Importing non-flat files from the web:

```
# Import package
import pandas as pd

# Assign url of file: url
url = 'http://s3.amazonaws.com/assets.datacamp.com/course/importing_data_into_r/
latitude.xls'

# Read in all sheets of Excel file: xl
xl = pd.read_excel(url, sheetname=None)

# Print the sheetnames to the shell
print(xl.keys())

# Print the head of the first sheet (using its name, NOT its index)
print(xl[‘1700’].head())
```

Turning a webpage into data using BeautifulSoup: getting the hyperlinks

```
# Import packages
import requests
from bs4 import BeautifulSoup

# Specify url
url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r
r = requests.get(url)

# Extracts the response as html: html_doc
html_doc = r.text

# create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc)

# Print the title of Guido's webpage
print(soup.title)

# Find all 'a' tags (which define hyperlinks): a_tags
a_tags = soup.find_all('a')

# Print the URLs to the shell
for link in a_tags:
    print(link.get(‘href'))
```

Loading and exploring a JSON

```
# Load JSON: json_data

with open("a_movie.json") as json_file:

    json_data = json.load(json_file)


# Print each key-value pair in json_data

for k in json_data.keys():

    print(k + ': ', json_data[k])
```

API requests

```
# Import requests package

import requests


# Assign URL to variable: url

url = 'http://www.omdbapi.com/?apikey=ff21610b&t=the+social+network'


# Package the request, send the request and catch the response: r

r = requests.get(url)


# Print the text of the response

print(r.text)
```

JSON-from the web to Python

```
# Import package

import requests


# Assign URL to variable: url

url = 'http://www.omdbapi.com/?apikey=ff21610b&t=social+network'


# Package the request, send the request and catch the response: r

r = requests.get(url)


# Decode the JSON data into a dictionary: json_data

json_data = r.json()


# Print each key-value pair in json_data

for k in json_data.keys():

    print(k + ': ', json_data[k])
```

Checking out the Wikipedia API

```
# Import package

import requests


# Assign URL to variable: url

url = 'https://en.wikipedia.org/w/api.php?
action=query&prop=extracts&format=json&exintro=&titles=pizza'
```

```
# Package the request, send the request and catch the response: r

r = requests.get(url)


# Decode the JSON data into a dictionary: json_data

json_data = r.json()


# Print the Wikipedia page extract

pizza_extract = json_data['query']['pages']['24768']['extract']

print(pizza_extract)
```

The Twitter API and Authentication

API Authentication

```
# Import package
import tweepy

# Store OAuth authentication credentials in relevant variables
access_token = "1092294848-aHN7DcRP9B4VMTQIhwqOYiB14YkW92fFO8k8EPy"
access_token_secret = "X4dHmhPfaksHcQ7SCbmZa2oYBBVSD2g8uIHXsp5CTaksx"
consumer_key = "nZ6EA0FxZ293SxGNg8g8aP0HM"
consumer_secret = "fJGEodwe3KiKUnsYJC3VRndj7jevVvXbK2D5EiJ2nehafRgA6i"

# Pass OAuth details to tweepy's OAuth handler
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
```

Streaming Tweets

```
# Initialize Stream listener
l = MyStreamListener()

# Create your Stream object with authentication
stream = tweepy.Stream(auth, l)


# Filter Twitter Streams to capture data by the keywords:
stream.filter(track=['clinton', 'trump', 'sanders', 'cruz'])
```

Load and explore your Twitter data

```
# Import package
import json

# String of path to file: tweets_data_path
tweets_data_path = 'tweets.txt'

# Initialize empty list to store tweets: tweets_data
tweets_data = []

# Open connection to file
tweets_file = open(tweets_data_path, "r")

# Read in tweets and store in list: tweets_data
for line in tweets_file:
    tweet = json.loads(line)
    tweets_data.append(tweet)

# Close connection to file
tweets_file.close()

# Print the keys of the first tweet dict
print(tweets_data[0].keys())
```

Twitter data to DataFrame

```
# Import package
import pandas as pd

# Build DataFrame of tweet texts and languages
df = pd.DataFrame(tweets_data, columns=['text', 'lang'])

# Print head of DataFrame
print(df.head())
```

A little bit of Twitter text analysis

```
# Initialize list to store tweet counts
[clinton, trump, sanders, cruz] = [0, 0, 0, 0]

# Iterate through df, counting the number of tweets in which
# each candidate is mentioned
for index, row in df.iterrows():
    clinton += word_in_text('clinton', row['text'])
    trump += word_in_text('trump', row['text'])
    sanders += word_in_text('sanders', row['text'])
    cruz += word_in_text('cruz', row[‘text'])
```

Plotting your Twitter data

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns

# Set seaborn style
sns.set(color_codes=True)

# Create a list of labels:cd
cd = ['clinton', 'trump', 'sanders', 'cruz']
```

```
# Plot histogram
ax = sns.barplot(cd, [clinton, trump, sanders, cruz])
ax.set(ylabel="count")
plt.show()
```

Basics of Relational Databases: Introduction to Databases

Engines and Connection Strings

```
# Import create_engine
from sqlalchemy import create_engine

# Create an engine that connects to the census.sqlite file: engine
engine = create_engine('sqlite:///census.sqlite')

# Print table names
print(engine.table_names())
```

Autoloading Tables from a Database

```
# Import Table
from sqlalchemy import Table

# Reflect census table from the engine: census
census = Table('census', metadata, autoload=True, autoload_with=engine)

# Print census table metadata
print(repr(census))
```

Viewing Table Details

```
# Reflect the census table from the engine: census
census = Table('census', metadata, autoload=True, autoload_with=engine)

# Print the column names
```

```
print(census.columns.keys())

# Print full table metadata
print(repr(metadata.tables['census']))
```

Selecting data from a Table: raw SQL

```
# Build select statement for census table: stmt
stmt = 'SELECT * FROM census'

# Execute the statement and fetch the results: results
results = connection.execute(stmt).fetchall()

# Print results
print(results)
```

Selecting data from a Table with SQLAlchemy

```
# Import select
from sqlalchemy import select

# Reflect census table via engine: census
census = Table('census', metadata, autoload=True, autoload_with=engine)

# Build select statement for census table: stmt
stmt = select([census])

# Print the emitted statement to see the SQL emitted
print(stmt)

# Execute the statement and print the results
print(connection.execute(stmt).fetchall())
```

Handling a ResultSet

```
# Get the first row of the results by using an index: first_row
first_row = results[0]

# Print the first row of the results
print(first_row)

# Print the first column of the row by using an index
print(first_row[0])

# Print the state column of the row by using its name
print(first_row['state'])
```

Applying Filtering, Ordering and Grouping to Queries:

Connecting to a PostgreSQL Database

```
# Import create_engine function
from sqlalchemy import create_engine

# Create an engine to the census database
engine = create_engine('postgresql+psycopg2://student:datacamp@postgresql.csrrinzqubik.us-east-1.rds.amazonaws.com:5432/census')

# Use the .table_names() method on the engine to print the table names
print(engine.table_names())
```

Filter data selected from a Table - Simple

```
# Create a select query: stmt
stmt = select([census])

# Add a where clause to filter the results to only those for New York
stmt = stmt.where(census.columns.state == 'New York')
```

```
# Execute the query to retrieve all the data returned: results
results = connection.execute(stmt).fetchall()

# Loop over the results and print the age, sex, and pop2008
for result in results:
    print(result.age, result.sex, result.pop2008)
```

Filter data selected from a Table - Expressions

```
# Create a query for the census table: stmt
stmt = select([census])

# Append a where clause to match all the states in_ the list states
stmt = stmt.where(census.columns.state.in_(states))

# Loop over the ResultProxy and print the state and its population in 2000
for result in connection.execute(stmt):
    print(result.state, result.pop2000)
```

Filter data selected from a Table - Advanced

```
# Import and_
from sqlalchemy import and_

# Build a query for the census table: stmt
stmt = select([census])

# Append a where clause to select only non-male records from California using and_
stmt = stmt.where(
    # The state of California with a non-male sex
    and_(census.columns.state == 'California',
         census.columns.sex != 'M'
         )
)
```

```
# Loop over the ResultProxy printing the age and sex
for result in connection.execute(stmt):
    print(result.age, result.sex)
```

Ordering by a Single Column

```
# Build a query to select the state column: stmt
stmt = select([census.columns.state])

# Order stmt by the state column
stmt = stmt.order_by(census.columns.state)

# Execute the query and store the results: results
results = connection.execute(stmt).fetchall()

# Print the first 10 results
print(results[:10])
```

Ordering in Descending Order by a Single Column

```
# Import desc
from sqlalchemy import desc

# Build a query to select the state column: stmt
stmt = select([census.columns.state])

# Order stmt by state in descending order: rev_stmt
rev_stmt = stmt.order_by(desc(census.columns.state))

# Execute the query and store the results: rev_results
rev_results = connection.execute(rev_stmt).fetchall()

# Print the first 10 rev_results
print(rev_results[:10])
```

Ordering by Multiple Columns

```
# Build a query to select state and age: stmt
stmt = select([census.columns.state, census.columns.age])

# Append order by to ascend by state and descend by age
stmt = stmt.order_by(census.columns.state, desc(census.columns.age))

# Execute the statement and store all the records: results
results = connection.execute(stmt).fetchall()

# Print the first 20 results
print(results[:20])
```