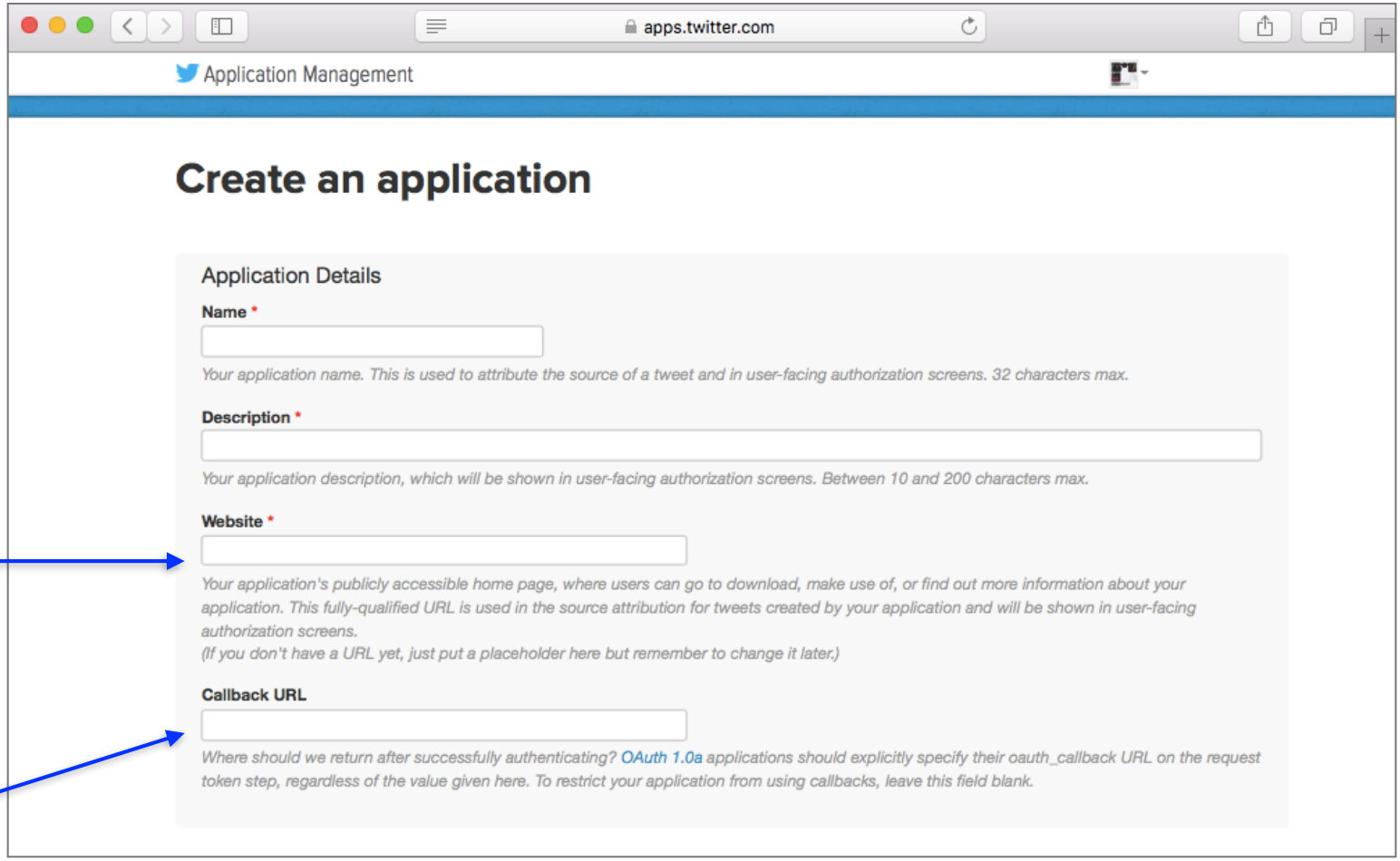


Create an App

Set Up Twitter Credentials

Use any URL with prefix of http://

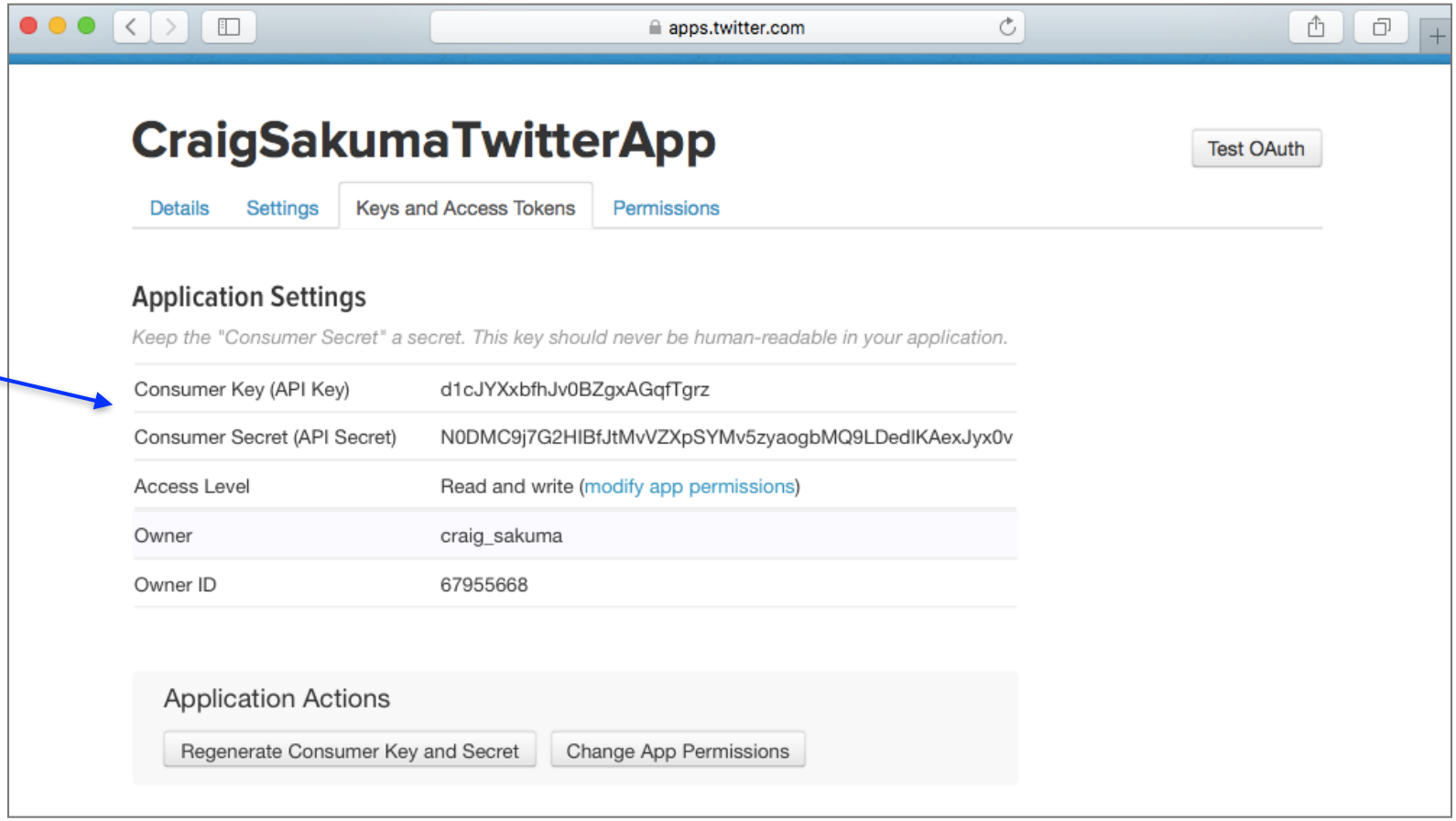
Leave blank



Set Up Twitter Credentials

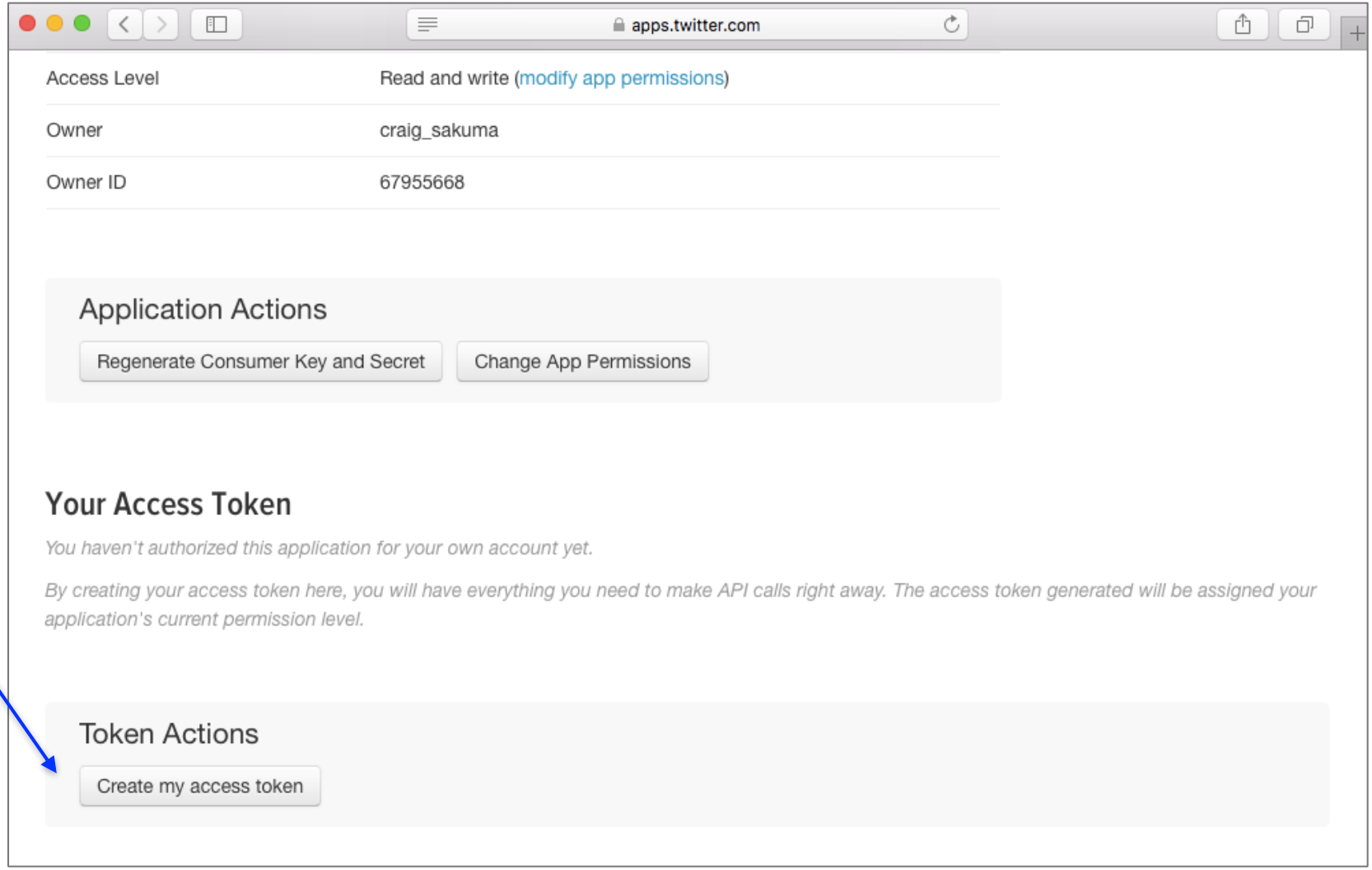
View your Keys and Access Tokens

Get your Key and Secret Data



Set Up Twitter Credentials

Create your access tokens



Use YAML for Credentials

Load your credentials file

```
import yaml
```

```
credentials = yaml.load(open('credentials.yml'))
```

Use YAML for Credentials

Load individual credentials

```
twitter_key = credentials['twitter_key']
twitter_secret = credentials['twitter_secret']
token = credentials['twitter_token']
token_secret = credentials['twitter_tokensecret']
```

Authenticate with Twitter

```
import twitter
```

```
auth = twitter.oauth.OAuth(token, token_secret,
                           twitter_key, twitter_secret)

twitter_api = twitter.Twitter(auth=auth)
```

Request Data from Twitter

```
search_string = '#goldenstatewarriors'
count = 10
search_results = twitter_api.search.tweets(
                        q=search_string, count=count)
```

```
print search_results
```