Data Engineering

Preparing Data in Python

June, 2018

GSA

D2D
DATA TO DECISIONS

First Things First

- Question from last session: Connect to Oracle using **cx_Oracle**

https://oracle.github.io/python-cx_Oracle/

- Questions on reading data?

GSA

D2D
DATA TO DECISIONS

Goal for the next 2 sessions

- Import data into python dataframes
- Prepare/clean the data
  - Slicing
    - Selecting rows based on Value
    - Dropping columns from DF vs Selecting Columns (inplace operations)
  - Changing the index
  - Extracting Unique Values
  - Data type conversion
  - Map functions
  - Renaming column identifiers
  - Handling Na values
  - Joining
  - Regular Expressions for string operations
  - Write Data to
    - File
    - Database

GSA

D2D
DATA TO DECISIONS

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

GSA

D2D
DATA TO DECISIONS

A word about data preparation

- Clean data is:
  - ➢Easier to work with
  - ➢Provide a common dataset for all operations that rely on it
  - ➢Clean once, use many times
  - ➢Can be saved for the future

- Ideally all data will always be clean:
  - ➢Columns named correctly
  - ➢No missing Values
  - ➢No anomalies
  - ➢Correct data types
  - ➢Correct values

Unfortunately….

GSA

D2D
DATA TO DECISIONS

A word about data preparation

- Reasons Why data is not always “perfect”:

  - ➢Different systems
  - ➢Different uses
  - ➢Naming conventions
  - ➢No integrity checks at source
  - ➢……. Etc.

GSA

D2D
DATA TO DECISIONS

Lets Code

- Load the data from the table tbl_RexusBuilding into a df
- Discard All Rows that have a BldgANSIUsable less than 200
- Set Building ID as the index
- Discard all columns except
  - BuildingID
  - BldgZip
  - CongressionalDistrict
  - BldgANSIUsable

GSA

D2D
DATA TO DECISIONS

CODE – Using loc

```
import pymysql
import pandas
# open connection to the database
conn = pymysql.connect(host='cfffv', port=3306, user='d2fffer', passwd='dfffd',
                       db='GSfffining', charset='utf8')

df_db = pandas.read_sql('SELECT * FROM tbl_RexusBuilding',conn)
# close connection to the database
conn.close()
#Checking DF size (rows and columns)
print(df_db.shape)
#Chcking column types
print(df_db.dtypes)
#Selecting Rows >= 200
df_db = df_db.loc[(df_db['BldgANSIUsable']>= 200)]
#Selecting columns
mod_df_db = df_db.loc[:,['LocationID', 'BldgZip', 'CongressionalDistrict', 'BldgANSIUsable']]
#Checking if LocationID is unique
print(mod_df_db['LocationID'].is_unique)
#Set LocationID as the index
mod_df_db.set_index('LocationID', inplace=True)

print(mod_df_db.head())
```

GSA

D2D
DATA TO DECISIONS

CODE – Using Drop

```
import pymysql
import pandas
# open connection to the database
conn = pymysql.connect(host='cfffv', port=3306, user='d2fffer', passwd='dfffd',
                       db='GSfffining', charset='utf8')

df_db = pandas.read_sql('SELECT * FROM tbl_RexusBuilding',conn)
# close connection to the database
conn.close()
#Checking DF size (rows and columns)
print(df_db.shape)
#Chcking column types
print(df_db.dtypes)
#Selecting Rows >= 200
df_db = df_db.loc[(df_db['BldgANSIUsable']>= 200)]
#Selecting columns
to_drop = ['BldgAddress1','BldgAddress2','BldgCity','BldgCounty']
df_db.drop(to_drop, inplace=True, axis=1)
#Checking if LocationID is unique
print(df_db['LocationID'].is_unique)
#Set LocationID as the index
mdf_db.set_index('LocationID', inplace=True)

print(df_db.head())
```

GSA

D2D
DATA TO DECISIONS

Checkpoint

- Use df_db.shape to see number of rows and columns
- Use df_db.dtypes to check the column datatypes
- Use dataframe['identifier'].is_unique to check if the column has unique values
- Inplace operations will change the original dataframe
- Only drop columns using dataframe.drop(to_drop, inplace=True, axis=1) if you will no longer use the columns anywhere

GSA

D2D
DATA TO DECISIONS

CODE - Continued

```
# CSV reading from Pandas into df
df_csv = pandas.read_csv("datagovbldgrexus.csv",
                         dtype={'BldgZip':'str'})
print(df_csv.head(3))
#Chcking Information on data Set
print(df_csv.dtypes)
print(df_csv.get_dtype_counts())
print(df_csv.info())
print(df_csv.columns)
print(df_csv.isnull().head())
#renaming columns
df_csv.rename(columns={'LocationCode':'LocationID'}, inplace=True)
print(df_csv.head(3))
#Checking unique values for Congressional District
print(df_csv['CongressionalDistrict'].unique())
#Applying mapfunction to CongressionalDistrict
df_csv['CongressionalDistrict'] = df_csv.CongressionalDistrict.apply(cleanDistrict)
print(df_csv['CongressionalDistrict'].unique())
```

GSA

D2D
DATA TO DECISIONS

CODE - Continued

```
#Applying mapfunction to COngressionalDistrict
df_csv['CongressionalDistrict'] = df_csv.CongressionalDistrict.apply(cleanDistrict)
print(df_csv['CongressionalDistrict'].unique())

#Applying a map function to map values
print(df_csv['OwnedLeased'].unique())
df_csv['OwnedLeasedFlag'] = df_csv.OwnedLeased.map({'OWNED':1,'LEASED':0})
print(df_csv['OwnedLeasedFlag'].unique())
print(df_csv.head())

#I can give lambda functions a name and use the name in the apply functino:
#f = lambda x : str.lower(x)

#Demonstrating Lambda functions in Apply Operations
df_csv['BldgCounty'] = df_csv.BldgCounty.apply(lambda x : str.lower(x))
#OR df_csv['BldgCounty'] = df_csv['BldgCounty'].apply(lambda x : str.lower(x))
print(df_csv.head())
```

GSA

D2D
DATA TO DECISIONS

Checkpoint

- Use df_db.dtypes and df.get_dtype_counts() to see information on columns datatypes
- Pass the dtype parameter to a read_csv call to specify types
  - ➢(Can I use this with read_sql?)
- Use df.info() to see information about a DataFrame including
  - ➢index dtype,column dtypes,non-null value counts and memory usage
- Use df.columns to get a list of all columns
- Use df.isnull() to get a Boolean dataframe with values corresponding to Null and Non Null Values
- You can rename columns in a df using the df.rename function
- Use the unique() method on a df column to extract a list of unique values

GSA

D2D
DATA TO DECISIONS

Checkpoint

- You can apply function to all values in a column (series) using the apply() function
- You can map all values in a column (series) using the map() function
- applymap() applies a function to every single element in the entire dataframe.
  - ➢Be careful of types
  - ➢df.applymap(lambda x : x*1000)
- Lambda functions are useful for anonymous computations
  - ➢They can be given a name though:

    f = lambda x : x*1000

GSA

D2D
DATA TO DECISIONS

Handling Na Values

NA values:

- Such as None or numpy.NaN
- Isnull(): NA values gets mapped to True values.
- Everything else gets mapped to False values.
- Characters such as empty strings '' or numpy.inf are not considered NA values
  - ➢ (unless you set pandas.options.mode.use_inf_as_na = True)

- You can specify a list of NA values to be considered while the data is being read into a df

```
na_list = ['None','Empty','Nothing']
Df_csv = pandas.read_csv('file.csv', na_values=na_list)
```

GSA

D2D
DATA TO DECISIONS

Handling Na Values

Dealing with missing data

1. Add in a default value for the missing data
2. Get rid of (delete) the rows that have missing data
3. Get rid of (delete) the columns that have a high incidence of missing data

GSA

D2D
DATA TO DECISIONS

Handling Na Values

Dealing with missing data

1. Add in a default value for the missing data:

```
df.parkingSpots =  df.parkingSpots.fillna(0)
```

OR

```
df.parkingSpots =
df.parkingSpots.fillna(df.parkingSpots.mean())
```

GSA

D2D
DATA TO DECISIONS

Handling Na Values

Dealing with missing data

2. Get rid of (delete) the rows that have missing data:

Dropping all rows with any NA value:

➢ `df.dropna()`

Drop rows that have all NA values:

➢ `df.dropna(how='all')`

Set a Threshold on how many non-null values need to be in a row in order to keep it

➢ `df.dropna(thresh=5)`

Choose which columns you want to look for null values

➢ `df.dropna(subset=['title_year'])`

GSA

D2D
DATA TO DECISIONS

Handling Na Values

Dealing with missing data

3. Get rid of (delete) the columns that have a high incidence of missing data :

Drop the columns with that are all NA values:

➢ `df.dropna(axis=1, how='all')`

Drop all columns with any NA values:

➢ `df.dropna(axis=1, how='any')`

Documentation:

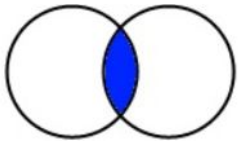
https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.isnull.html

GSA

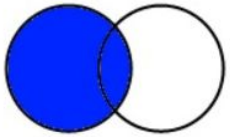
D2D
DATA TO DECISIONS

Merging

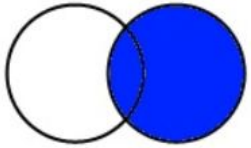
- The words “merge” and “join” are used relatively interchangeably
- “Merging” two datasets is bringing two datasets together into one
  - Aligning the rows from each based on common attributes or columns.
- Dataframes are merged using the pandas.merge() function
- Different types of Merging:
  - Inner



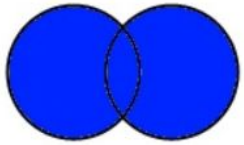
  - Left



  - Right



  - Outer



GSA

D2D
DATA TO DECISIONS

CODE - Merging

```
#Merging DFs

new_df = pandas.merge(df_db, df_csv, on='LocationID', how='inner')


#or


new_df = pandas.merge(df_db, df_csv, left_on='LocationID', right_on='LocationID', how='left')
```

GSA

D2D
DATA TO DECISIONS

Merging

- **Inner Merge / Inner join**:
  - ➢The default Pandas behaviour, only keep rows where the merge “on” value exists in both the left and right dataframes.
- **Left Merge / Left outer join:**
  - ➢Keep every row in the left dataframe. Where there are missing values of the “on” variable in the right dataframe, add empty / NaN values in the result.
- **Right Merge / Right outer join:**
  - ➢Keep every row in the right dataframe. Where there are missing values of the “on” variable in the left column, add empty / NaN values in the result.
- **Outer Merge / Full outer join:**
  - ➢A full outer join returns all the rows from the left dataframe, all the rows from the right dataframe, and matches up rows where possible, with NaNs elsewhere.

- The merge type to use is specified in the “how” parameter

GSA

D2D
DATA TO DECISIONS

Regular Expressions

- A sequence of characters
- Define a String search pattern
- Used to "find", or “validate”
- Very powerful!

GSA

D2D
DATA TO DECISIONS

Regular Expressions

- **\d:** Matches any decimal digit; this is equivalent to the class [0-9].
- **\D:** Matches any non-digit character; this is equivalent to the class [^0-9].
- **\s:** Matches any whitespace character; this is equivalent to the class [\t\n\r\f\v].
- **\S:** Matches any non-whitespace character; this is equivalent to the class [^ \t\n\r\f\v].
- **\w:** Matches any alphanumeric character; this is equivalent to the class [a-zA-Z0-9_].
- **\W:** Matches any non-alphanumeric character; this is equivalent to the class [^a-zA-Z0-9_].
- **+**: One or more
- ***:** Zero or More
- **?**: zero or one
- **{n}:** n times
- **{n,}:** n or more times
- **{n,m}**: between n and m times (inclusive)

GSA

- Full List: https://docs.python.org/3.4/howto/regex.html

D2D
DATA TO DECISIONS

CODE - Regex

```
#Splitting Address using REGEXP

df_csv['Matched'] = df_csv['BldgAddress1'].str.contains('^[\d]+\s[\D]+[ST|STREET]$', regex=True)
df_csv['Number'] = df_csv['BldgAddress1'].str.extract('(\d[\d]*)', expand=False)
df_csv['StreetName'] = df_csv['BldgAddress1'].str.extract('(\D[\D]*)', expand=False)
```

GSA

D2D
DATA TO DECISIONS

Writing Data out

- Writing Data from a pandas DF is just easier than reading it in!!

  - ➢To Write to a CSV File use df.to_csv(filename)
    - More Parameters available for fine grain control
    - https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_csv.html

  - ➢To Write to a DB table use df.to_sql(TableName, conn, f_exists=[fail|replace|append]
    - More Parameters available for fine grain control
    - Con is an sqlalchemy.engine.Engine or sqlite3.Connection
    - https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_sql.html

GSA

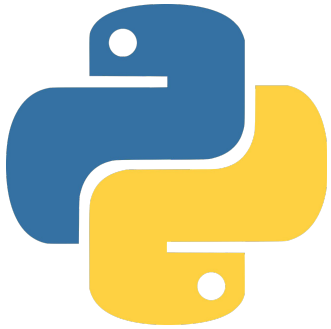
D2D
DATA TO DECISIONS

Questions?

GSA

D2D
DATA TO DECISIONS

Thank You



GSA

D2D
DATA TO DECISIONS