Importing & Cleaning Data In Python

▼ Introduction to Importing Data In Python

Importing of Data

```
Reading Data from a Text File:
filename = ‘textfile.txt’
file = open(filename, mode=’r’)
text = file.read()
file.close()

# to close a file and work on it at the same time aka (Context Manager)

with open(’textfile.txt’, ‘r’) as file:
     print(file.read())

file.readline() # To print a single line
```

▼ Importing flat files using NumPy

```
import numpy as np

filename = 'file.txt'

data = np.loadtxt(filename, delimiter=',', skiprows=1, usecols=[0, 2], dtype=str)

print(data)
```

```
# Import package
import numpy as np

# Assign filename to variable: file
file = 'digits.csv'

# Load file as array: digits
digits = np.loadtxt(file, delimiter=',')

# Print datatype of digits
print(type(digits))

# Select and reshape a row
im = digits[21, 1:]
im_sq = np.reshape(im, (28, 28))

# Plot reshaped data (matplotlib.pyplot already loaded as plt)
plt.imshow(im_sq, cmap='Greys', interpolation='nearest')
plt.show()

# To read from csv file using a nupy function

np.recfromcsv(filename)
```

Importing Flat files using Pandas

```
import pandas as pd
filename = 'file.csv'
data = pd.read_csv(filename)
data.head()

// for a specific number of rows
```

```
pd.read_csv(file, nrows=5, header=None)

//Another Variant

data = pd.read_csv(file, sep='\t', comment='#', na_values=['Noting'])
```

Introduction to other file types

- Excel Spreadsheet

  ```
  import pandas as pd
  file = 'urbanpop.xlsx'
  data = pd.ExcelFile(file)
  print(data.sheet_names)

  # to convert into data frame

  df1 = data.parse('1960-1966')   #sheet name, as a string
  df2 = data.parse(0)   # sheet index, as a float
  ```

- MATLAB files
  - Matrix Laboratory

  ```
  import scipy.io
  filename = 'workspace.mat'
  mat = scipy.io.loadmat(filename)
  ```

- SAS files
  - Statistical Analysis System
  - Business analytics and biostatistics

  ```
  import pandas as pd
  from sas7bdat import SAS7BDAT
  with SAS7BDAT('urbanpop.sas7bdat') as file:
    df_sas = file.to_data_frame()
  ```

- Stata files
  - Statistics + data
  - Academic social sciences research

  ```
  import pandas as pd
  data = pd.read_stata('urbanpop.dta')
  ```

- HDF5 files
  - Hierarchical Data Format Version 5
  - Store Large quantity of numerical data

  ```
  import h5py
  filename = 'filename.hdf5'
  data = h5py.File(filename, 'r')

  for key in data.keys():
    print(key)
  // meta
  // quality
  // strain

  for key in data['meta'].keys():
    print(key)
  // Some list of another set of keys

  // Converting into an array
  ```

```
print(np.array(data['meta']['Description']), np.array(data['meta']['Detector']))
```

- Pickled files (native to python: .pkl : ‘rb’)

```
import pickle
with open('pickled_fruit.pkl', 'rb') as file:
  data = pickle.load(file)
print(data)
```

▼ Introduction to relational database

Row: represents the entity type

Column: represent the attribute

Can also consist of secondary keys which are the primary keys of other tables

Creating a database engine in Python

- SQLite database
  - Fast and Simple
- SQLAlchemy
  - Works with many Relational Database Management Systems

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///Northwind.sqlite')

// To find the name of tables in the database
table_names = engine.table_names()
print(table_names)
```

Querying relational databases in Python

```
from sqlalchemy import create_engine
import pandas as pd
engine = create_engine('s1lite:///Northwind.sqlite')

with engine.connect() as con:
  rs = con.execute("SELECT OrderID, OrderDate, ShipName FROM Orders")
  df = pd.DataFrame(rs.fetchmany(size=5))
  df.columns = rs.keys()
```

Querying relational databases directly with pandas

```
// Single line code for querying

df = pd.read_sql_query("SELECT * FROM Orders", engine)

//Some SQL Queries

//To select all records from a table where a specific attribute is greater than 6 and has a specific order
SELECT * FROM Employee WHERE EmployeeId >= 6 ORDER BY BirthDate
```

Advanced querying: exploiting table relationships

Joining of tables in Python(Pandas)
Consider Two tables, Orders, and Customers

- The Orders Table has a secondary column “CustomerID”
- The Customers Table has a primary column “CustomerID”

Now, The Customers Table is a nester or structured part of Orders Table, in simple terms nested table.

```
//INNER JOIN in Python

from sqlalchemy import create_engine
import pandas as pd
engine = create_engine('sqlite:///Northwind.sqlite')
df = pd.read_sql_query("SELECT OrderID, CompanyName FROM Orders INNER JOIN Customers on Orders.CustomerID = Customers.CustomerID"

//Assign to rs the results from the following query: select all the records,
//extracting the Title of the record and Name of the artist of each record
//from the Album table and the Artist table, respectively. To do so,
//INNER JOIN these two tables on the ArtistID column of both.

SELECT Album.Title, Artist.Name FROM Album INNER JOIN Artist on Album.ArtistID = Artist.ArtistID
SELECT * FROM PlaylistTrack INNER JOIN Track on PlaylistTrack.TrackId = Track.TrackId WHERE Milliseconds < 25000Intermediate Impo
```

▼ Intermediate Importing Data In Python

Importing flat files from the web

- Import and locally save datasets from the web
- Load datasets into pandas’ DataFrames
- Make HTTP requests (GET requests)
  - URL = Uniform Resource Locator
  - Protocol identifier - HTTP
  - Resource name - datacamp.com
  - HTTP = HyperText Transfer Protocol
  - HTTPS = More Secure
  Going to a website = sending HTTP request
  GET request
  - urlretrieve( ) performs a GET request

```
from urllib.request import urlopen, Request
url = 'http://wikipedia.org'
request = Request(url)
response = urlopen(request)
HTML = response.read()
response.close()

// Package requests

import requests
url = 'http://wikipedia.org'
r = requst.get(url)
text = r.text
```

- Scrape web data such as HTML
- Parse HTML into useful data (BeautifulSoup)

```
from bs4 import BeautifulSoup
import requests

URL = 'someurl'
```

```
r = requests.get(url)
html_doc = r.text
soup = BeautifulSoup(html_doc)

//some methods

soup.title
soup.get_text()

//Extract URL and hyperlinks

for link in soup.find_all('a'):
  print(link.get('href'))
```

- Use the urllib and request packages
  - Provides interface for fetching data across the web
  - urlopen( ) - accepts URLs instead of files names

```
from urllib.request import urlretrieve
URL = 'some URL of a file'
urlretrieve(url, 'winequality-white.csv)
```

Introduction to APIs and JSONs

Application Programming Interface, used for protocols and routines, building and interacting with software applications

JSON

- JavaScript Object Notation
- Real-time server-to-browser communication
- Human readable

```
import json
with open('snakes.json', 'r') as json_file:
  json_data = json.load(json_file)

for key, values in json_data.items():
  print(key + ':', value)
```

APIs and interacting with the world wide web

```
import requests
URL = 'some ur'
r = requests.get(URL)
json_data = r.json()
for key, values in json_data.items():
  print(key + ':', value)

# Import requests package
import requests

# Assign URL to variable: url
url = 'http://www.omdbapi.com'

# Package the request, send the request and catch the response: r
r = requests.get(url, 'apikey=72bc447a&t=the+social+network')

# Print the text of the response
print(r.text)
```

?t=hackers —- It means to give the title of all the movies with hacker

The Twitter API and Authentication

To access the Twitter API

- Create a Twitter account
- Log in to Twitter apps and create a new app
- Go to the Keys and Access Tokens tab
- Copy API Key, API Secret, Access Token, and Access Token Secret

You can use these when accessing Twitter API

```
import tweepy, json
access_token='...'
access_token_secret='...'
consumer_key='...'
consumer_secret='...'

# Create Streamin Object
stream = tweepy.Stream(consumer_key, consumer_secret, access_token, access_token_secret)

stream.filter(track=['apples', 'oranges']

# 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())

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

▼ Cleaning Data In Python

Data type Constraints

```
#Import csv and output header
sales = pd.read_csv('sales.csv')
sales.head(2)

#Get Data types of columns
```

```
sales.dtypes

# Get DataFrame information
sales.info()

#Print sum of all Revenue column
sales['Revenue'].sum()     # Returns a large concatenated string

# Remove $ from Revenue column
sales['Revenue'] = sales['Revenue'].str.strip('$')
sales['Revenue'] = sales['Revenue'].astype('int')

# Verify that Revenue is now an integer
assert sales['Revenue'].dtype == 'int'

assert 1+1 == 2 # This will pass

# Categorical Data
df['marriage_status'].describe()

# Convert to categorical
df['marriage_status'] = df['marriage_status'].astype('category')
df.describe()
```

Data range constraints

```
import datetime as dt
today_date = dt.date.today()
user_signups[user_signups['subscription_date'] > dt.date.today()]
```

Data range constraints are the data which is well exceeding the given limit for example rating of movies is done from 1-5 but there is a movie with a rating of 6 so that is basically due to a human error.
How to deal with out-of-range data:

- Dropping data
- Setting custom minimums and maximums
- Treat as missing and impute
- Setting custom value depending on business assumptions

```
import pandas as pd
# Output movies with rating > 5
movies[movies['avg_rating'] > 5] # Displays all the movies with rating greater than 5

# Dropping the values using filtering
movies = movies[movies['avg_rating'] <= 5]
# Drop values using .drop()
movies.drop(movies[movies['avg_rating'] > 5].index, inplace=True)
# Assert results
assert movies['avg_rating'].max() <= 5

# Convert avg_rating > 5 to 5
movies.loc[movies['avg_rating'] > 5, 'avg_rating'] = 5
# Assert statement
assert movies['avg_rating'].max() <= 5

# Data range example
import datetime as dt
import pandas as pd
#output data types
user_signups.dtypes

# Convert to date object
user_signups['subscription_date'] = pd.to_datetime(user_signups['subscription_date']).dt.date

# Drop the data

# Drop values using filtering
user_signups = user_signups[user_signups['subscription_date'] < today_date]
# Drop values using .drop()
user_signups.drop(user_signups['subscription_date'] > today_date].index, inplace=True)
```

```
# Hardcore dates with upper limits

# Drop values using filtering
user_signups.loc[user_signups['subscription_date'] > today_date, 'subscription_date'] = today_date
# Assert is true
assert user_signups.subscription_date.max().date() <= today_date
```

Uniqueness Constraints

```
# Duplicate Values

# Finding Duplicate Values
duplicates = height_weight.duplicated()   # Return booleans for duplicated values
print(duplicates)

# To Directly see duplicated values
height_weight[duplicates]

# Find Duplicated rows
# subset: List of column names to check for duplication
# keep: Whether to keep first('first'), last('last') or all(False) duplicate values
column_names=['first_name', 'last_name', 'address']
duplicates=height_weight.duplicated(subset=column_names, keep=False)

# Sorting
height_weight[duplicates].sort_values(by='first_name')

# Dropping Duplicates
height_weight.drop_duplicates(inplace=True)

# Mean or add the different values in the duplicated rows
# Group by column names and produce statistical summaries
column_names = ['first_name', 'last_name', 'address']
summaries = {'height': 'max', 'weight': 'mean'}
height_weight = height_weight.groupby(by=column_names).agg(summaries).reset_index()
```

Membership Constraints

Test and Categorical Data (Predefined finite set of categories)

e.g: (0,1) or (married, unmarried)

Joins

1. Anti Joins (What is in A and not in B)
2. Inner Joins (What is in both A and B)

```
# Finding inconsistent categories
inconsistent_categoreis = set(study_data['blook_type']).difference(categories['blood_type'])

# Get and print rows with inconsistent categories
inconsistent_rows = study_data['blood_type'].isin(inconsistent_categories)

# Dropping inconsistent categories
inconsistent_data = study_data[inconsistent_rows]
consistent_data = study_data[~inconsistent_rows]

# Print the uniqueness of categories of a single column
categories['cleanliness'].unique()
```

Categorical Variables

```
# Get the marriage status column
marriage_status = demographics['marriage_status']
marriage_status.value_counts()

# Get Value counts on Data Frame
```

```
marriage_status.groupby('marriage_status').count()

# Capitalize
marriage_status['marriage_status'] = marriage_status['marriage_status'].str.upper()
marriage_status['marriage_status'].value_counts()

# Strip all spaces
demographics = demographics['marriage_status'].str.strip()
demographics['marriage_status'].value_counts()

# Collapsing data into categories

# Using qcut()
import pandas as pd
group_names = ['0-200K', '200K-500K', '500K+']
demographics['income_group'] = pd.qcut(demographics['household_income'], q=3, labels=group_names)

# Print income_group column
demographics[['income_group', 'household_income']]

# Using cut()
ranges = [0, 200000, 500000, np.inf]
group_names = ['0-200K', '200K-500K', '500K+']

# Create income group column
demographics['income_group'] = pd.cut(demographics['household_income'], bins=ranges, labels=group_names)
demographics[['income_group', 'household_income']]

# Reducing categories

# Create a mapping dictionary and replace
mapping = {'Microsoft':'DesktopOS', 'MacOS':'DesktopOS', 'Linux':'DesktopOS', 'IOS':'MobileOS', 'Andriod':'MobileOS'}
devices['operating_system'] = devices['operating_system'].replace(mapping)
devices['operating_system'].unique()
```

Cleaning Text Data

```
# Replace data
phones['Phone number'] = phones['Phone number'].str.replace('+', '00')

# Replace phone numbers with lower than 10 digits to NaN
digits = phones['Phone number'].str.len()
phones.loc[digits < 10, 'Phone number'] = np.nan

# Assert min phone number length is 10]
assert sanity_check.min() >= 10

# Assert all numbers do not have + or -
assert phone['Phone number'].str.contains('+|-').any() == False

# Regular expression, Replace letters with nothing
phones['Phone number'] = phones['Phone number'].str.replace(r'\D+', '')
```

Uniformity

```
# Convert to date time
birthdays['Birthday'] = pd.to_datetime(birthday['Birthday'], infer_datetime_format=True, errors='coerce')
```

Crossfield Validation

```
# Sum of all rows
sum_classes = flights[['economy_class', 'business_class', 'first_class']].sum(axis=1)
passenger_equ = sum_classes == flights['total_passengers']
# Filter inconsistent rows
inconsistent_pass = flights[!passenger_equ]
consistent_pass = flights[passenger_equ]

# Date and time
# Convert to datetime and get today's date
users['Birthday'] = pd.to_datetime(users['Birthday'])
```

```
today = dt.date.today()

# For each row in the Birthday column, calculate the year difference
age_manual = today.year - users['Birthday'].dt.year
age_equ = age_manual == users['Age']
```

Completeness

```
# Return missing values
airquality.isna()
# Get summary
airquality.isna().sum()

# Packages to visualize missing data
import missingno as msno
import matplotlib.pyplot as plt

msno.matrix(airquality)
plt.show()

# Isolate missing and complete values aside
missing = airquality[airquality['CO2'].isna()]
complete = airquality[~airquality['CO2'].isna()]

# Describe DataFrames
complete.describe()
missing.describe()

# Visalizing
sorted_airquality = airquality.sort_values(by="Temperature")
msno.matrix(sorted_airquality)
plt.show()

# Deop missing values
airquality_dropped = airquality.dropna(subset = ['CO2'])
airquality_dropped.head()

# Replace missing values
co2_mean = airquality['CO2'].mean()
airquality_imputed = ariquality.fillna({'CO2': co2_mean})
airquality_imputed.head()
```

Comparing Strings

```
# Compare between two strings
import thefuzz as fuzz

# Compare reading vs reading
fuzz.WRatio('Reeding', 'Reading')

# Import process
from thefuzz import process

# Define string
string = 'Houston Rockets vs Los Angeles Lakers'
choices = pd.Series(['dfa', 'adfs', 'asdf', 'afds'])
process.extract(string, choices, limit=2)

# Collapsing categories with string matches
# Consiter a categorical data where there are too many typos so we use string comparison to fix all the categorical data
for state in categories['state']:
  matches = process.extract(state, survey['state'], limit=survey.shape[0])
  for potential_matc in matches:
    if potential_match[1] >= 80:
      survey.loc[survey['state'] == potential_match[0], 'state'] = state
```

Generating Pairs

```
import recordlinkage
```

```
indexer = recorded linkage.Index()
indexer.block('state')
pairs = indexer.index(census_A, census_B)

# Generate pairs
pairs = indexer.index(census_A, census_B)

# Create a compare object
compare_cl = recordlinkage.Compare()

# Find exact matches for pairs of date_of_birht and state
compare_cl.exact('date_of_birth', 'date_of_birth', label='date_of_birth')
compare_cl.exact('state', 'state', label='state')

compare_cl.string('surname', 'surname', threshold=0.85, lable='surname')
compare_cl.string('address_1', 'address_1', threshold=0.85, label='address_1')

potential_matches = compare_cl.compute(pairs, census_A, census_B)
```

Linking DataFrames

```
matches = potential_matches[potential_matches.sum(axis = 1) >= 3]
print(matches)

# Finding Duplicates in cesus_B
census_B_duplicates = census_B[census_B.index.isin(duplicate_rows)]

# Finding new rows in census_B
census_B_new = census_B[~census_B.index.isin(duplicate_rows)]

# Link the DataFrames!
full_census = census_A.append(census_B_new)
```