Intermediate

Python For Data Science

December, 2017

GSA

D2D

DATA TO DECISIONS

Goal For Today:

- Install python packages
- Load Data from 2 Different Sources
- Merge, and slice the data
- Perform some calculations
- Some analysis
- A Simple Plot

GSA

D2D
DATA TO DECISIONS

1.Packages & Modules

1.Packages & Modules

- Modules and Packages provide a way of code reuse
- Python comes with a library of standard modules/packages
  - Such as datetime
  - …or the statistics module
    - Import statistics
    - print(statistics.mean([1,2,3,4,5,6]))
- A package is a collection of modules
- You can import an entire package, or a module within the package
  - Import matplotlib
  - Import matplotlib.pyplot
- Additional packages can be installed using **pip**
  - ***To install a new package:*** pip install < package_name >
  - ***To uninstall a package:*** pip uninstall < package_name >
  - ***To list all installed packages:*** pip list
  - ***To see information about a package:*** pip show <package_name>
- **Anaconda has another package management system**: conda

1.Packages & Modules (Continued)

- Today we will be using:

1. **pymysql:**
   - ➢ A MySQL client library written in pure python.
2. **pandas**:
   - ➢A library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language
   - ➢The de facto python library when working with heterogeneous tabular data
3. **matplotlib:**
   - ➢One of the most widely used libraries for plotting in Python
   - ➢The first Python data visualization library
   - ➢While good for getting a “feel” of the data, not suitable for publication charts
   - ➢Very powerful, and can get very complex!

Note: There are alternatives to these packages (ggplot, pyodbc…etc)

Data Sets

- We will be using public data sets from **Data.gov**

- **Public Building Services data sets containing PBS building inventory that consists of both owned and leased buildings with active and excess status.**

- PBS REXUS Buildings:

https://catalog.data.gov/dataset/real-estate-across-the-united-states-rexus-inventory-building

- PBS REXUS Lease:

https://catalog.data.gov/dataset/real-estate-across-the-united-states-rexus-lease

2.Reading Data Into Python

2.Reading Data Into Python

To read from a MySQL DB, we will use the **pymysql** package

End Goal: Read the data from a table into python

- **Pseudo Code:**

```
➢Define Connection Parameters
➢Establish a Connection
➢Execute A Query with a cursor
    ▪ A control structure that allows the traversal over the records in a database table
➢Store and display the results
```

2.Reading Data Into Python (Continued)

Code:

```
import pymysql
# open connection to the database
conn = pymysql.connect(host='XXXXXXXXXXXXX',
                       port=3306,
                       user=‘my_user',
                       passwd='YYYYYYYY',
                       db=‘MY_DB',
                       charset='utf8')
#Defining Cursor on the connecion
cur = conn.cursor()
cur.execute("SELECT * FROM My_Table" )
data = cur.fetchall()
#Loop over the result set and print record
for i in data:
    print(i)
# close connection to the database
cur.close()
conn.close()
```

2.Reading Data Into Python (Continued)

To read from a CSV file, we will use the **csv** package

End Goal: Read the data from a csv file into python

- **Pseudo Code:**

```
➢Open The file
➢Read Contents into Object
➢Store and display the results
```

2.Reading Data Into Python (Continued)

Code:

```
import csv

with open(‘file.csv') as my_csv:
    #csv_reader = csv.DictReader(my_csv) #To Read into a DictObject
    csv_reader = csv.reader(my_csv)
    for row in csv_reader:
        print(row)
        #print(row[1]) #To Access First Column Values
        #print(row['RegionCode']) #To Access Dictionary Values with Key RegionCode If DictObject
```

2.Reading Data Into Python (Continued)

- Data Might not be in the same structure
- Manipulation will not be the same for all sources
- Why not use a common object?

3.pandas

3.pandas

- **Pandas DataFrame**

  - The primary pandas data structure
  - Two-dimensional size, mutable, tabular data structure with labeled axes (rows and columns)
  - Provide a common structure for all data sources

GSA

D2D
DATA TO DECISIONS

3.Pandas (Continued)

Now Lets read Both our sources again, this time in dataFrames:

End Goal: Read the data from both DB and csv file into python

- **Pseudo Code:**

  - ➢Define Connection Parameters
  - ➢Establish a Connection
  - ➢Execute A Query with a cursor
  - ➢**Store into dataFrame**

  - ➢Open The file
  - ➢Read Contents into Object
  - ➢**Store in dataFrame**

3.Pandas (Continued)

Code:

```
import pymysql
import pandas

# open connection to the database
conn = pymysql.connect(host=‘XXXX',port=3306,user=‘my_user',passwd=‘XXXX',db=‘my_db',charset='utf8')

df_db = pandas.read_sql('SELECT * FROM my_table',conn)
print(df_db.head(3))

# close connection to the database
conn.close()

# CSV reading from Pandas into df
df_csv = pandas.read_csv(“my_file.csv")
print(df_csv.head())
```

3.Pandas (Continued)

Joining DataFrames:

End Goal: Have one Data Frame based on a Join

- SQL Joins are not possible when both sources are not in the same database!

- **Pseudo Code:**

  - Define Connection Parameters
  - Establish a Connection
  - Execute A Query with a cursor
  - Store into dataFrame
  - Open The file
  - Read Contents into Object
  - Store in dataFrame
  - **Create new dataFrame by joining both dataFrames**

3.Pandas (Continued)

Code:

```
import pymysql
import pandas

# open connection to the database
conn = pymysql.connect(host=‘XXXX',port=3306,user=‘my_user',passwd=‘XXXX',db=‘my_db',charset='utf8')

df_db = pandas.read_sql('SELECT * FROM my_table',conn)
# close connection to the database
conn.close()

# CSV reading from Pandas into df
df_csv = pandas.read_csv(“my_file.csv")

full_df = pandas.merge(df_db, df_csv, on=‘JoinColumnName', how=‘inner') #or outer, Left or right
print(full_df.head())
```

3.Pandas (Continued)

- Slicing A Data Frame
  - ➢Use the .loc function (There are other ways!)
  - ➢Accepts the same slice notation that Python lists do for both row and columns.
    - Notation: start:stop:step

Examples:

Our Columns: ColA ColB ColC ColD ColE

`df.loc[:, ‘ColA':’ColE']`→ All rows, ColA,ColB,ColC,ColD,ColE

`df.loc[:, :’ColE‘]`→ All rows, ColA,ColB,ColC,ColD,ColE

`df.loc[:, ‘ColA':’ColE‘:2]`→All rows, ColA, ColC, ColE

`df.loc[:, ‘ColA'::3]`→All rows, ColA, ColD

`df.loc[0:49, ‘ColD’:’ColB’:-1]`→ First 50 rows, ColD, ColC, ColB

`df.loc[0:100, [‘ColA,’ColE’]]`→ First 101 rows, ColA, ColE

3.Pandas (Continued)

- .loc can be used to specify data filters

Examples:

```
df.loc[df[‘ColA’]==‘value’]
df.loc[df[‘ColA’]!=‘value’]
df.loc[df[‘ColA’].isin([‘value1’,’value2’])]
df.loc[~df[‘ColA’].isin([‘value1’,’value2’])]
df.loc[(df[‘ColA’]==‘value’) & (df[‘ColB’]==‘anotherValue’)]
```

- You can sort DataFrames using sort_values function

Example:

```
df.sort_values('ColA', ascending=1)
df.sort_values(['ColA', ‘ColB'], ascending=[1, 0])
```

3.Pandas (Continued)

- **Adding a Column to a DataFrame Based on other columns:**

Example:

```
df[‘ColF’] = df[‘ColA’]+df[‘ColB’]
```

- **Data can be grouped using groupby**

Example:

```
df.groupby(‘ColA’)[‘ColB’].mean()
df.groupby(‘ColA’)[[‘ColB’,‘ColC’]].describe()
```

- **DataFrame can be written out to a file using :**
  - ➢dataFrame.to_csv("test2.csv", sep='\t', encoding='utf-8',index=False)

3.Pandas (Continued)

Full Code:

```
import pymysql
import pandas
# open connection to the database
conn = pymysql.connect(host=‘XXXX',port=3306,user=‘my_user',passwd=‘XXXX',db=‘my_db',charset='utf8')
df_db = pandas.read_sql('SELECT * FROM my_table',conn)
# close connection to the database
conn.close()
# CSV reading from Pandas into df
df_csv = pandas.read_csv(“my_file.csv")
full_df = pandas.merge(df_db, df_csv, on=‘JoinColumnName', how=‘inner') #or outer, Left or right
print(full_df.head())
sliced_df = new_df.loc[:, [‘ColA',‘ColB',‘ColC']]
sliced_df = sliced_df.loc[(sliced_df[‘ColA'].isin(['1','2','3'])) & (sliced_df[‘ColB']>200)
].sort_values([‘ColB',‘ColC'], ascending=[1,0])
sliced_df[‘ColD']=sliced_df[‘ColB']/sliced_df[‘ColC']
print(sliced_df.groupby(‘ColA')[‘ColD'].mean())
print(sliced_df.groupby(‘ColA')[‘ColD'].std())
print(sliced_df.groupby(‘ColA')[‘ColD'].describe())
```

4.Plotting with matplotlib

4.Plotting with matplotlib

- **Goal:** Plot the average price per square for each congressional district

Statistics Reminder:

- **Mean:**
  - A single value that describes the average of an entire set
  - To calculate the mean, add up the values in the data set and then divide by the number of values that you added
- **Standard Deviation:**
  - The amount of variation of a set of data values in relation to the mean
    - A low standard deviation indicates that the data points tend to be close to the mean
    - A high standard deviation indicates that the data points are spread out over a wider range of values
  - To Calculate Standard Deviation:

$$s = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}}$$

DATA TO DECISIONS

4.Plotting with matplotlib

Note: The plot method is just a simple wrapper around matplotlib.plot

Full Code:

```
import pymysql

import pandas

import matplotlib

conn = pymysql.connect(host=‘XXXX',port=3306,user=‘my_user',passwd=‘XXXX',db=‘my_db',charset='utf8')

df_db = pandas.read_sql('SELECT * FROM my_table',conn)

conn.close()

df_csv = pandas.read_csv(“my_file.csv")

full_df = pandas.merge(df_db, df_csv, on=‘JoinColumnName', how=‘inner') #or outer, Left or right

print(full_df.head())

sliced_df = new_df.loc[:, [‘ColA',‘ColB',‘ColC']]

sliced_df = sliced_df.loc[(sliced_df[‘ColA'].isin(['1','2','3'])) & (sliced_df[‘ColB']>200)
].sort_values([‘ColB',‘ColC'], ascending=[1,0])

sliced_df[‘ColD']=sliced_df[‘ColB']/sliced_df[‘ColC']

mean_df = sliced_df.groupby(‘ColA')[‘ColD'].mean()

std_df = sliced_df.groupby(‘ColA')[‘ColD'].std()

mean_plot = mean_df.plot(figsize=(12,6),fontsize=12,color='red',kind='bar',rot=0,yerr=std_df)

mean_plot.set_title(‘Chart Totle',fontsize=12)

mean_plot.set_xlabel(‘X Axis Label')

mean_plot.set_ylabel(‘Y Axis Label')

mean_plot.set_ylim(-15,15)
```

Questions?

Resources

- pip reference: https://pip.pypa.io/en/stable/reference/pip/
- pandas reference: https://pandas.pydata.org/
- matplotlib reference: https://matplotlib.org/
- pymysql reference: https://pymysql.readthedocs.io/en/latest/
- csv reference: https://docs.python.org/2/library/csv.html
- Pandas Merging: https://pandas.pydata.org/pandas-docs/stable/merging.html
- Pandas groupby: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.groupby.html
- Plotting with matplotlob: http://pandas.pydata.org/pandas-docs/version/0.13/visualization.html
- Matplotlib reference: https://matplotlib.org/contents.html

Thank You

GSA

D2D
DATA TO DECISIONS