

# STAT 177, CLASS 4

Richard Waterman

July 2020

# OBJECTIVES

# OBJECTIVES

- Importing data into Python:
  - Data file types:
    - CSV.
    - HTML.
    - JSON.
  - Data locations:
    - A local file.
    - A remote (web) resource.
    - A database.
- Joining datasets.
- Dates and times.

# DATA FORMATS

# DATA FORMATS

- Though typing data into Python directly will work, it is not a realistic exercise for a large data set.
- Typically the data will be stored in an external resource, such as a file or a database.
- It can also be stored in various data file formats.

# CSV FILES

- This is a very popular and portable file format.
- It stands for “comma separated values”.
- Data is stored as plain text.
- Each row of the CSV file corresponds to a case/subject/observation.
- The first row of the file is usually termed a “header” and contains the column/variable names (variable is being used here in a statistical sense).
- For each row the values of the variables are separated by commas.
- Things can get tricky when the data itself contains commas and special characters, and then the strings are quoted to avoid ambiguity.

# EXAMPLE RAW DATA IN CSV FORMAT

```
Patient ID,Sex,Age,Schedule lag,Schedule minutes,Status
P456,male,4,41,30,No show
P126,female,40,29,15,No show
P563,female,23,5,30,No show
P884,male,22,18,30,No show
P102,female,60,1,15,Show
P067,male,50,17,10,Show
P120,male,55,29,30,No show
P943,female,70,3,30,No show
P496,female,58,4,15,Show
P805,male,28,2,30,No show
```

# HTML FORMAT

- This is the format for web page content.
- HTML stands for Hyper Text Markup Language and along with the “http” (Hypertext Transfer Protocol) protocol provided the foundation for the world wide web.
- HTML documents are written in plain text.
- Elements in an HTML document are identified with “tags”.
- Tags can be provided with more information via “attributes”.
- Here is a very simple HTML document and it is “rendered” on the next slide:

```
<!DOCTYPE html>
<html lang="en">
<meta charset="utf-8">
<title>Hello Stat Students!</title>
<body>
  <h1>Important information</h1>
  <p>Work hard.</p>
  <p>Do well.</p>
</body>
</html>
```



**THIS IS HOW THE HTML IS RENDERED.**

**IMPORTANT INFORMATION**

Work hard.

Do well.

# THE “TABLE” ELEMENT

- *Web scraping* is the term applied when you pull down a web page(s) for subsequent analysis.
- You may be interested in an entire web page, but more likely you are interested in some data that the page contains.
- If you are lucky the data is contained in an HTML element called “table”.
- The basic web scraping workflow is:
  - Download the page.
  - Parse the page.
  - Identify table elements.
  - Extract table information into a data frame.
- The parsing can sometimes be very hard, partly because the html itself is a nested/hierarchical structure, and partly because the table may be hard to isolate.
- Luckily there are tools in Python, in particular the *BeautifulSoup* package (which we will see later) that help to do this.

# THE OUTPATIENT DATA AS AN HTML TABLE

- Below is an extract of the first row of the outpatient clinic data in an HTML table. Note the elements “tr” and “td”. They enclose a table *row* and a table *data* point respectively.
- The tags “thead” and “tbody” delineate the table head and body respectively.

```
<table border="1" id="patient_data">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>Sex</th>
      <th>Age</th>
      <th>Schedule lag</th>
      <th>Schedule minutes</th>
      <th>Status</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>P456</th>
      <td>male</td>
```

# COMMENTS

# THE JSON FORMAT.

- json stands for JavaScript Object Notation.
- It can be used for storing and exchanging data between programs.
- When a web browser interacts with a web server, it is quite likely that both are using JavaScript to perform tasks and exchange objects and data.
- json then becomes a natural format for such an exchange.
- json files are in plain text format which makes them portable.
- Because json can be used to represent data, it doesn't have to be used in the context of a server.
- In fact, Jupyter notebooks are saved as json files. Open one with a text editor and have a look!
- pandas can read json files into a data frame using the “read\_json” function.
- On the next slide is the json representation of the outpatient data which looks very much like a Python dict with key value pairs.

# EXAMPLE DATA FILE, STORED USING THE JSON FORMAT

```
{ "Sex":  
  { "0": "male", "1": "female", "2": "female", "3": "male", "4": "female", "5": "male", "6": "male", "7": "female", "8": "  
"Age": { "0": 4, "1": 40, "2": 23, "3": 22, "4": 60, "5": 50, "6": 55, "7": 70, "8": 58, "9": 28 },  
"Schedule lag": { "0": 41, "1": 29, "2": 5, "3": 18, "4": 1, "5": 17, "6": 29, "7": 3, "8": 4, "9": 2 },  
"Scheduled Minutes": { "0": 30, "1": 15, "2": 30, "3": 30, "4": 15, "5": 10, "6": 30, "7": 30, "8": 15, "9": 30 },  
"Status": { "0": "No show", "1": "No show", "2": "No show", "3": "No show", "4": "Show", "5": "Show", "6": "No  
show", "7": "No show", "8": "Show", "9": "No show" } }
```

# DATABASES

- When companies have large amounts of data to manage they will almost always store the data in a *database* .
  - At the top level, a database is simply a structured set of data stored on a computer.
  - But it will be stored in a fashion to optimize retrieval and updating.
  - There are different types of database, but the type we will look at is very popular and called a “relational database”.
  - In this class, we are not learning how to create and maintain databases, but rather, how to extract data from a database for subsequent analysis.
  - The particular type of database we will interact with is called “MySQL” and is an open source database.
  - There are proprietary databases available from the likes of Microsoft, Oracle and Amazon etc.
-

# DATABASE STRUCTURE

- A database is made up of a collection of tables.
- Each table has a set of rows and columns.
- One column will act as a “key”.
- If the tables share a common key, then they can be merged (joined) by using this key.
- The language used to interact with a relational database is called SQL (Structure Query Language).
- We will see just enough SQL to pull data from a database, issuing the commands from within Python and storing the table in a pandas data frame.
- In order to interact with a database you typically need to know:
  - The location of the database.
  - The name of the database.
  - The names of the table(s).
  - Be authorized to access the database by having a username and password.

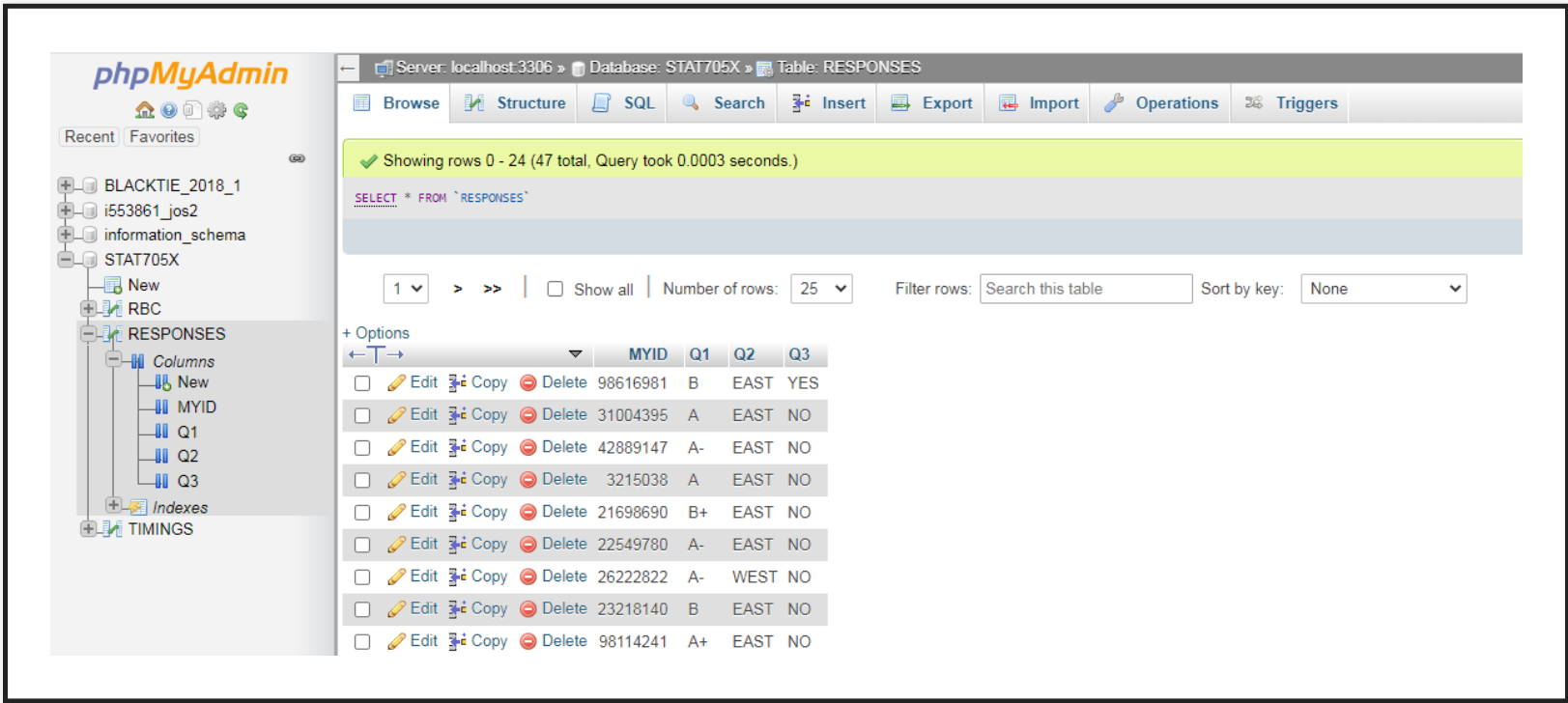


## A VIEW OF A DATABASE

- On the next slide is a view of the databases hosted on the “mathmba” site.
- The database of interest to us is called Stat705X.
- It has a table called “Responses”.
- The table contains columns called “MYID”, “Q1”, “Q2”, “Q3”.

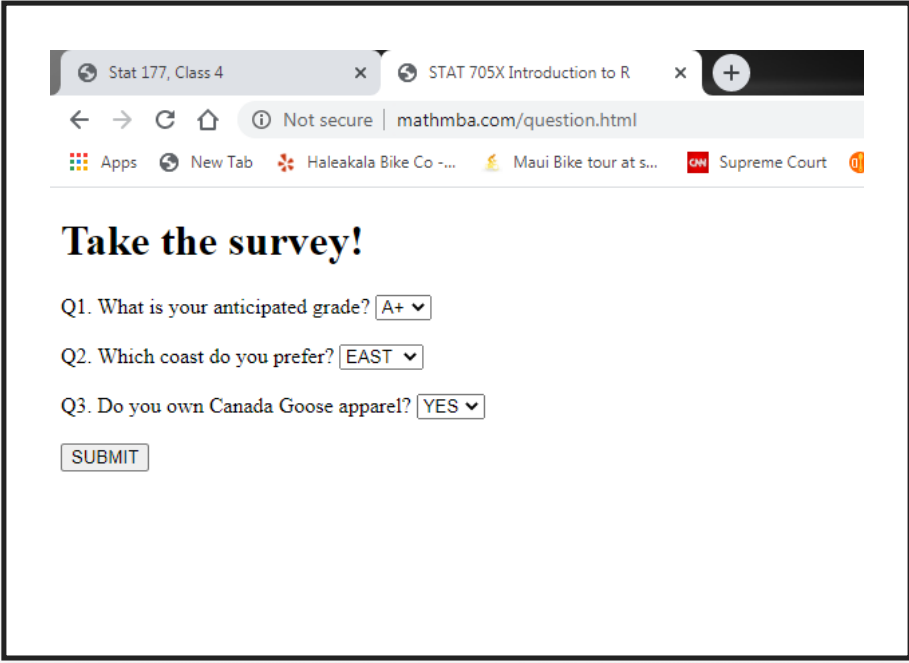
# INTERFACE FOR DATABASE MANAGEMENT

- Below you can see the interface for managing the databases.



# POPULATING THE DATABASE

- You can add to the database by going to the web page: [survey site](#) .
- When you submit this form a script on the server inserts the form data into the database.



The screenshot shows a web browser window with two tabs: 'Stat 177, Class 4' and 'STAT 705X Introduction to R'. The address bar shows 'Not secure | mathmba.com/question.html'. The browser's bookmark bar includes 'Apps', 'New Tab', 'Haleakala Bike Co -...', 'Maui Bike tour at s...', 'CNN', 'Supreme Court', and 'd'. The main content area displays the survey form.

**Take the survey!**

Q1. What is your anticipated grade?

Q2. Which coast do you prefer?

Q3. Do you own Canada Goose apparel?

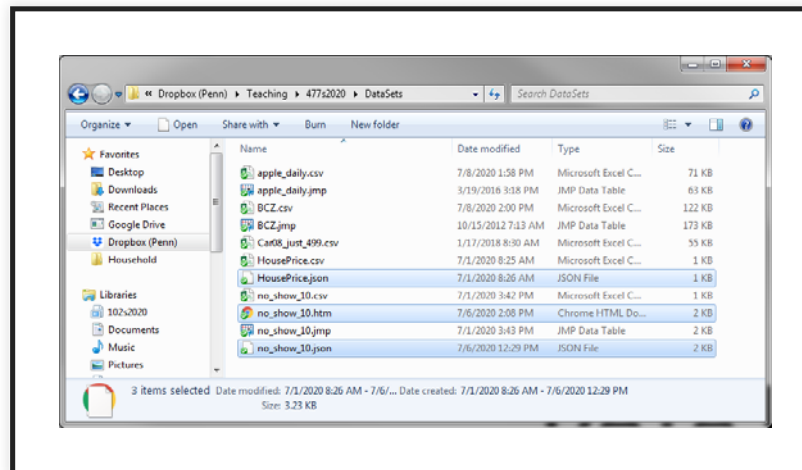
# DATA LOCATIONS

# DATA LOCATIONS

- The data files can be stored either:
  - Locally, or
  - Remotely.
- Locally usually means as a file on your **own** computer.
- Remotely usually means as a file on **another** computer.
- Likewise, a database could be running on your own computer or more likely on another computer.

# LOCAL RETRIEVAL

- I have three versions of the same outpatient data file on my computer (you can download them from Canvas):
  - no\_show\_10.csv
  - no\_show\_10.htm
  - no\_show\_10.json
- We will use pandas commands to read then into Python.
- They will always be imported into a data frame.





## PANDAS “READ\_CSV” COMMAND

- Because this is such a popular data file format, pandas has a specific command to read in such data.
- It is “read\_csv” and will read a csv file right into a pandas data frame.
- The command has a large number of arguments that you can use to get control over how exactly the file is read in.
- This control is important because there may be date columns or numeric columns with commas and special characters, for example \$1,000,000.00, that need to be filtered.
- Date and time variables may also need to be formatted so they are read in correctly.



# CSV IMPORT

```
import pandas as pd
import os

# Change the working directory to where the datasets are stored.
os.chdir('C:\\Users\\richardw\\Dropbox (Penn)\\Teaching\\477s2020\\DataSets')
no_show_01 = pd.read_csv("no_show_10.csv", index_col=0) # The read_csv command. Use the first column as
index.
print(no_show_01)
```

Patient ID	Sex	Age	Schedule lag	Schedule minutes	Status
P456	male	4	41	30	No show
P126	female	40	29	15	No show
P563	female	23	5	30	No show
P884	male	22	18	30	No show
P102	female	60	1	15	Show
P067	male	50	17	10	Show
P120	male	55	29	30	No show
P943	female	70	3	30	No show
P496	female	58	4	15	Show
P805	male	28	2	30	No show

# HTML IMPORT

- pandas will look through the html file, returning a *list* that contains data frames, one for each “table” it finds.
- There happens to be only one table in this file, so we need the data frame from the list in position 0.

```
no_show_02 = pd.read_html("no_show_10.htm", index_col=0)[0] # The [0] pulls off the first (and only) data frame.  
no_show_02.index.name = 'Patient ID' # Give the index a name.  
print(no_show_02)
```

	Sex	Age	Schedule lag	Schedule minutes	Status
Patient ID					
P456	male	4	41	30	No show
P126	female	40	29	15	No show
P563	female	23	5	30	No show
P884	male	22	18	30	No show
P102	female	60	1	15	Show
P067	male	50	17	10	Show
P120	male	55	29	30	No show
P943	female	70	3	30	No show
P496	female	58	4	15	Show
P805	male	28	2	30	No show

# JSON IMPORT

- This one uses the “read\_json” command.
- This format can be tricky because there are different ways to represent the data frame, for example by row, or by column.
- The read\_json command takes an optional argument “orient” that you can use to specify the data structure of the json file.

```
no_show_03 = pd.read_json('no_show_10.json')
no_show_03.index.name = 'Patient ID' # Give the index a name.
print(no_show_03)
```

Patient ID	Sex	Age	Schedule lag	Schedule minutes	Status
P456	male	4	41	30	No show
P126	female	40	29	15	No show
P563	female	23	5	30	No show
P884	male	22	18	30	No show
P102	female	60	1	15	Show
P067	male	50	17	10	Show
P120	male	55	29	30	No show
P943	female	70	3	30	No show
P496	female	58	4	15	Show
P805	male	28	2	30	No show



# OPENING DATA FROM A REMOTE RESOURCE

- The three data files are also in a directory accessible through a web site whose url is “<http://mathmba.com/data/>”.
- This can be used as the “file path” in the “read” command, instead of the local location:

```
no_show_04 = pd.read_csv("http://mathmba.com/data/no_show_10.csv", index_col=0)
print(no_show_04.shape) # "shape" gives you the dimensions of the data frame, the number of rows and
                        # columns.
no_show_05 = pd.read_html("http://mathmba.com/data/no_show_10.htm", index_col=0)[0]
print(no_show_05.shape)
no_show_06 = pd.read_json("http://mathmba.com/data/no_show_10.json")
print(no_show_06.shape)
```

```
(10, 5)
(10, 5)
(10, 5)
```

# READING FROM A DATA BASE

# READING FROM A DATABASE

- You need to install the “mysqlclient” package in order to make the database connection.
- You only have to do the following setup once:
  - Open the Anaconda Powershell prompt.
  - Type in:

```
conda install -c anaconda mysqlclient
```

- conda is a package management system.
  - The -c argument says to search the anaconda “channel” for packages.
  - If you get prompts during the install press “y”.
  - This took about 3 minutes on my laptop.
  - The *mysqlclient* package allows Python to talk specifically to a MySQL database.
- Restart the Python kernel if you have Jupyter notebooks or Spyder already running.

# WHAT’S NEEDED TO CONNECT TO AND QUERY THE DATABASE

- To connect to a database you will typically need to know:

Component	Value
hostname	mathmba.com
Database name	STAT705X
user id	stat705_student
password	\$(h0CC*TtKO~



# CONNECTING TO THE DATABASE

- The following code includes the credentials to connect to the database.
- It requires the hostname, username, password and the name of the database.
- pandas will be able to read the retrieved table straight into a data frame.

```
from sqlalchemy import create_engine # The database toolkit.

# An "engine" is set up to establish a connection to the database when the time comes.
sqlEngine = create_engine('mysql://stat705_student:${h0CC*TtKO~@mathmba.com/STAT705X}')

# The connect method makes the connection.
dbConnection = sqlEngine.connect()

# The key pandas command to put the results from the query right into a data frame is "read_sql".
# It sends the query "SELECT * FROM RESPONSES" over the connection.
survey_from_db = pd.read_sql("SELECT * FROM RESPONSES", dbConnection, index_col='MYID')
print(type(survey_from_db)) # Confirm we have a data frame.
dbConnection.close() # Close the connection to conserve resources.
```

```
<class 'pandas.core.frame.DataFrame'>
```

# REVIEWING THE RETRIEVED DATA

- Note the “head” method to show just a few lines from the data frame.

```
print(survey_from_db.head(10))
```

MYID	Q1	Q2	Q3
98616981	B	EAST	YES
31004395	A	EAST	NO
42889147	A-	EAST	NO
3215038	A	EAST	NO
21698690	B+	EAST	NO
22549780	A-	EAST	NO
26222822	A-	WEST	NO
23218140	B	EAST	NO
98114241	A+	EAST	NO
55528599	A-	EAST	NO

# A MORE COMPLEX QUERY

- If you know some SQL you can do a lot of work on the database itself, which may be more efficient than doing the same thing using Python.
- Below is a more complex SQL query (we can also achieve the same result using Python directly).

```
dbConnection = sqlEngine.connect()

# The key pandas command to put the results from the query right into a data frame.
# It sends the query over the connection.
db_summary = pd.read_sql("SELECT Q1, COUNT(Q1) FROM RESPONSES GROUP BY Q1", dbConnection)
dbConnection.close() # Close the connection to conserve resources.

print(db_summary)
```

	Q1	COUNT (Q1)
0		6
1	A	8
2	A+	8
3	A-	12
4	B	5
5	B+	8
6	C-	1



## USING PANDAS FOR THE SAME GOAL

- The `.value_counts()` method will do a frequency tabulation of a column.

```
survey_from_db.Q1.value_counts()
```

```
A-    12
A+     8
B+     8
A      8
      6
B      5
C-     1
Name: Q1, dtype: int64
```

# MERGING DATA FRAMES

# MERGING DATA FRAMES

- One of the most essential tasks that you quickly come across in practice is to “join/merge” data frames.
- This happens because often the data set you want to analyze, needs to be built from separate pieces.
- For example, you may need to combine marketing data with finance data.

# MERGING DATA FRAMES

- We will obtain two data sets and explore how to merge them using pandas.
- The underlying business problem that the data addresses is to predict whether a web site is compromised, based on features of the web site.
- The features (x-variables) and compromise **Status** (y-variable) happen to be in two separate files, and will need to be joined before a predictive model can be built.
- The key to successful joining is that there is a **key** that is common to both data sets:
- Examples:
  - social security number.
  - DEA number.
  - Account ID.
- If there isn't a suitable key, then there will need to be a fuzzy match which is a potential time-sink/nightmare.



# OBTAIN THE DATA

- The *key* we will use to join the data sets is “ACCOUNTID”.
- This has nothing to do with a *key* in a Python dict structure!

```
# Read in the data sets using pandas. They are both csv files, residing on the web.
import pandas as pd
x_data = pd.read_csv("http://mathmba.com/richardw/x_var_join.csv")
y_data = pd.read_csv("http://mathmba.com/richardw/y_var_join.csv")
print(x_data.head(3))
print("Rows and columns are ", x_data.shape)
print(y_data.head(3))
print("Rows and columns are ", y_data.shape)
```

```

SocialMediaIndex WordPress ACCOUNTID
0                1         YES    INF85
1                0         NO     JEM10
2               12         NO     CYN02
Rows and columns are (9, 3)
AccountID Status
0     NDE65      1
1     INF85      1
2     JEM10      0
Rows and columns are (7, 2)
```

# USING THE PANDAS “MERGE” OPERATION.

- The pandas “merge” operation provides the ability to join data frames.
- You can specify the merge key using the “on” argument.
- It is possible to use the index (row names) as the key as well.
- Unfortunately, the below code fails.
- Can you see why?

```
pd.merge(x_data,y_data, on = 'ACCOUNTID')
```

```
-----  
KeyError
```

```
Traceback (most recent call last)
```

```
<ipython-input-10-a33e71cc2525> in <module>
```

```
----> 1 pd.merge(x_data,y_data, on = 'ACCOUNTID')
```

```
D:\anaconda\lib\site-packages\pandas\core\reshape\merge.py in merge(left, right, how, on, left_on, right_on, left_index, right_index, sort, suffixes, copy, indicator, validate)
```

```
84         copy=copy,
```

```
85         indicator=indicator,
```

```
---> 86         validate=validate,
```

```
87     )
```

```
88     return op.get_result()
```



# MERGING THE DATA FRAMES

- The reason the command failed was that the Account ID was all upper case in one data frame and lower case in the other.
- We could rename one of the columns, so that the key's name agrees across the data frames or use a more explicit version of the merge command. We will do both.

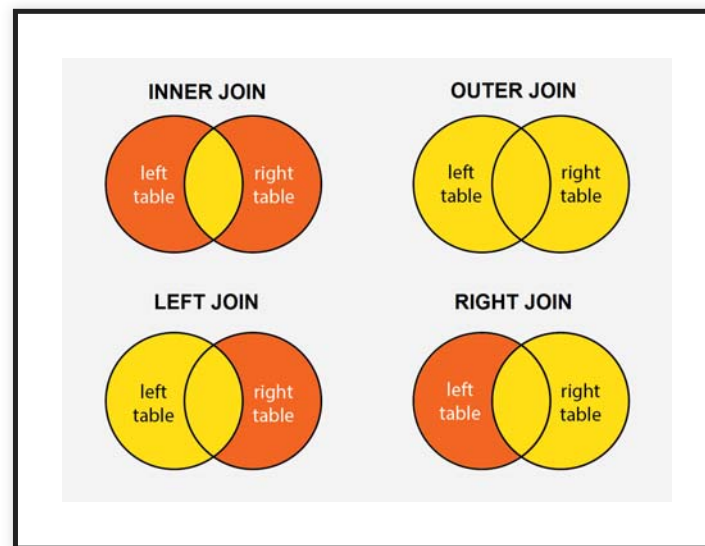
```
# Below we specify the name of the key in each data frame with "left_on" and "right_on".
inner_join_data = pd.merge(x_data,y_data, left_on = 'ACCOUNTID', right_on = 'AccountID')
print(inner_join_data)
print("Rows and columns are", inner_join_data.shape)
```

	SocialMediaIndex	WordPress	ACCOUNTID	AccountID	Status
0	1	YES	INF85	INF85	1
1	0	NO	JEM10	JEM10	0
2	5	NO	ASC53	ASC53	1
3	12	YES	QUT35	QUT35	0
4	34	NO	XCT80	XCT80	0

Rows and columns are (5, 5)

# DIFFERENT TYPES OF JOINS

- Notice how only the IDs that exist in both data frames are returned.
- This is called an “inner join” in database speak.
- The other types of joins are called “left”, “right” and “outer”.
- The yellow piece in the below graphic shows the rows returned with the different joins.



## DIFFERENT TYPES OF JOINS

- left ensures that all IDs from the left table are in the join, entering NaN if there is no match for the key in the right table.
- right ensures that all IDs from the right table are in the join, entering NaN if there is no match in the left table.
- outer ensures that all IDs from both tables are in the join.
- We use the argument “how” to specify the type of join we want.

# LEFT JOIN

```
# Note the 'how="left"' argument.
left_join_data = pd.merge(x_data,y_data, left_on = 'ACCOUNTID', right_on = 'AccountID', how="left")
print(left_join_data)
print("Rows and columns are", left_join_data.shape)
```

	SocialMediaIndex	WordPress	ACCOUNTID	AccountID	Status
0	1	YES	INF85	INF85	1.0
1	0	NO	JEM10	JEM10	0.0
2	12	NO	CYN02	NaN	NaN
3	5	NO	ASC53	ASC53	1.0
4	12	YES	QUT35	QUT35	0.0
5	76	NO	DIU03	NaN	NaN
6	34	NO	XCT80	XCT80	0.0
7	18	NO	WCV02	NaN	NaN
8	21	NO	PYT29	NaN	NaN

Rows and columns are (9, 5)

- Every row in x is in the resulting data frame, so it has nine rows just as the x-data has.

# RIGHT JOIN

```
# Note the 'how="right"' argument.  
right_join_data = pd.merge(x_data,y_data, left_on = 'ACCOUNTID', right_on = 'AccountID', how="right")  
print(right_join_data)  
print("Rows and columns are", right_join_data.shape)
```

	SocialMediaIndex	WordPress	ACCOUNTID	AccountID	Status
0	1.0	YES	INF85	INF85	1
1	0.0	NO	JEM10	JEM10	0
2	5.0	NO	ASC53	ASC53	1
3	12.0	YES	QUT35	QUT35	0
4	34.0	NO	XCT80	XCT80	0
5	NaN	NaN	NaN	NDE65	1
6	NaN	NaN	NaN	XRO15	0

Rows and columns are (7, 5)

- Every row in y is in the resulting data frame, so it has seven rows just as the y-data has.



# OUTER JOIN

```
# Note the 'how="outer"' argument.  
outer_join_data = pd.merge(x_data,y_data, left_on = 'ACCOUNTID', right_on = 'AccountID', how="outer")  
print(outer_join_data)  
print("Rows and columns are ", outer_join_data.shape)
```

	SocialMediaIndex	WordPress	ACCOUNTID	AccountID	Status
0	1.0	YES	INF85	INF85	1.0
1	0.0	NO	JEM10	JEM10	0.0
2	12.0	NO	CYN02	NaN	NaN
3	5.0	NO	ASC53	ASC53	1.0
4	12.0	YES	QUT35	QUT35	0.0
5	76.0	NO	DIU03	NaN	NaN
6	34.0	NO	XCT80	XCT80	0.0
7	18.0	NO	WCV02	NaN	NaN
8	21.0	NO	PYT29	NaN	NaN
9	NaN	NaN	NaN	NDE65	1.0
10	NaN	NaN	NaN	XRO15	0.0
Rows and columns are (11, 5)					

# DUPLICATE KEYS

- If you have the same key twice, the join will match both instances.

```
# Insert a new row in the data frame with the .append method.
new_x_data = x_data.append({'SocialMediaIndex': 0, 'WordPress': 'NO', 'ACCOUNTID': 'INF85'},
                             ignore_index=True)

# Now do an inner join and note how "INF85" is the new data frame twice.:
print(pd.merge(new_x_data,y_data, left_on = 'ACCOUNTID', right_on = 'AccountID'))
```

	SocialMediaIndex	WordPress	ACCOUNTID	AccountID	Status
0	1	YES	INF85	INF85	1
1	0	NO	INF85	INF85	1
2	0	NO	JEM10	JEM10	0
3	5	NO	ASC53	ASC53	1
4	12	YES	QUT35	QUT35	0
5	34	NO	XCT80	XCT80	0

# RENAMING A COLUMN

- Recall that the columns that were used as keys had different names.
- We will rename the “AccountID” column in `y_data` as “ACCOUNTID”.

```
# Rename a column. The "inplace" overwrites the existing data frame.  
# The axis argument specifies if it is the rows or column index you wish to change. 0 = rows, 1 =  
# columns.  
# The old and new names are specified in a dict structure.  
y_data.rename({'AccountID': 'ACCOUNTID'}, axis = 1, inplace = True)  
print(y_data)
```

	ACCOUNTID	Status
0	NDE65	1
1	INF85	1
2	JEM10	0
3	ASC53	1
4	QUT35	0
5	XRO15	0
6	XCT80	0

# A SIMPLER MERGE STATEMENT

- With the column names for the two keys the same, the inner join can be accomplished with just:

```
# pandas looks for the common column name and uses that as the key automatically.  
print(pd.merge(x_data, y_data))
```

	SocialMediaIndex	WordPress	ACCOUNTID	Status
0	1	YES	INF85	1
1	0	NO	JEM10	0
2	5	NO	ASC53	1
3	12	YES	QUT35	0
4	34	NO	XCT80	0

# DATES AND TIMES

# DATE AND TIMES

- Being able to read and manipulate dates and times is an essential skill, especially with business data.
- Dates can be quite challenging because of the many representations and conventions that can differ within and across countries.
- For example, what day is "070820"?
- The problem is that it depends on the convention you are following.
- It could be July 8th, 2020.
- it could be August 7th, 2020.
- It could even be August 20, 1907!
- Clearly we need some way of standardizing dates and being able to move between formats.
- Pandas will give us these tools.

# THE INTERNAL REPRESENTATION OF A POINT IN TIME

- We will use the datetime library that will give us plenty of date and time functionality.
- You can ask Python for the current date and time.

```
from datetime import datetime, date, time

today = date.today()
print("Today's date:", today)

now = datetime.now()
print("Now =", now)
```

```
Today's date: 2020-07-14
Now = 2020-07-14 15:25:58.259669
```

# THE EPOCH

- Many computer systems define time as the number of seconds from 1970-1-1 00:00:00, which is known as epoch time.
- Here's a link to an epoch time converter: <https://www.epochconverter.com/>

```
# The .timestamp method returns the number of seconds since 1970-1-1 00:00:00  
epoch_time = datetime.now().timestamp()  
print(epoch_time)
```

```
1594754758.264669
```



# READING DATES WITH PANDAS

- To create a datetime object we can use the pandas “to\_datetime” method.
- The critical element is the format argument, which tells Python how to interpret the date string.
- In this instance it says the first two numbers are the month (%m), then the next two the day of the month (%d) and finally the 4 digit year (%Y)

```
time_one = pd.to_datetime('03211989', format="%m%d%Y")
print(time_one)
print(type(time_one)) # It is of type "Timestamp"

time_two = pd.to_datetime('07132020', format="%m%d%Y")
print(time_two)
```

```
1989-03-21 00:00:00
<class 'pandas._libs.tslibs.timestamps.Timestamp'>
2020-07-13 00:00:00
```

# THE DIFFERENCE BETWEEN TWO TIMES

- By storing dates and times internally as seconds from the epoch, it is easy to find out how far two dates are from one another, simply using subtraction:

```
print(time_two - time_one)
print(type(time_two - time_one)) # It is of type "Timedelta"
```

```
11437 days 00:00:00
<class 'pandas._libs.tslibs.timedeltas.Timedelta'>
```

## USING AN ALTERNATIVE TIME FORMAT

- Time can come in different formats, for example ‘Jan 13, 2020’.
- But using appropriate formatting, we can create a date time object here too.
- All the formatting rules can be found here:

<https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior>

```
# The time format used by Yahoo Finance stock data downloads (see homework 2).  
time_three = pd.to_datetime('Jan 13, 2020', format="%b %d, %Y")  
print(time_three)
```

```
2020-01-13 00:00:00
```

# OBTAIN THE FULL OUTPATIENT DATA SET

- The data set is called “Outpatient.csv” and is available for download from the Canvas site.
- Our goal is to find the average Schedule lag, where schedule lag is defined as the difference between appointment date and schedule date.
- For example, if the appointment is made on January 01 2020, for February 28, 2020 then the schedule lag is 58 days ( $30 + 28$ ).
- Note that the dates have been read in as type = object, which means they are currently viewed as strings and not dates.

# REVIEWING THE OUTPATIENT DATASET

- It is a good idea to get a sense of the dimensions and data types in the DataFrame:

```
import os
os.chdir('C:\\Users\\richardw\\Dropbox (Penn)\\Teaching\\477s2020\\DataSets')
outpatient_data = pd.read_csv("Outpatient.csv", )

# Basic top level data summaries.
print(outpatient_data.shape) # Rows and columns.
print(outpatient_data.columns) # Column names.
print(outpatient_data.dtypes) # Column data types.
```

```
(3699, 9)
Index(['PID', 'SchedDate', 'ApptDate', 'Dept', 'Language', 'Sex', 'Age',
      'Race', 'Status'],
      dtype='object')
PID          object
SchedDate    object
ApptDate     object
Dept         object
Language     object
Sex          object
Age          object
Race         object
Status       object
dtype: object
```

# THE PARSE\_DATES ARGUMENT

- When you know you have a date(s) column, it is always worth trying the “parse\_dates” argument to read\_csv.
- It will attempt to discern the date format and read in the date data as a datetime object.
- It is successful here, but not always.
- You may have to use the ‘pd.to\_datetime’ function when the optimistic approach fails.

```
outpatient_data = pd.read_csv("Outpatient.csv", parse_dates=['SchedDate', 'ApptDate']) # Pass in the
    date columns.
print(outpatient_data.dtypes) # This looks better!
```

```
PID                object
SchedDate          datetime64[ns]
ApptDate           datetime64[ns]
Dept              object
Language           object
Sex               object
Age              object
Race             object
Status           object
dtype: object
```

## FINDING THE SCHEDULE LAG

- Because pandas uses “vectorized” functions, we can find the schedule lag for all the rows without the need for a “for” loop:

```
outpatient_data['ScheduleLag'] = outpatient_data['ApptDate'] - outpatient_data['SchedDate']  
  
# What is the average schedule lag? It is about a month.  
outpatient_data['ScheduleLag'].mean()
```

```
Timedelta('33 days 11:31:46.569343')
```

# DAY-OF-THE-WEEK AND MONTH-OF-THE-YEAR EXTRACTIONS

- It is very common to want to find the day of the week, or month of the year.
- pandas has methods to do this: *dt.day\_name* and *dt.month\_name* .
- By using the “locale” argument, you can even specify a language.
- Below I am using French.
- If you don't specify a *locale* , Python will use whatever default your computer is set to.

```
outpatient_data['ApptDayOfWeek'] = outpatient_data['ApptDate'].dt.day_name(locale='fr')
outpatient_data['ApptMonthOfYear'] = outpatient_data['ApptDate'].dt.month_name(locale='fr')
print(outpatient_data.columns)
```

```
Index(['PID', 'SchedDate', 'ApptDate', 'Dept', 'Language', 'Sex', 'Age',
      'Race', 'Status', 'ScheduleLag', 'ApptDayOfWeek', 'ApptMonthOfYear'],
      dtype='object')
```



# BASIC SUMMARIES ON THE DATA FRAME

# BASIC SUMMARIES ON THE DATA FRAME

- The `value_counts()` method, which we have seen before counts the frequency of the values in a column.
- The code below tells you how many appointments there are on each day of the week.

```
outpatient_data['ApptDayOfWeek'].value_counts()
```

```
Mercredi      590  
Jeudi         569  
Dimanche     544  
Lundi        539  
Mardi        518  
Vendredi     513  
Samedi       426  
Name: ApptDayOfWeek, dtype: int64
```

# A FIRST LOOK AT THE GROUPBY COMMAND

- One of the fundamental tasks of pandas is to make summarizing and reviewing data as simple and flexible as possible.
- A key command is “groupby” which will identify which rows are in each level of a column.
- These rows can then be summarized in some fashion, for example with their count or average.

```
# This will count the number of appointments in each Status level.  
print(outpatient_data.Status.value_counts())
```

```
Arrived      2167  
Cancelled    796  
No Show      526  
Bumped       209  
Rescheduled    1  
Name: Status, dtype: int64
```

# THE NUMBER OF APPOINTMENTS IN EACH STATUS CATEGORY BY DAY OF WEEK

- By using the *groupby* function, we can do the same calculation, but now by day-of-the-week:

```
# Using groupby gives the value counts by day of the week.  
outpatient_data.groupby('ApptDayOfWeek').Status.value_counts()
```

ApptDayOfWeek	Status	
Dimanche	Arrived	319
	Cancelled	109
	No Show	88
	Bumped	28
Jeudi	Arrived	328
	Cancelled	133
	No Show	79
	Bumped	29
Lundi	Arrived	309
	Cancelled	122
	No Show	79
	Bumped	29
Mardi	Arrived	280
	Cancelled	130

# OBTAINING PERCENTAGES RATHER THAN FREQUENCIES

```
# The optional argument, "normalize" will return a proportion. They add to 1 for each day of the week.
outpatient_data.groupby('ApptDayOfWeek').Status.value_counts(normalize=True)
```

ApptDayOfWeek	Status	
Dimanche	Arrived	0.586397
	Cancelled	0.200368
	No Show	0.161765
	Bumped	0.051471
Jeudi	Arrived	0.576450
	Cancelled	0.233743
	No Show	0.138840
	Bumped	0.050967
Lundi	Arrived	0.573284
	Cancelled	0.226345
	No Show	0.146568
	Bumped	0.053803
Mardi	Arrived	0.540541
	Cancelled	0.250965

# A LOOK AT THE INDEX

- Recall that the row index can be thought of as simply the row names.
- But after the groupby, it is known as a “MultiIndex”.

```
# Save the result into a Series object with a MultiIndex.
category_props = outpatient_data.groupby('ApptDayOfWeek').Status.value_counts(normalize=True)
print(type(category_props))
print(category_props.index)
```

```
<class 'pandas.core.series.Series'>
MultiIndex([( 'Dimanche',    'Arrived'),
            ( 'Dimanche',    'Cancelled'),
            ( 'Dimanche',    'No Show'),
            ( 'Dimanche',    'Bumped'),
            (   'Jeudi',      'Arrived'),
            (   'Jeudi',      'Cancelled'),
            (   'Jeudi',      'No Show'),
            (   'Jeudi',      'Bumped'),
            (   'Lundi',      'Arrived'),
            (   'Lundi',      'Cancelled'),
            (   'Lundi',      'No Show'),
            (   'Lundi',      'Bumped'),
            (   'Mardi',      'Arrived'),
            (   'Mardi',      'Cancelled')])
```

# EXTRACTING ELEMENTS WHEN THERE IS A MULTIINDEX

- Say we want to pull out just the ‘Cancelled’ proportions from each day-of-the-week.
- This a cross section of the result, where we are zooming in on just the “Cancelled” values.
- This can be achieved through the ‘.xs’ method:

```
# xs stands for "cross section" and 'level' specifies which level of the MultiIndex to look in.  
category_props.xs('Cancelled', level=1)
```

```
ApptDayOfWeek  
Dimanche      0.200368  
Jeudi         0.233743  
Lundi         0.226345  
Mardi         0.250965  
Mercredi      0.188136  
Samedi        0.206573  
Vendredi      0.200780  
Name: Status, dtype: float64
```

## AN ALTERNATIVE APPROACH USING CROSSTABS

- Remember there are always a 101 ways to do the same thing.
- pandas has a crosstabs function (cross-tabulation) that will cross-classify the levels of two categorical variables.
- The 'normalize' argument, controls whether you normalize by overall total, column total or row total.
- In the argument to xs, 'index' identifies which variable will correspond to the rows of the table and 'columns' which one will be in the columns.
- Below, we normalize by 'column', so the proportions in a column add to 1.
- The command will produce appointment status by day of the week.



# CROSSTABS OUTPUT

```
pd.set_option('precision', 3) # Control output formatting for floats to 3 digits of precision.
print(pd.crosstab(index = outpatient_data.Status, columns = outpatient_data.ApptDayOfWeek, normalize =
'columns'))
```

ApptDayOfWeek	Dimanche	Jeudi	Lundi	Mardi	Mercredi	Samedi	Vendredi
Status							
Arrived	0.586	0.576	0.573	0.541	0.625	0.622	0.579
Bumped	0.051	0.051	0.054	0.058	0.063	0.049	0.068
Cancelled	0.200	0.234	0.226	0.251	0.188	0.207	0.201
No Show	0.162	0.139	0.147	0.149	0.124	0.122	0.152
Rescheduled	0.000	0.000	0.000	0.002	0.000	0.000	0.000

# CLASS SUMMARY

# SUMMARY

- Importing data into Python:
  - Data file types.
    - CSV.
    - HTML.
    - JSON.
  - Data locations:
    - A local file.
    - A remote (web) resource.
    - A database.
- Joining datasets.
- Dates and times.

# NEXT TIME

## NEXT TIME

- Writing basic functions.
- Common data cleaning activities.
- More on the “groupby” command.
- From transactional to behavioral data sets.