Data Wrangling and Curation

Claudio Delrieux

Data Wrangling and Curation, Part I

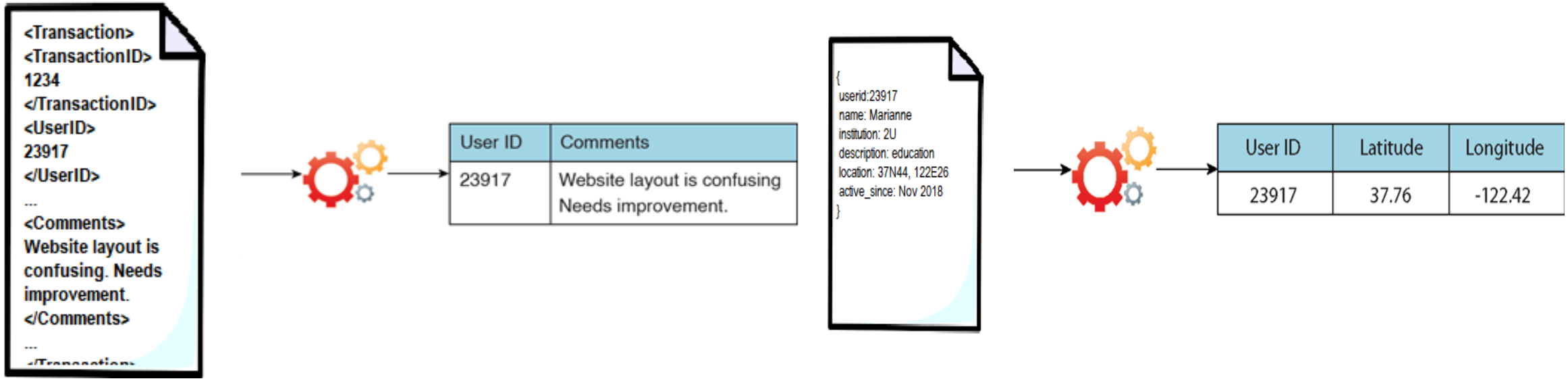
- Data may be munged into several possible formats, for example, CSV, SQL, Json, XML, Free text, NoSQL blobs, Web Scraping, platform APIs (for example, Twitter or Facebook), or sometimes raw dataframes.
- In each case, different readers must be used, and data syntax understanding is required to verify a correct record/field access.

Data Wrangling and Curation, Part II

- Filter out corrupt/missing data (though, keep a copy of the original ingested file just in case).
- If possible, integrate easy-to-parse metadata within the filtered file to keep track of the process and to fold back if necessary.

Data Wrangling and Curation, Part III

- Extract data fields from the original format into the intended format.

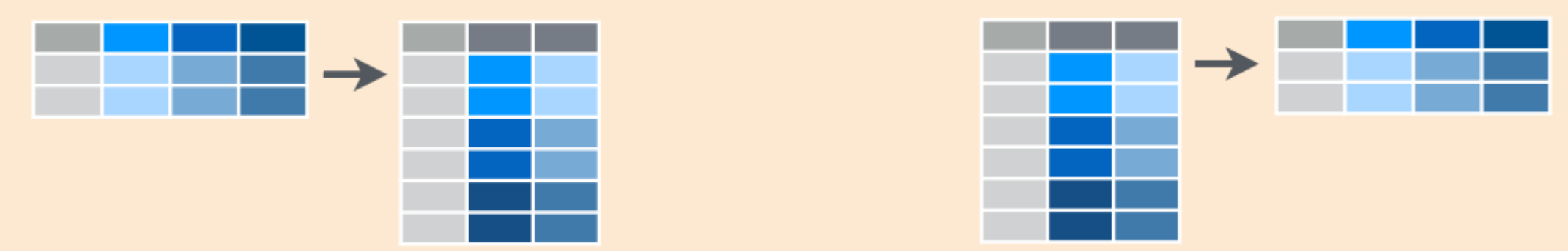


Data Wrangling and Curation, Part IV

- Data field values sometimes require checking data representation formats, and validation that the values are plausible.
- Corrupt or missing data may be filtered out, or plausible values can be ascribed via imputation.
- If two or more datasets are used together, sometimes reconciliation rules have to be adopted.

Data Wrangling and Curation, Part V

- Finally, represent the dataset in a format that is adequate for the analysis steps. This may require reshaping the dataset (for example, gathering columns into rows, or vice versa, spreading rows into columns). Again, keep a copy of the original dataset and add readable metadata.



The End

Data Preparation Tools

Claudio Delrieux

Data Preparation Tools, Part I

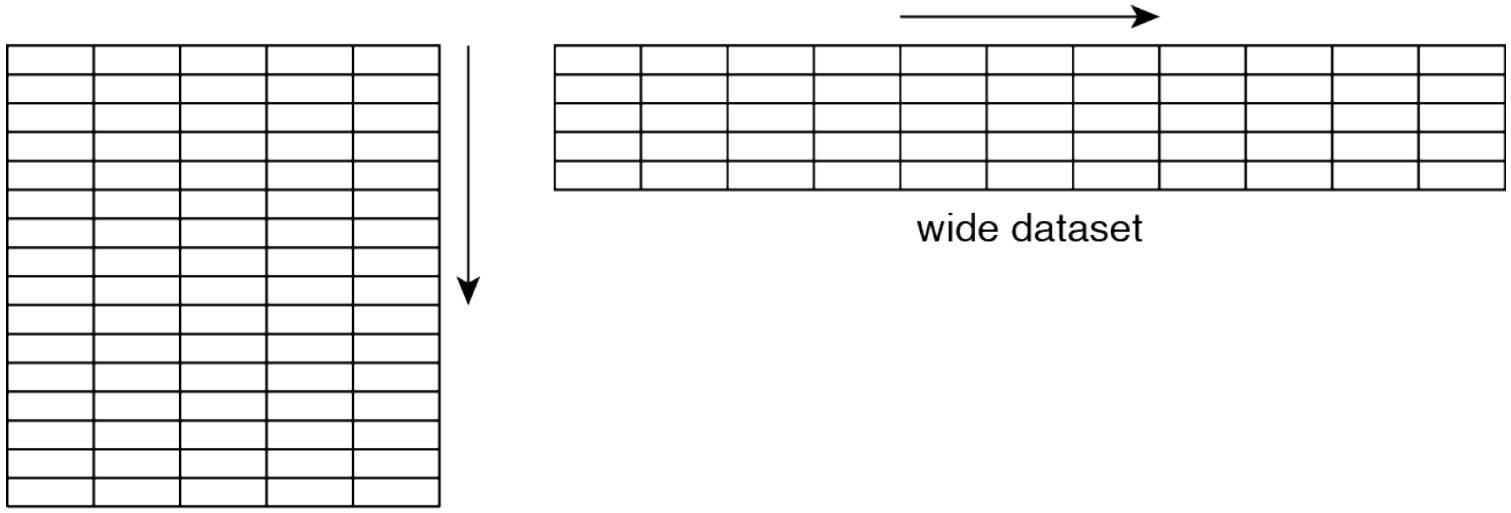
- Among the most used libraries for data manipulation is *pandas*, which serves many purposes.
  - Reading and writing data in a variety of popular formats
  - Data alignment, missing values handling, reshaping, grouping, and mutability
  - Many other that we will see

Data Preparation Tools, Part II

- Identifying outliers and bad data is probably one of the most difficult parts of data cleanup (how to handle outliers or bad records and considering their removal).
- Detecting bad data can be performed manifold: range checking, parametric statistics (i.e., z-values), percentiles.

Data Preparation Tools, Part III

- Depending on the dataset shape, and the kind of data analysis to be performed, we may decide to filter out the whole record, or just to assign a supposed (imputed) value.



Data Preparation Tools, Part IV

- Data extraction and representation require ad hoc programming. Python provides simple **percent** formatters for data types, and the **format** method, mainly to convert complex objects into more manageable strings.

Data Preparation Tools, Part V

- **Imputation** is the process of replacing missing or corrupt data with substituted values. Most of the time, we proceed this way only for data fields, but other times (for example, in time series), complete record imputation may be required. This also requires ad hoc programming strongly dependent on the dataset and analysis tasks.

Data Preparation Tools, Part VI

- Many times the dataset(s) require(s) format changes prior to the data analysis steps. For instance, if only a few attributes (features) are required, or if two or more tables have to be joined together.

The End

Data Preparation Example I

Claudio Delrieux

Data Preparation Example I, Part I

We will illustrate some of the concepts, retrieving and processing raw spreadsheet data from NASA Global Surface Air Temperature Anomaly. The dataset contains data from 1880 to 2019 of the change in global average surface air temperature, as compared to the average global temperature measured in the period 1951 to 1980.

Data Preparation Example I, Part II

After opening and visualizing the file, we notice the format is not easy to use.

```
[15] source_data_url_temp_anomaly = 'http://data.giss.nasa.gov/gistemp/graphs_v3/Fig.A.txt'
     resp_temp_anomaly = requests.get(source_data_url_temp_anomaly)
     resp_temp_anomaly.text
```

```
'Global Surface Air Temperature Anomaly (C) (Base: 1951-1980)\n--------------------------
-------------------------------\n 1880     -0.47          *\n 1881     -0.45          *\n
-0.59\n 1886     -0.67     -0.61\n 1887     -0.71     -0.53\n 1888     -0.48     -0.54\n
-0.54\n 1893     -0.56     -0.49\n 1894     -0.43     -0.44\n 1895     -0.36     -0.38\n
-0.25\n 1900     -0.16     -0.29\n 1901     -0.17     -0.29\n 1902     -0.38     -0.34\n
-0.41\n 1907     -0.53     -0.38\n 1908     -0.38     -0.38\n 1909     -0.40     -0.40\n..
```

Data Preparation Example I, Part III

Parsing can be performed by means of regular expressions, using RegExr or another online regular expression handling tool.

```
[3] pattern_temp_anomaly     = re.compile(r'^ +(\d{4}) +(-?\d+\.\d+|[*]?) +(-?\d+\.\d+|[*]?)', re.MULTILINE)
    data_tuples_temp_anomaly = re.findall(pattern_temp_anomaly, resp_temp_anomaly.text)
    data_tuples_temp_anomaly
```

```
('1967', '-0.02', '-0.07'),
('1968', '-0.10', '-0.02'),
('1969', '0.01', '-0.02'),
('1970', '0.07', '-0.02'),
('1971', '-0.07', '0.04'),
('1972', '-0.02', '0.03'),
('1973', '0.22', '0.02'),
('1974', '-0.04', '0.00'),
('1975', '0.02', '0.05'),
('1976', '-0.17', '0.03'),
```

Data Preparation Example I, Part IV

We can now load it as a Pandas dataframe.

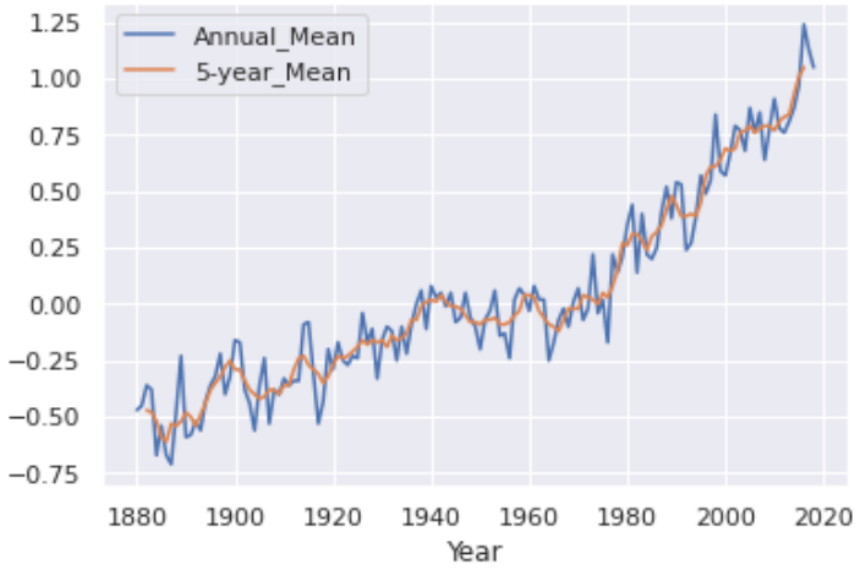
```
# import pandas to use pandas DataFrame
import pandas as pd
pd.set_option("max_rows", 8)
# create the dataframe
df_temp_anomaly = pd.DataFrame(data_tuples_temp_anomaly, columns =['Year', 'Annual_Mean', '5-year_Mean'])
# replace '*' with NaNs
df_temp_anomaly = df_temp_anomaly.replace('*', 'nan')
# use a dictionary of types to convert specific columns
types_dict_temp_anomaly = {'Year': int, 'Annual_Mean': float, '5-year_Mean': float}
df_temp_anomaly = df_temp_anomaly.astype(types_dict_temp_anomaly)
df_temp_anomaly
```

	Year	Annual_Mean	5-year_Mean
0	1880	-0.47	NaN
1	1881	-0.45	NaN
2	1882	-0.36	-0.47
3	1883	-0.38	-0.48
...	...	...	...
136	2016	1.24	1.05

Data Preparation Example I, Part V

We can now plot the time series.

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(style="darkgrid")
df_temp_anomaly.plot.line(x='Year', y=['Annual_Mean','5-year_Mean'])
```



Data Preparation Example I, Part VI

Suppose now we wish to correlate this warming with the observed atmospheric CO^2 as published in this source file: http://data.giss.nasa.gov/modelforce/ghgases/Fig1A.ext.txt

Data Preparation Example I, Part VII

The file looks like this.

Global Mean CO2 Mixing Ratios (ppm): Observations

Data Source	Year	MixR	Yar	MixR	Data Source	Year	MixR		Year	MixR
Ice-	1850	285.2	1900	295.7		1950	311.3		2000	369.64
Core	1851	285.1	1901	296.2		1951	311.8		2001	371.15
Data	1852	285.0	1902	296.6		1952	312.2		2002	373.15
Adjus-	1853	285.0	1903	297.0		1953	312.6		2003	375.64
ted	1854	284.9	1904	297.5		1954	313.2	NOAA/	2004	377.44
for	1855	285.1	1905	298.0		1955	313.7	ESRL/	2005	379.46
Global	1856	285.4	1906	298.4		1956	314.3	trends	2006	381.59
Mean	1857	285.6	1907	298.8		1957	314.8	change	2007	383.37
	1858	285.9	1908	299.3	SIO	1958	315.34	added	2008	385.46
	1859	286.1	1909	299.7	Mauna	1959	316.18	to	2009	386.95
	1860	286.4	1910	300.1	Loa	1960	317.07	2003	2010	389.21
	1861	286.6	1911	300.6	&	1961	317.73	data	2011	391.15

Data Preparation Example I, Part VIII

The file contains two sections: *observations* and *future scenarios;* the latter section we will not consider.

Global Mean CO2 Mixing Ratio (ppm): Future Scenarios

Alternative Scenario				2 Degree C Scenario			
Year	MixR	Year	MixR	Year	MixR	Year	MixR
2000	370.0	2050	445.0	2000	370.0	2050	486.2
2001	371.7	2051	446.2	2001	371.7	2051	489.2
2002	373.4	2052	447.4	2002	373.4	2052	492.1
2003	375.1	2053	448.5	2003	375.1	2053	494.9

Data Preparation Example I, Part IX

Again we use RegExr to extract the tuples (note they are not in chronological order).

```
[9]  # Get the tuples
     pattern_co2     = re.compile(r'(\b\d{4})   (\d{3}\.\d{1,2}\b)')
     data_tuples_co2 = re.findall(pattern_co2, excerpt)
     data_tuples_co2
```

```
[('1850', '285.2'),
 ('1900', '295.7'),
 ('1950', '311.3'),
 ('2000', '369.64'),
 ('1851', '285.1'),
 ('1901', '296.2'),
 ('1951', '311.8'),
 ('2001', '371.15'),
 ('1852', '285.0'),
 ('1902', '296.6'),
 ('1952', '312.2'),
 ('2002', '373.15'),
 ('1853', '285.0'),
 ('1903', '297.0'),
```

Data Preparation Example I, Part X

We convert the tuples into a dataframe, using a dictionary to convert the data types.

```
df_co2 = pd.DataFrame(data_tuples_co2, columns =['Year', 'MixR'])
# use a dictionary of types to convert specific columns
types_dict_co2 = {'Year': int, 'MixR': float}
df_co2 = df_co2.astype(types_dict_co2)
# sort by year
df_co2 = df_co2.sort_values(by=['Year'])
df_co2
```

	Year	MixR
0	1850	285.20
4	1851	285.10
8	1852	285.00
12	1853	285.00

Data Preparation Example I, Part XI

We merge the dataframes, drop the five-year average, and index both frames with regard to year.

```
# We merge both dataframes using Year as the merging key
df_temp_anomaly_co2 = pd.merge(df_co2, df_temp_anomaly, on='Year')
# Drop the 5-year_Mean column
df_temp_anomaly_co2 = df_temp_anomaly_co2.drop(columns=['5-year_Mean'])
# Set Year as the index
df_temp_anomaly_co2 = df_temp_anomaly_co2.set_index('Year')
df_temp_anomaly_co2
```

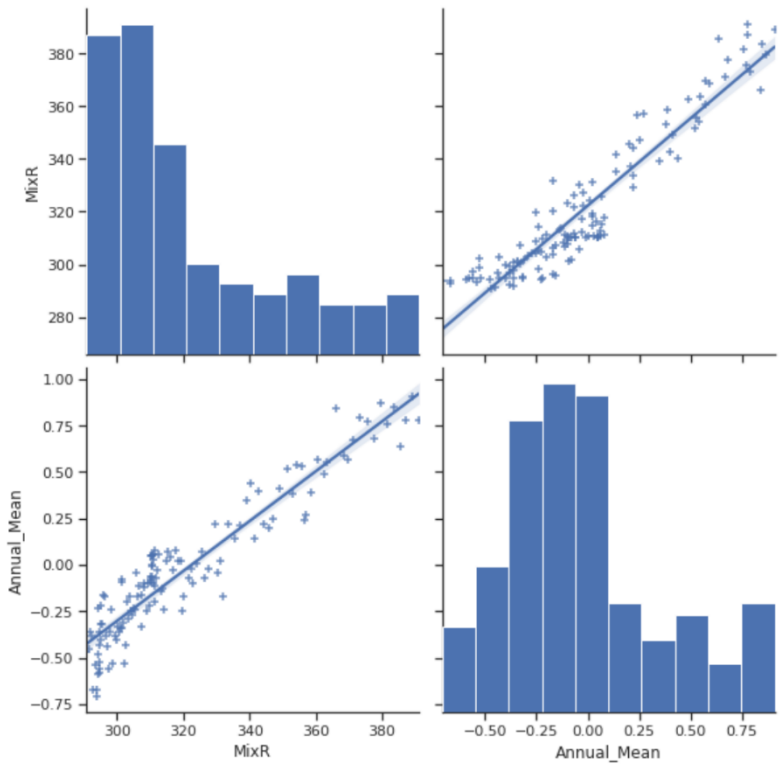
Year	MixR	Annual_Mean
1880	290.80	-0.47
1881	291.40	-0.45
1882	292.00	-0.36
1883	292.50	-0.38
...	...	...

Data Preparation Example I, Part XII

Finally, we plot both frames as scatterplots.

```
[ ]  sns.set(style="ticks", color_codes=True)
     sns.pairplot(df_temp_anomaly_co2, kind="reg", markers="+",height=4)
```

```
<seaborn.axisgrid.PairGrid at 0x7f592bc25400>
```



The End

Data Preparation Example II

Claudio Delrieux

Data Preparation Example II, Part I

Another example: reading an Excel file, we will import California educational information data, in a file that provides standardized test results from the SAT, ACT, and AP tests, which measure high school students' achievement:

http://www3.cde.ca.gov/researchfiles/satactap/sat19.xlsx

Data Preparation Example II, Part II

```
# perform web requests from a url
import requests
# https://xlrd.readthedocs.io/en/latest/ for Excel files
from xlrd import open_workbook
# download and save file locally
source_data_url_sat =
http://www3.cde.ca.gov/researchfiles/satactap/sat19.xlsx
sat_file = requests.get(source_data_url_sat)
with open('sat19.xlsx', 'wb') as f: f.write(sat_file.content)
```

Data Preparation Example II, Part III

- We can now open the workbook and get the labels of the headings row (assuming we know at least one of its values, “CDS”):

```
book = open_workbook('sat19.xlsx')
# first sheet
sheet = book.sheets()[0]
# index of the heading row
 first_row_idx = sheet.col_values(0).index('CDS')
```

Data Preparation Example II, Part IV

- We can now load the book as a dataframe:

```
sat19 = pd.DataFrame(datarows,
   columns = sheet.row_values(first_row_idx))
```

	CDS	CCode	CDCode	SCode	RType	SName	DName	CName	Enroll12	NumTSTTakr12	...
0	06615980630046	06	0661598	0630046	S	Colusa Alternative Home	Colusa Unified	Colusa	18	0.0	...
1	06616060634758	06	0661606	0634758	S	Maxwell Sr High	Maxwell Unified	Colusa	29	10.0	...
2	19647331930924	19	1964733	1930924	S	Belmont Senior High	Los Angeles Unified	Los Angeles	206	102.0	...
3	19647331931476	19	1964733	1931476	S	Canoga Park Senior High	Los Angeles Unified	Los Angeles	227	113.0	...
...	...	...	...	...	...	...	...	...	...	...	...

The End

Data Preparation Example III

Claudio Delrieux

Data Preparation Example III, Part I

Many official sources provide information in raw online **HTML** data. We will download such a file from the Texas Department of Criminal Justice (Texas Executed Offenders):

https://www.tdcj.texas.gov/death_row/dr_executed_offenders.html

Data Preparation Example III, Part II

The website looks something like this:

Texas Department of Criminal Justice

"...to provide public safety, promote positive change in offender behavior, reintegrate offenders into society, and assist victims of crime."

How may we help you?

- Home
- Information for Victims
- Career Opportunities
- Offender Information
- Find a Facility

HOME | DEATH ROW | Death Row Information

Death Row Information

Executed Offenders

Execution	Link	Link	Last Name	First Name	TDCJ Number	Age	Date	Race	County
570	Offender Information	Last Statement	Wardlow	Billy	999137	45	7/8/2020	White	Titus
569	Offender Information	Last Statement	Ochoa	Abel	999450	47	2/6/2020	Hispanic	Dallas
568	Offender Information	Last Statement	Gardner	John	999516	64	1/15/2020	White	Collin
567	Offender Information	Last Statement	Runnels	Travis	999505	46	12/11/2019	Black	Potter
566	Offender Information	Last Statement	Hall	Justen	999497	38	11/6/2019	White	El Paso
565	Offender Information	Last Statement	Sparks	Robert	999542	45	9/25/2019	Black	Dallas
564	Offender Information	Last Statement	Soliz	Mark	999571	37	9/10/2019	Hispanic	Johnson
563	Offender Information	Last Statement	Crutsinger	Billy	999459	64	9/4/2019	White	Tarrant

Data Preparation Example III, Part III

The HTML file can be retrieved and displayed:

```
source_data_url_teo = "http:// ... dr_executed_offenders.html"
teo_file = requests.get(source_data_url_teo)
teo_html = teo_file.text
from IPython.display import display_html
display_html(teo_html, raw=True)
```

Data Preparation Example III, Part IV

This will produce something like this:

Skip to Main Content
Employee Resources TDCJ Intranet Contact Us Información en Español

Texas Department of Criminal Justice

- Home
- Public Resources
- Employment
- About TDCJ
- Online Services
- Search

Executed Offenders

Execution	Link	Link	Last Name	First Name
531	Offender Information	Last Statement	Holiday	Raphael
530	Offender Information	Last Statement	Escamilla	Licho
529	Offender Information	Last Statement	Garcia	Juan
528	Offender Information	Last Statement	Lopez	Daniel
527	Offender Information	Last Statement	Russeau	Gregory
526	Offender Information	Last Statement	Bower	Lester
525	Offender Information	Last Statement	Charles	Derrick
524	Offender Information	Last Statement	Garza	Manuel
523	Offender Information	Last Statement	Sprouse	Kent

Data Preparation Example III, Part V

We can use the HTML parser BeautifulSoup. This allows many useful things, like navigating the HTML file and accessing the different fields in raw format:

```
executed_doc = BeautifulSoup(teo_html, 'html.parser')
executed_rows = executed_doc.select('table.os tr')[1:]
```

Data Preparation Example III, Part VI

This will produce something like this:

```
[<tr>
 <td>531</td>
 <td><a href="dr_info/holidayraphael.html" title="Offender Information for
Raphael Holiday">Offender Information</a></td>
 <td><a href="dr_info/holidayraphaellast.html" title="Last Statement of
Raphael Holiday">Last Statement</a></td>
 <td>Holiday</td>
 <td>Raphael</td>
 <td>999419</td>
 <td>36</td>
 <td>11/18/2015</td>
 <td>Black</td>
 ...
```

Data Preparation Example III, Part VII

Another option is to use Pandas dataframes:

```
teo_df = pd.read_html(teo_html)[0]
```

	Execution	Link	Link.1	Last Name	First Name	TDCJ Number	Age	Date	Race	County
0	531	Offender Information	Last Statement	Holiday	Raphael	999419	36	11/18/2015	Black	Madison
1	530	Offender Information	Last Statement	Escamilla	Licho	999432	33	10/14/2015	Hispanic	Dallas
2	529	Offender Information	Last Statement	Garcia	Juan	999360	35	10/6/2015	Hispanic	Harris
3	528	Offender Information	Last Statement	Lopez	Daniel	999555	27	08/12/2015	Hispanic	Nueces
...	...	...	...	...	...	...	...	...	...	...
527	4	Offender Information	Last Statement	Barefoot	Thomas	621	39	10/30/1984	White	Bell
528	3	Offender Information	Last Statement	O'Bryan	Ronald	529	39	03/31/1984	White	Harris
529	2	Offender Information	Last Statement	Autry	James	670	29	03/14/1984	White	Jefferson
530	1	Offender Information	Last Statement	Brooks, Jr.	Charlie	592	40	12/07/1982	Black	Tarrant

531 rows × 10 columns

The End

Data Preparation Example IV

Claudio Delrieux

Data Preparation Example IV, Part I

- Finally, for Java Script Object Notation (JSON), Pandas also has a specific method
- We will explore the file https://www.kaggle.com/eswarchandt/amazon-music-reviews/download, which is a zipped JSON containing reviews of Amazon music instruments

Data Preparation Example IV, Part II

- The raw file looks something like this:


```
{"reviewerID": "A14VAT5EAX3D9S", "asin": "1384719342", "reviewerName": "Jake",
"helpful": [13, 14], "reviewText": "The product does … etc. … Buy this product!
:]", "overall": 5.0, "summary": "Jake", "unixReviewTime": 1363392000, "reviewTime":
"03 16, 2013"}
{"reviewerID": "A195EZSQDW3E21", "asin": "1384719342", "reviewerName": "Rick
Bennette \"Rick Bennette\"", "helpful": [1, 1], "reviewText": "The primary job of
this device is to … etc. … where you put it.", "overall": 5.0, "summary": "It Does
The Job Well", "unixReviewTime": 1377648000, "reviewTime": "08 28, 2013"}
...
{ ... }
```

Data Preparation Example IV, Part III

```
source_data_url_ami="https:// ... /Musical_Instruments_5.json"
# Notice lines=True means the json file contains multiple objects
data = pd.read_json(source_data_url_ami, lines=True)
```

	reviewerID	asin	reviewerName	helpful	reviewText	overall
0	A2IBPI20UZIR0U	1384719342	cassandra tu "Yeah, well, that's just like, u...	[0, 0]	Not much to write about here, but it does exac...	5
1	A14VAT5EAX3D9S	1384719342	Jake	[13, 14]	The product does exactly as it should and is q...	5
2	A195EZSQDW3E21	1384719342	Rick Bennette "Rick Bennette"	[1, 1]	The primary job of this device is to block the...	5
3	A2C00NNG1ZQQG2	1384719342	RustyBill "Sunday Rocker"	[0, 0]	Nice windscreen protects my MXL mic and preven...	5
4	A94QU4C90B1AX	1384719342	SEAN MASLANKA	[0, 0]	This pop filter is great. It looks and perform...	5
...	...	...	...	...	...	...
10256	A14B2YH83ZXMPP	B00JBIVXGC	Lonnie M. Adams	[0, 0]	Great, just as expected. Thank to all.	5
10257	A1RPTVW5VEOSI	B00JBIVXGC	Michael J. Edelman	[0, 0]	I've been thinking about trying the Nanoweb st...	5
10258	AWCJ12KBO5VII	B00JBIVXGC	Michael L. Knapp	[0, 0]	I have tried coated strings in the past (incl...	4
10259	A2Z7S8B5U4PAKJ	B00JBIVXGC	Rick Langdon "Scriptor"	[0, 0]	Well, MADE by Elixir and DEVELOPED with Taylor...	4
10260	A2WA8TDCTGUADI	B00JBIVXGC	TheTerrorBeyond	[0, 0]	These strings are really quite good, but I wou...	4

10261 rows × 9 columns

The End

Data Analysis Types I

Claudio Delrieux

Data Analysis Types I, Part I

- Among the main analysis types, we can mention:
  - Classification
  - Clustering
  - Regression
  - Dimension reduction
  - Text analysis
  - Outlier detection

Data Analysis Types I, Part II

- We will focus first in the classification models, of which we can mention the following:
  - Naïve Bayes
  - Logistic regression
  - K-nearest neighbors
  - Support vector machines
  - Decision trees
  - Random forests
  - Stochastic gradient descent

Data Analysis Types I, Part III

- **Naïve Bayes** is perhaps the simplest and fastest: used as a baseline; best suited for nominal features; not very accurate
- **Logistic regression** uses a logistic function to regress a binary class probability; helps understanding the influence of each feature; sensitive to outliers

Data Analysis Types I, Part IV

- **K-nearest neighbors** does not require training, classifies via simple majority vote: robust to outliers, very dependent on N
- **Support vector machines** represent the training set in a higher dimension space in which classes are separable: almost the most accurate

Data Analysis Types I, Part V

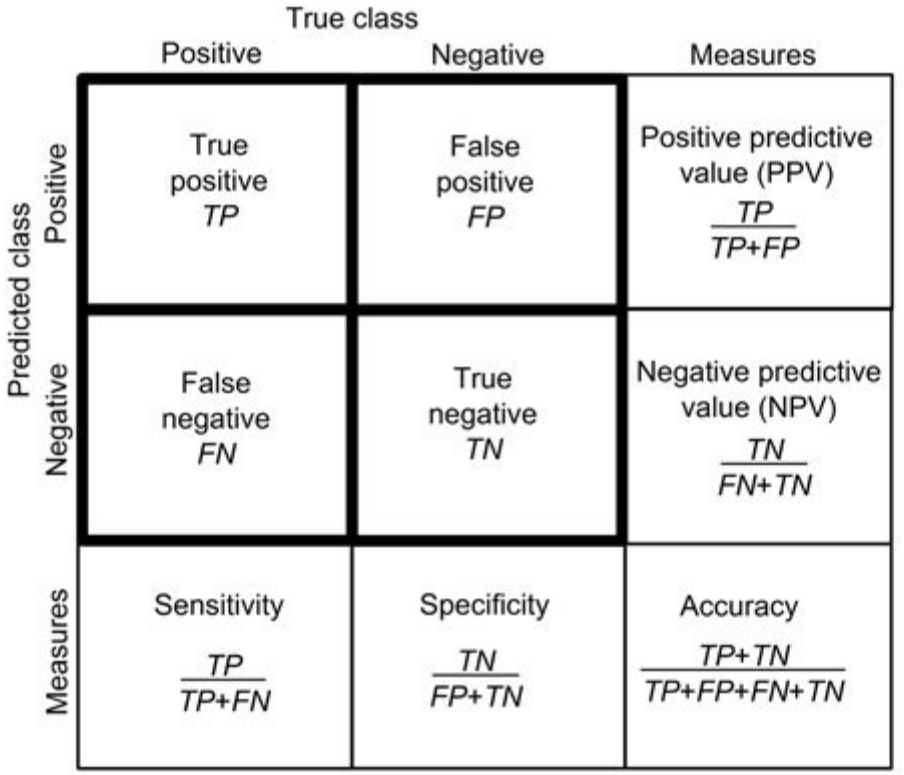
- **Decision trees** are induced by recursively splitting the dataset according to the highest information gain: the most transparent and understandable model; training can be unstable and requires much intervention to avoid overfitting

Data Analysis Types I, Part VI

- **Random forests** are collections of randomly built decision trees and classifies by majority: almost the most accurate, but complex
- **Stochastic gradient descent** uses a simple rule to fit linear models: easy to implement and execute, and flexible to use different loss functions

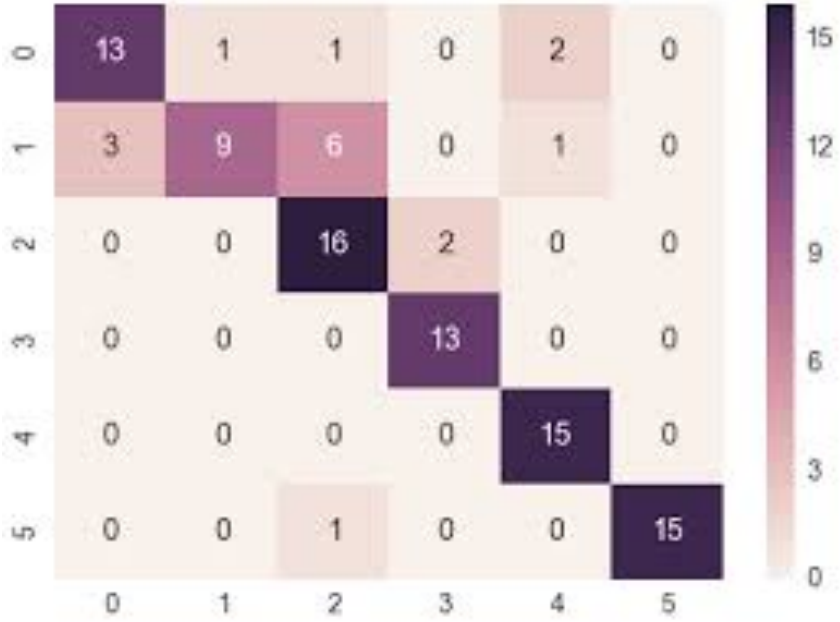
Data Analysis Types I, Part VII

- Regarding classifier model evaluation, the starting point is to analyze the contingency table (also “confusion matrix”)



Data Analysis Types I, Part VIII

- If the classification is n-ary, the matrix can be extended to more rows and columns, and the evaluation parameters (sensitivity, TPR, etc.) are extended naturally



Data Analysis Types I, Part IX

- In many contexts, the expected amount of TN is much higher than the rest of outcomes, thus distorting the meaning of the quality parameters; for this, F-measure or IoU are suggested
- For parametric classifiers (in which, for instance, there is a classification threshold), the ROC curve shows the trade-off between sensitivity and specificity

The End

Data Analysis Types II

Claudio Delrieux

Data Analysis Types II, Part I

- Clustering is much more complex than classification, since the very concept is difficult to formalize. There are several models.
  - Partitioning methods
  - Hierarchical clustering
  - Fuzzy clustering
  - Density-based clustering
  - Model-based clustering

Data Analysis Types II, Part II

- **Partitioning algorithms** subdivide the data sets into a set of K groups (K-means). An alternative is the K-medoids clustering or PAM (partitioning around medoids), which is less sensitive to outliers.

Data Analysis Types II, Part III

- **Hierarchical clustering** does not require to pre-specify the number of clusters to be generated. It results in a tree-based representation (dendrogram). Observations can be subdivided into groups by cutting the dendrogram at a desired similarity level.

Data Analysis Types II, Part IV

- **Fuzzy clustering** allows items to be members of more than one cluster. Each item has a set of membership coefficients corresponding to the degree of being in a given cluster (fuzzy c-means).

Data Analysis Types II, Part V

- **Density-based clustering** is based on a notion of “clusters” and “noise.” The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.
- Clusters are dense regions in the data space, separated by regions of lower density of points.

Data Analysis Types II, Part VI

- In **model-based clustering**, data is considered as coming from a mixture of densities. Each cluster component can be modeled by statistical parameters (mean and covariance), and an associated probability to pertain to the cluster.

Data Analysis Types II, Part VII

- Clustering model evaluation can be broadly divided into *extrinsic* and *intrinsic*.
- In extrinsic evaluation, there is actual information about data pertaining to the same clusters, and therefore, a consensus measure can be established (for example, Cohen's kappa).

Data Analysis Types II, Part VIII

- Intrinsic evaluation regards clusters' homogeneity (as the mean distance of every item to the cluster centroid).
- The Davies-Bouldin index adds to that a relative distance between clusters.
- The silhouette coefficient can be applied to every data item to see how well the item was assigned to a given cluster.

The End

Data Analysis Types III

Claudio Delrieux

Data Analysis Types III, Part I

Regression models are also among the most important data analysis type. We can use the following regression types.

- Linear, nonlinear
- Lasso, Ridge, Elastic Net
- Decision tree, random forest
- SVR, kernel SVR
- KNN regressor
- LARS
- RANSAC (for dealing with outliers)

Data Analysis Types III, Part II

Linear regression is the simplest: assumes the dependent variable **y** and independent variables $\mathbf{x_i}$ are continuous and assumed to be linearly related. Also, assumptions are independent of the samples—no outliers, no heteroskedasticity, normal distribution of the error, no autocorrelation among variables.

$$y = \beta_1 + \beta_2 X_2 + \beta_3 X_3 + \ldots + \beta_k X_k + \varepsilon$$

Data Analysis Types III, Part III

This model can be easily adapted to polynomial (or other nonlinear) regression models by means of a simple variable replacement.

$$y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots + \beta_k X^k + \varepsilon$$

Data Analysis Types III, Part IV

If any of the assumptions is violated (heteroscedasticity is present in the data, there are outliers, data is skewed, etc.), then it is better to find a nonparametric linear regression, for instance, quantile-quantile fit.

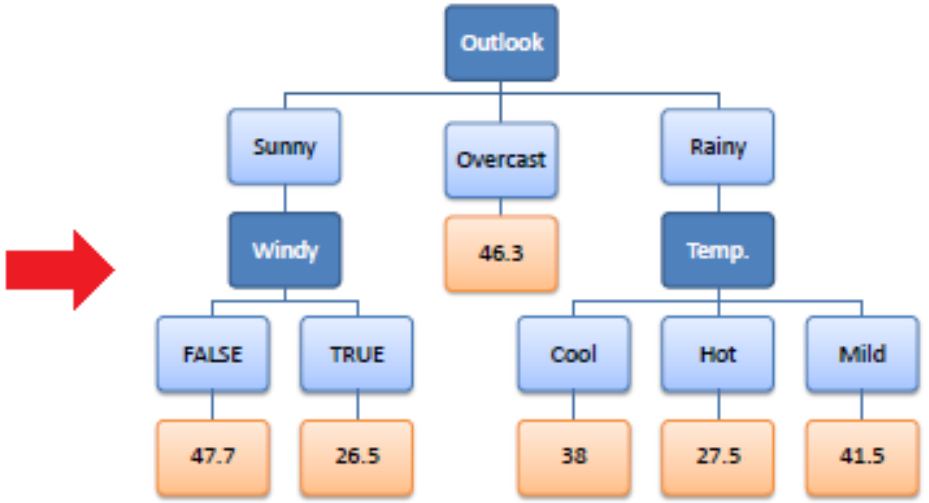
Data Analysis Types III, Part V

Regularization helps solve overfitting by adding a penalty term to the objective function that controls the model complexity. We can have L1 (accumulates the absolute values of the fitting coefficients) or L2 (accumulates the squared values) regularization models (i.e., penalties). The first is the **LASSO** model, the second is the **ridge regression**, and the two combined is the **elastic-net model.**

Data Analysis Types III, Part VI

Decision trees and **random forests** can also be trained to predict quantitative values. Instead of using information gain for splitting, minimize the SD of the target value.

Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	26
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



Data Analysis Types III, Part VII

SVMs can also be used as a regression method, maintaining all the main features that characterize the algorithm. A tolerance margin is set that minimizes the absolute error, individualizing a hyperplane which maximizes the margin.

Data Analysis Types III, Part VIII

K-NN classifier can also be easily extended for regression. In this case, instead of predicting the label using the majority rule, simply average (perhaps with weighing) the values of the K training cases that are closest to the case to be predicted.

Data Analysis Types III, Part IX

Regression model evaluation is in general numeric in nature, including error metrics (the standard error of the estimate and the RMSE), predictive power (determination coefficient), and loss functions (Poisson deviance loss).

The End

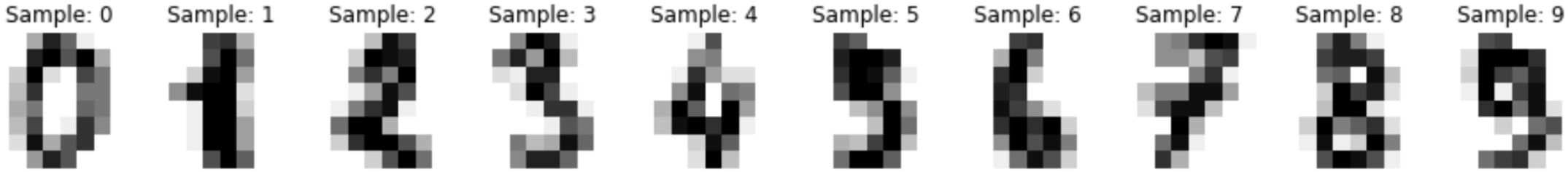
Data Analysis Examples I

Claudio Delrieux

Data Analysis Examples I, Part I

- As a classification worked example, we will be using a classic dataset in image processing. It contains 8x8 images of digits distributed in 10 classes. The dataset contains about 180 samples per class.

```
from sklearn import datasets
digits = datasets.load_digits()
```



Data Analysis Examples I, Part II

- First, in order to train a classifier on this image dataset, we need to flatten it into an *(samples,features)* array.

```
data = digits.images.reshape((n_samples, -1))
```

- Second, we need to split the dataset into *train/test* subsets.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,
                    digits.target, test_size=0.2, random_state=1)
```

Data Analysis Examples I, Part III

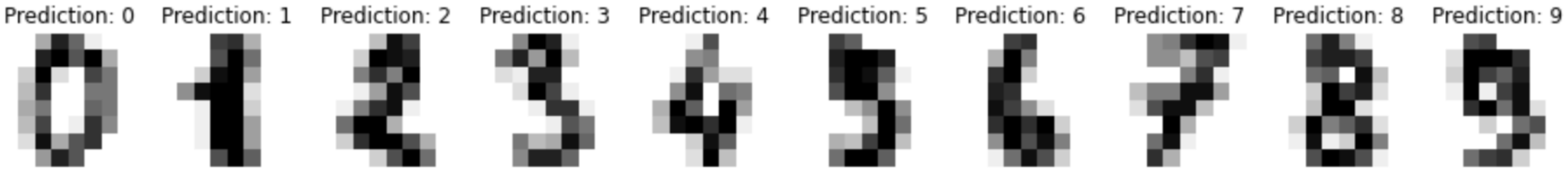
- We are finally ready to train our classifier. We are using C-Support Vector Classification implemented in Scikit-Learn, which offers several implementations.

```
classifier = svm.SVC(gamma=0.001)
classifier.fit(X_train, y_train)
```

- And that's it!

Data Analysis Examples I, Part IV

- Now, we need to evaluate the performance of our classifier using the testing subset. Predicting and plotting predictions along with testing images allows us to qualitatively address the classifier accuracy.

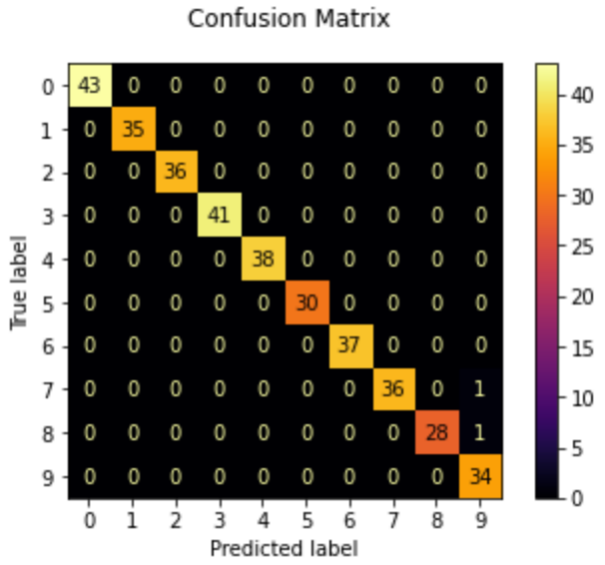


Data Analysis Examples I, Part V

- We can assess the overall quality metrics and plot the confusion matrix.

```
disp = metrics.plot_confusion_matrix(classifier, X_test, y_test, cmap='inferno')
disp.figure_.suptitle("Confusion Matrix")
print("Confusion matrix:\n%s" % disp.confusion_matrix)
```

```
Confusion matrix:
[[43  0  0  0  0  0  0  0  0  0]
 [ 0 35  0  0  0  0  0  0  0  0]
 [ 0  0 36  0  0  0  0  0  0  0]
 [ 0  0  0 41  0  0  0  0  0  0]
 [ 0  0  0  0 38  0  0  0  0  0]
 [ 0  0  0  0  0 30  0  0  0  0]
 [ 0  0  0  0  0  0 37  0  0  0]
 [ 0  0  0  0  0  0  0 36  0  1]
 [ 0  0  0  0  0  0  0  0 28  1]
 [ 0  0  0  0  0  0  0  0  0 34]]
```



The End

Data Analysis Examples II

Claudio Delrieux

Data Analysis Examples II, Part I

- As a clustering example, we will clusterize pixel RGB values in order to get a multi-label segmentation of the following image.

Data Analysis Examples II, Part II

- First, we load the image into an array.

```
from PIL import Image
import numpy as np
import requests
from io import BytesIO

response = requests.get('https://…/butterfly.jpg')
image = Image.open(BytesIO(response.content))
image = np.array(image)
original_shape = image.shape
# (height, weight, 3) 3 refers to the RGB color planes
```

Data Analysis Examples II, Part III

- To perform a simple pixel clustering, we first need to eliminate width and height dimensions, therefore *flattening* the array.

```
X = np.reshape(image, [-1, 3])
print(X.shape)
(461760, 3)
# we have 460K points in 3D space
```

Data Analysis Examples II, Part IV

- We can use MeanShift() to clusterize the pixels array. It is an agglomerative centroid-based algorithm.

```
from sklearn.cluster import MeanShift, estimate_bandwidth


# Run MeanShift
bandwidth = estimate_bandwidth(X, quantile=0.1, n_samples=100)
ms = MeanShift(bandwidth=bandwidth,bin_seeding=True, n_jobs=4)
ms.fit(X)
```

Data Analysis Examples II, Part V

- This yields the following results.

```
MeanShift(bandwidth=50.548…, bin_seeding=True,
  cluster_all=True,max_iter=300, min_bin_freq=1, n_jobs=4, seeds=None)
```

- Which means that with the initial parameters, four color clusters were found. Different parameter settings may produce different amount of clusters.

Data Analysis Examples II, Part VI

- To get a clearer sense of what was done, we may output the results.

```
labels = ms.labels_

cluster_centers = ms.cluster_centers_
print("Cluster centers:\n{}".format(cluster_centers))

labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)
print("Number of estimated clusters :{}".format(n_clusters_))
```

Data Analysis Examples II, Part VII

- This produces this result.

```
Cluster centers:
[[ 82.77994529 116.03817873 59.46351522]
 [195.77418627 185.08153131 208.68783953]
 [239.57766667 181.53952778 76.63338889]
 [187.06755614 104.72885266 53.70528299]]
Number of estimated clusters :4
```

- The values are not exactly RGB colors; they have to be quantized into bytes.

Data Analysis Examples II, Part VIII

```
import seaborn as sns
color_clusters = np.floor(cluster_centers).astype(int)/255
sns.palplot(sns.color_palette(color_clusters))
```

- Finally, we assemble a new image that uses only these four colors.

Data Analysis Examples II, Part IX

```
segmented_image = np.reshape(labels, original_shape[:2])
image_color_clusters = np.array(image, copy=True)

for i in range(0,n_clusters_):
   mask = (segmented_image == i)
   image_color_clusters[mask] = color_clusters[i]*255
```

The End

Data Analysis Examples III

Claudio Delrieux

Data Analysis Examples III, Part I

- As a worked-out regression example, we will explore a dataset that contains the amount spent on advertising for a single product in a given medium (TV, radio, and newspaper) to see if any relationship can be found against sales

```
example_path = "https:// ... /advertising.csv"
```

Data Analysis Examples III, Part II

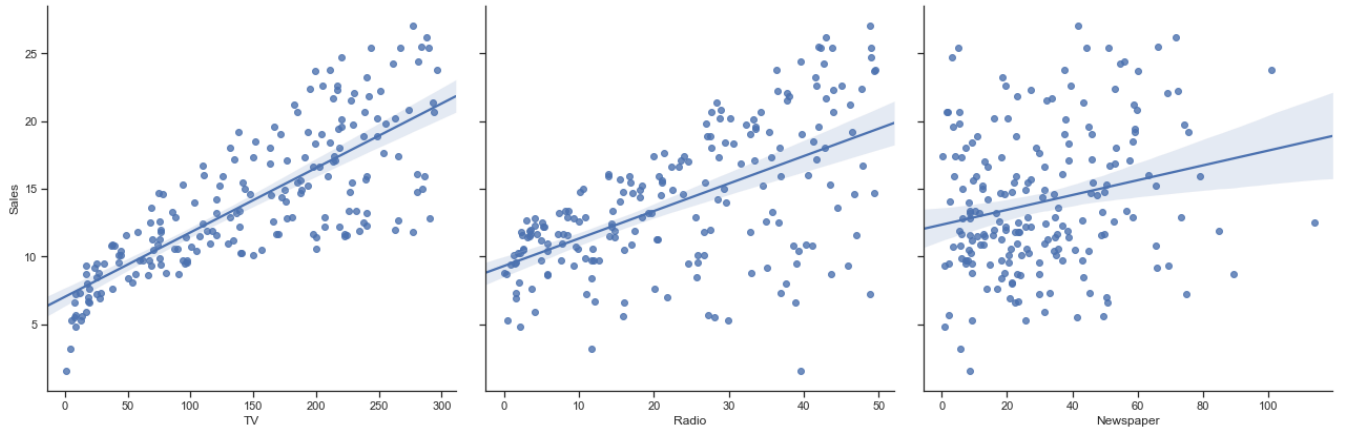
```
import pandas as pd
advertising = pd.read_csv(example_path).drop(['Unnamed: 0'], axis=1)
```

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6

Data Analysis Examples III, Part III

- We can visualize a scatterplot lattice with all the independent variables

```
import seaborn as sns
sns.set(style="ticks", color_codes=True)
sns.pairplot(advertising, x_vars=['TV','Radio','Newspaper'],
                    y_vars='Sales', height=6, aspect=1., kind='reg')
```



Data Analysis Examples III, Part IV

- To train our multinomial linear regressor, first split the data into train and test

```
from sklearn.model_selection import train_test_split
X = advertising[['TV', 'Radio', 'Newspaper']]
y = advertising['Sales']
X_train, X_test, y_train, y_test = train_test_split(X, y,
  random_state=1)
# Default split is 75% for training and 25% for testing
```

Data Analysis Examples III, Part V

- We are now ready to go

```
import numpy as np
from sklearn.linear_model import LinearRegression
linear_regressor = LinearRegression()
fit = linear_regressor.fit(X_train, y_train)
```

- To assess the model quality we examine the R^2

```
fit.score(X_train, y_train)
0.8903…
```

Data Analysis Examples III, Part VI

- To explore the model parameters, we use:

```
print(fit.intercept_)
print(fit.coef_)
```

```
2.876966623179335
[0.04656457 0.17915812 0.00345046]
```

- Which means that the model looks like:
  - Sales=2.88+0.0466*TV*+0.179*Radio*+0.00345*Newspaper*+ϵ

Data Analysis Examples III, Part VII

- We can now evaluate the error metrics against the test set

```
from sklearn import metrics
y_pred = fit.predict(X_test)
print("RMSE =",np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
RMSE = 1.404651423032894 8
```

The End

Data Analysis Examples IV

Claudio Delrieux

Data Analysis Examples IV, Part I

- As a final worked example, we will focus on some techniques used in outlier detection
- First, we create a synthetic 1,000 by four dataset with normally distributed features

```
np.random.seed(1)
features_number = 4
dummy_df = pd.DataFrame(np.random.normal(scale=10.0, size=(1000,
features_number)), columns=['feature{}'.format(i)
   for i in range(features_number)])
```

Data Analysis Examples IV, Part II

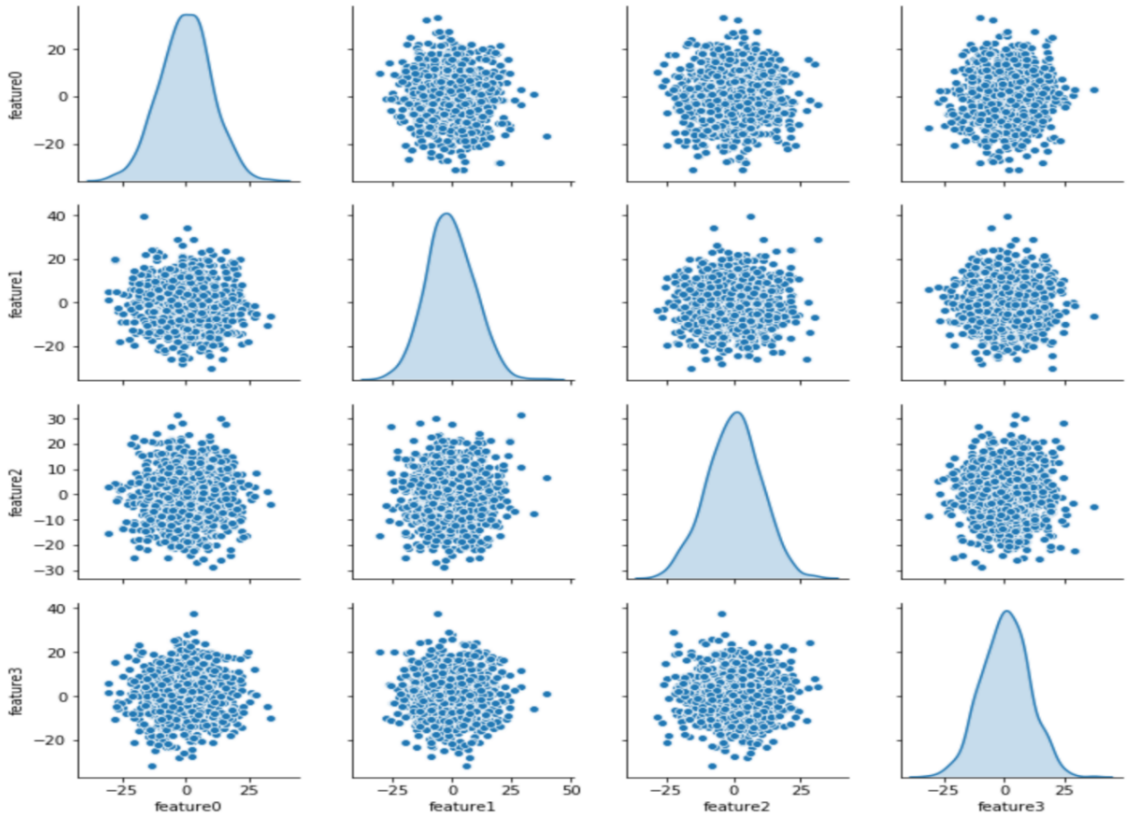
- This will generate a dataset that will look like this:

	feature0	feature1	feature2	feature3
0	16.243454	–6.117564	–5.281718	–10.729686
1	8.654076	–23.015387	17.448118	–7.612069
2	3.190391	–2.493704	14.621079	–20.601407
3	–3.224172	–3.840544	11.337694	–10.998913
4	–1.724282	–8.778584	0.422137	5.828152
...	...	...	...	...
999	-9.603464	-8.439133	6.283417	5.372145

Data Analysis Examples IV, Part III

- A scatterplot lattice shows data normality

```
sns.pairplot(dummy_df, diag_kind="kde")
```



Data Analysis Examples IV, Part IV

- One of the main univariate parametric outlier detection techniques is to test for high absolute z-value (i.e., values farther than *n* standard deviations from the mean, *n,* usually over 3)

```
feature = 'feature0'
data_mean, data_std = dummy_df[feature].mean(), dummy_df[feature].std()
print("feature0 mean:{} std:{}".format(data_mean, data_std))
```

```
feature0 mean:0.130295… std:10.144231…
```

Data Analysis Examples IV, Part V

- Outliers are over three std farther from the mean

```
nstd = 3.0
dist = data_std * nstd
lower_bound, upper_bound = data_mean - dist, data_mean + dist
outliers_mask = [True if x < lower_bound or x > upper_bound else False
                 for x in dummy_df[feature]]
```

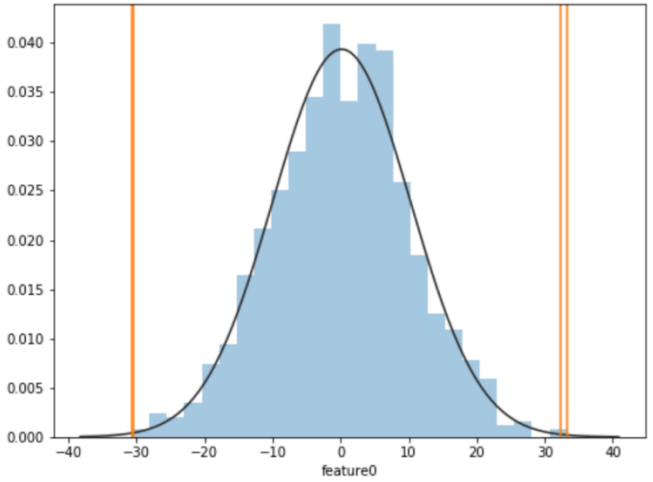
- Outliers:

```
183 33.210788
223 -30.537644
567 -30.641414
762 32.383432
```

Data Analysis Examples IV, Part VI

- We show the outliers against a histogram and fit to normal

```
from scipy.stats import norm
plt.figure(figsize=(8,6))
sns.distplot(dummy_df[feature], fit=norm ,kde=False);
for outlier in dummy_df[feature][outliers_mask]:
  plt.axvline(outlier, 0, dummy_df[feature].max(), color = '#FF7F0E')
```



Data Analysis Examples IV, Part VII

- Another technique for (nonparametric) univariate outlier detection is to take the interquartile range (between Q25 and Q75), and mark as outliers data outside $k*IR$ units away from these quartiles, for a given k (usually 1.5)

Data Analysis Examples IV, Part VIII

```
q25 = np.percentile(dummy_df[feature], 25)
q75 = np.percentile(dummy_df[feature], 75)
iqr = q75 - q25
k = 1.5
range_ir = iq5*k
lower_bound_ir, upper_bound_ir = q25 - range_ir, q75 + range_ir
outliers_mask_ir = [True if x < lower_bound_ir or x > upper_bound_ir
                    else False for x in dummy_df[feature]]
```

Data Analysis Examples IV, Part IX

- Outliers are now:

```
 63    -27.930850
 83     33.210788
223    -30.537644
244    -26.641259
427    -27.914440
567    -30.641414
684    -27.294621
762     32.383432
771    -26.768414
783    -27.391417
969     27.061249
976     27.421552
```

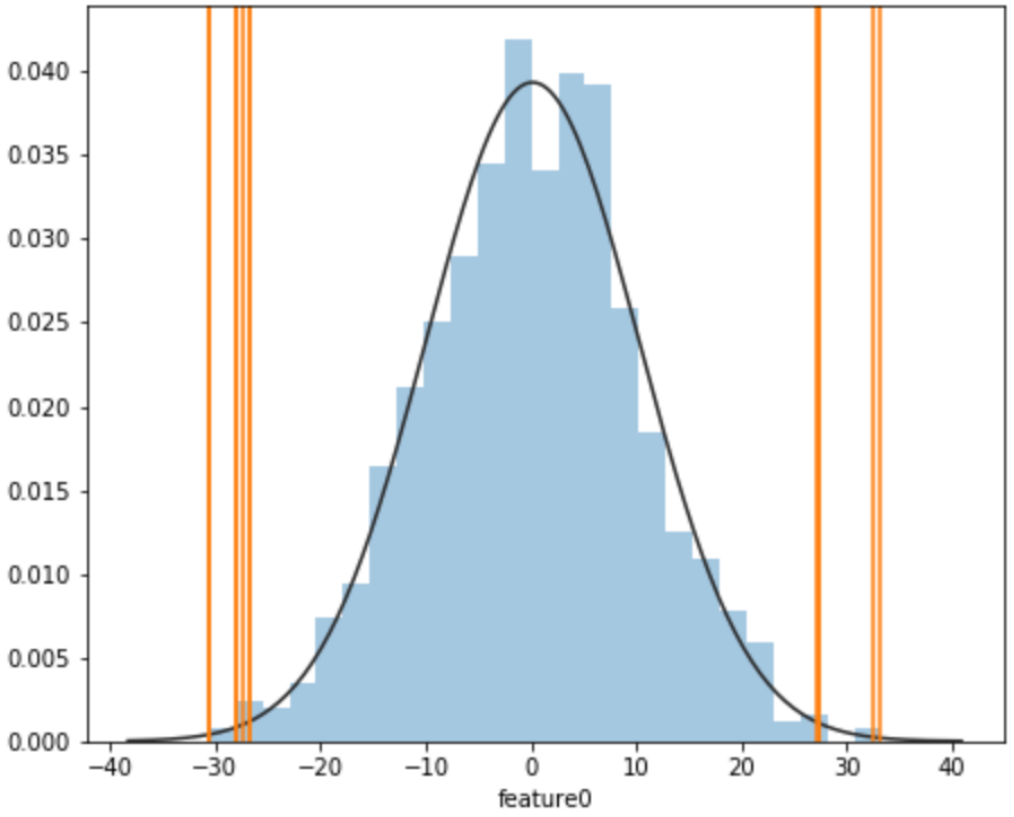
Data Analysis Examples IV, Part X

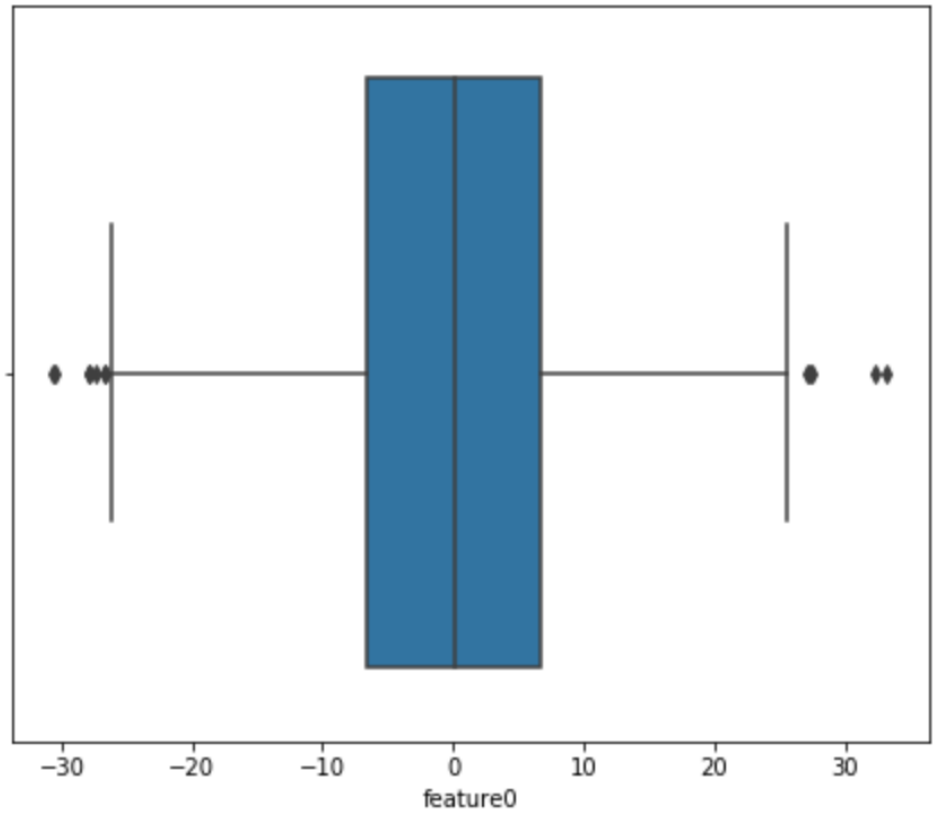
- We can plot the new outliers:

```
plt.figure(figsize=(16,6))
ax1=plt.subplot(1, 2, 1)
sns.distplot(dummy_df[feature], fit=norm ,kde=False, ax = ax1);
for outlier in dummy_df[feature][outliers_mask_ir]:
   plt.axvline(outlier, 0, dummy_df[feature].max(), color = '#FF7F0E')
ax2=plt.subplot(1, 2, 2)
sns.boxplot(x=dummy_df[feature], whis=k, ax = ax2)
```

Data Analysis Examples IV, Part XI

- In the boxplot, the box represents the IQR and the whiskers the range





The End

Data Visualization

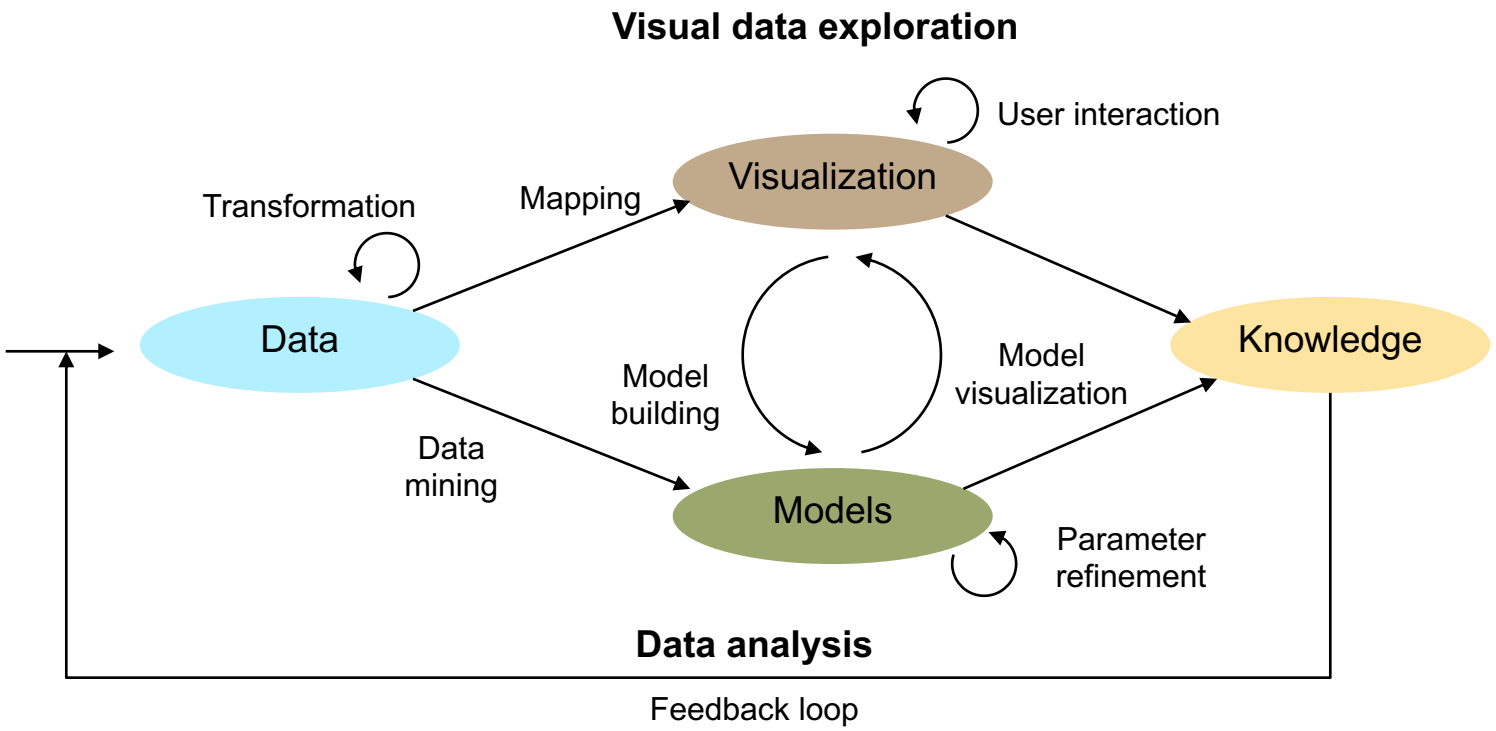
Claudio Delrieux

Data Visualization, Part I

- Visualization takes advantage of the superior capabilities of the human brain to gather, filter, process, and understand huge volumes of information in a fast and reliable manner.
- Data visualization leverages this in the data science life cycle, providing means to convey the right results in the right way to the right team members.

Data Visualization, Part II

- This chart helps understand some of the underlying processes and entities involved in plain data analysis and in visualization.



Data Visualization, Part III

- In plain data analysis, we go from data to the model (the data mining edge), and from the model to understanding (knowledge), which in turn can lead us back to data gathering through a feedback process.
- We can have data transformation (edge data to data) and model parameter refinement (edge model to model).

Data Visualization, Part IV

- When visualization is introduced, we have several other paths. The most important is the **“mapping”** edge, in which the dataset is converted into some visual object that makes sense to the observer through visualization.
- This mapping can be performed in several ways, depending much on the dataset, the nature of the analysis task, and the intended audience.

Data Visualization, Part V

- In addition, we have other new edges. The **“user interaction”** edge represents the capability of our data visualization to be actionable and therefore to allow for interactive explorations. This is a fundamental feature when the data analysis task is complex and open-ended, requiring support for analytical reasoning in decision-making. This called also **visual analytics.**

Data Visualization, Part VI

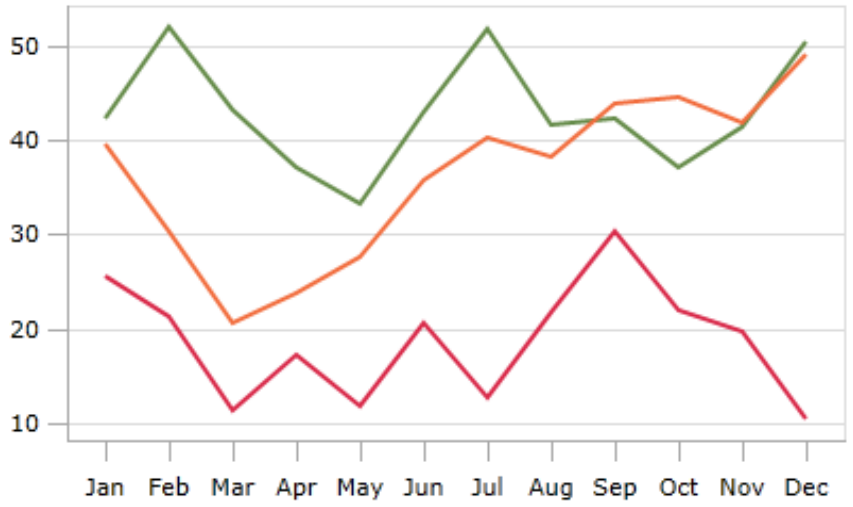
- The **“model building”** edge also arises when, after a raw but adequate visualization of the dataset, a good model gets apparent immediately. For instance, a plot of a time series may show quasi-periodic patterns that in the raw data are impossible to detect.

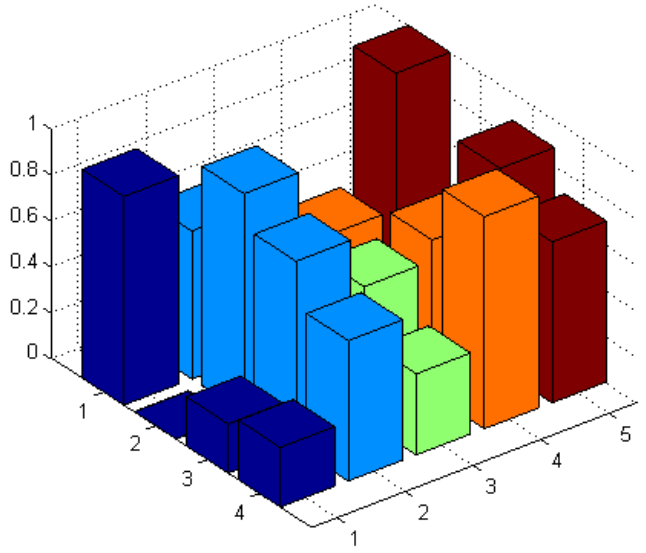
Data Visualization, Part VII

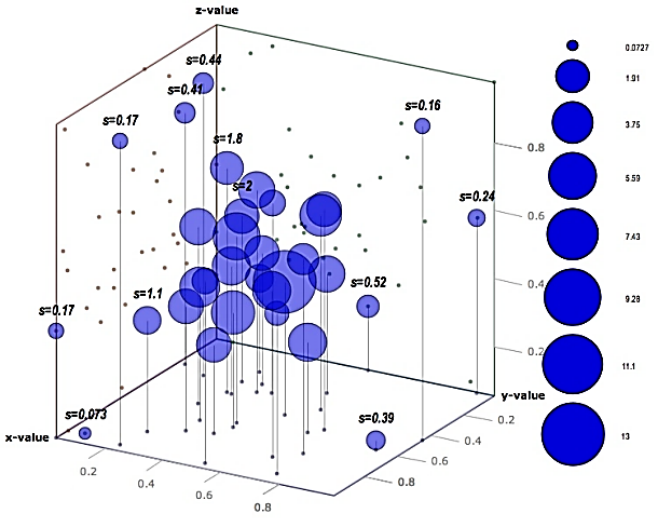
- The **“model visualization”** edge is also useful to shed light on aspects of the data model itself when it is complex.
- Interestingly, this may trigger another feedback loop between model visualization and model building.

Data Visualization, Part VIII

- The central concept in data visualization is the “view,” in which the dataset is mapped into a visual object. Below, we have three different views (line graph, bar graph, and scatterplot) for three different datasets.





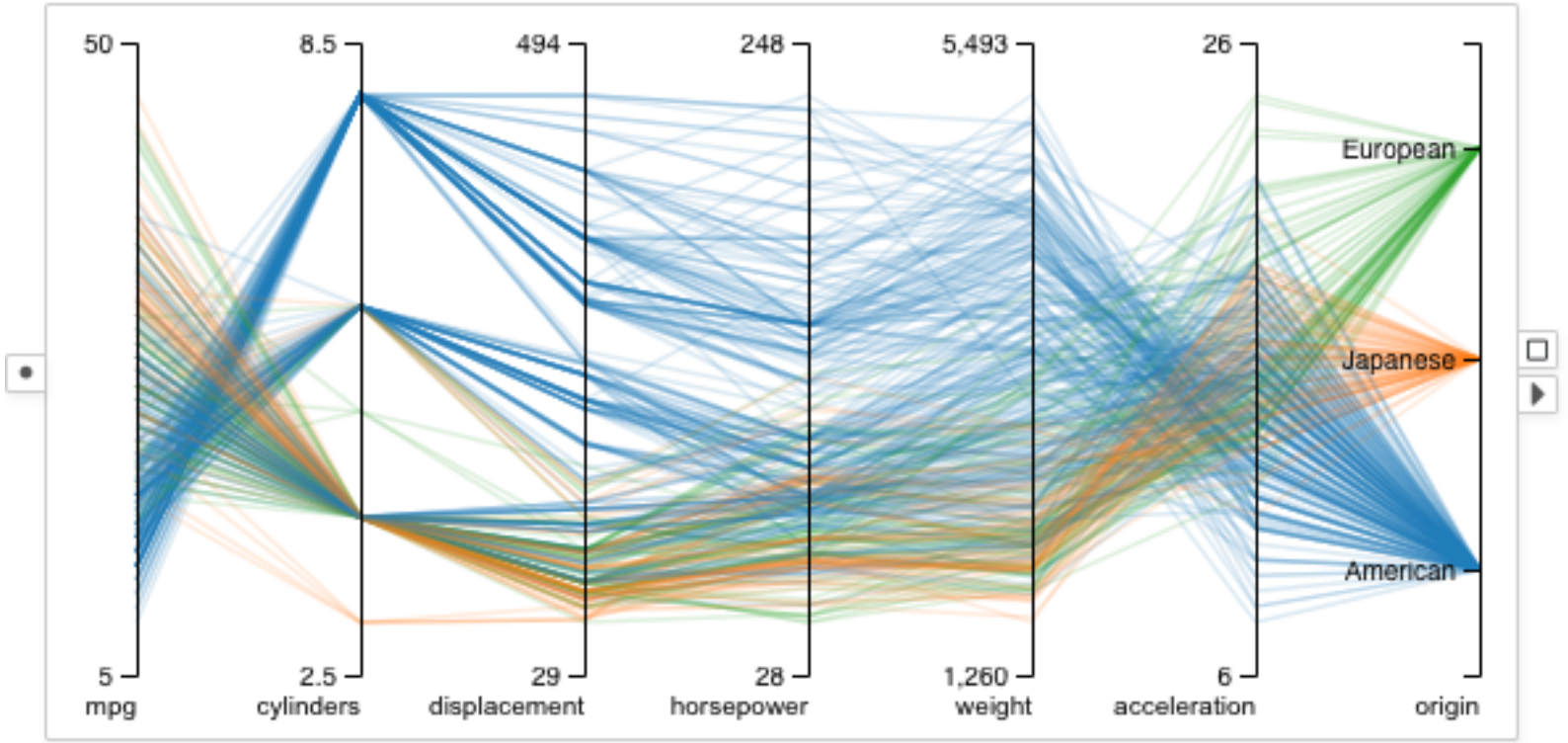


Data Visualization, Part IX

- These views are mostly intended for tabular data in which the independent variable(s) can be nominal (categorical), ordinal, or quantitative, and the dependent variable(s) are quantitative.

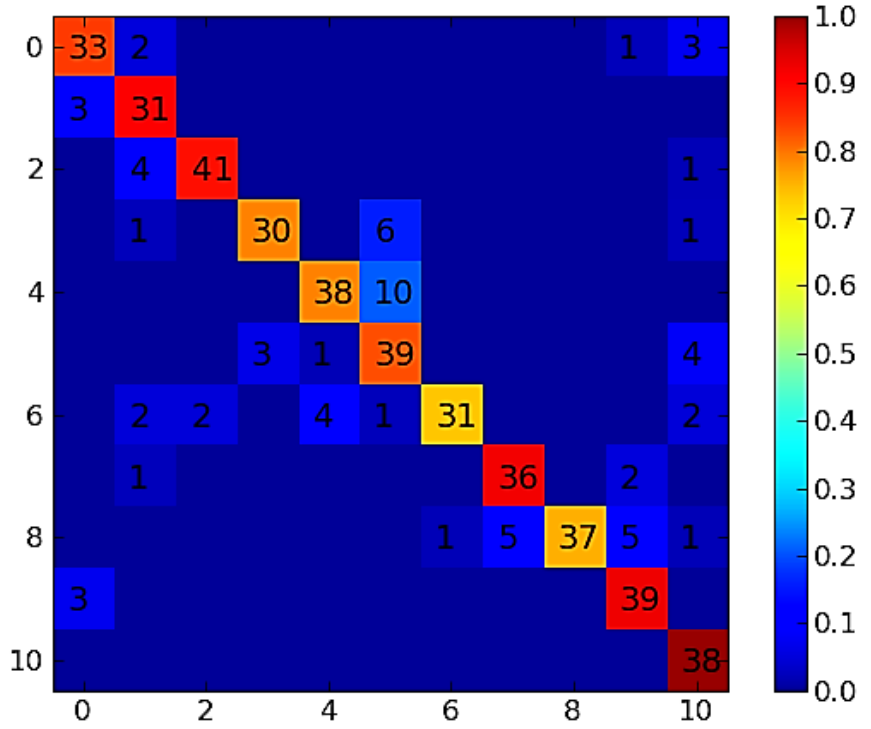
Data Visualization, Part X

- If the dimensionality of the dataset is too high, then parallel coordinates may be used.



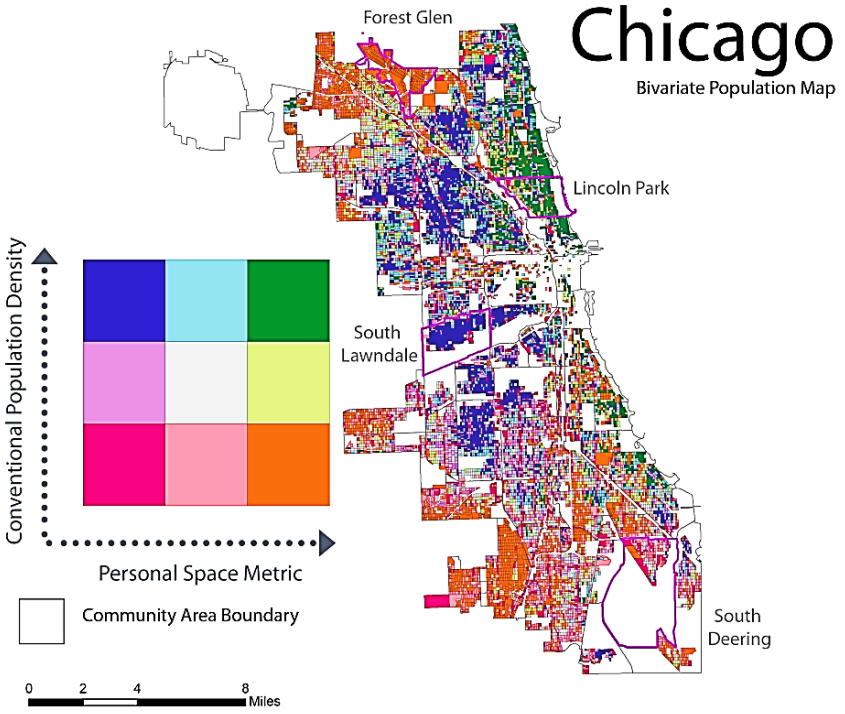
Data Visualization, Part XI

- Sometimes, instead of a bar chart, a **color map** or heat map can represent values with less visual clutter. Judge carefully what color chart (“palette”) is adequate in each case.



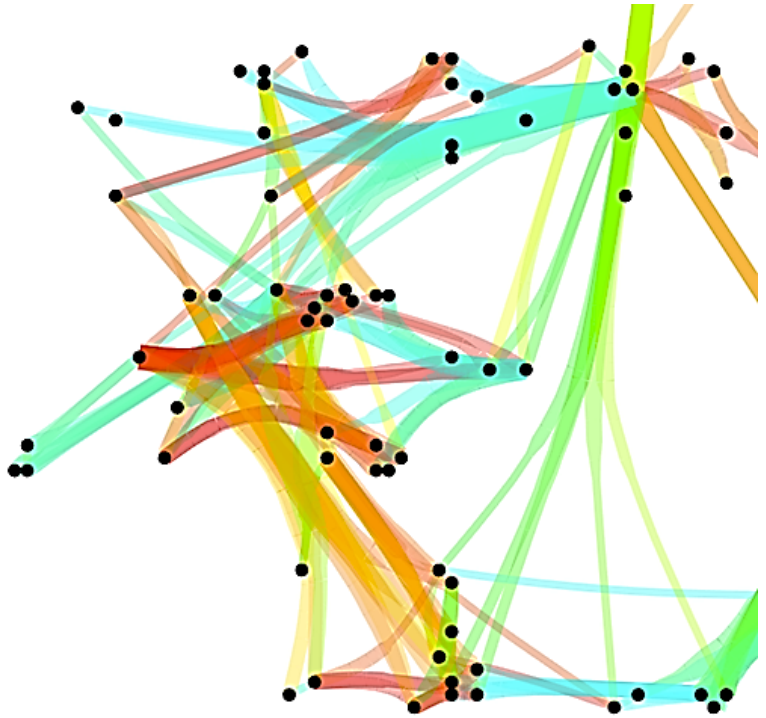
Data Visualization, Part XII

- Geolocated visualization is a topic in itself, requiring specific libraries to import and manage maps and location information.



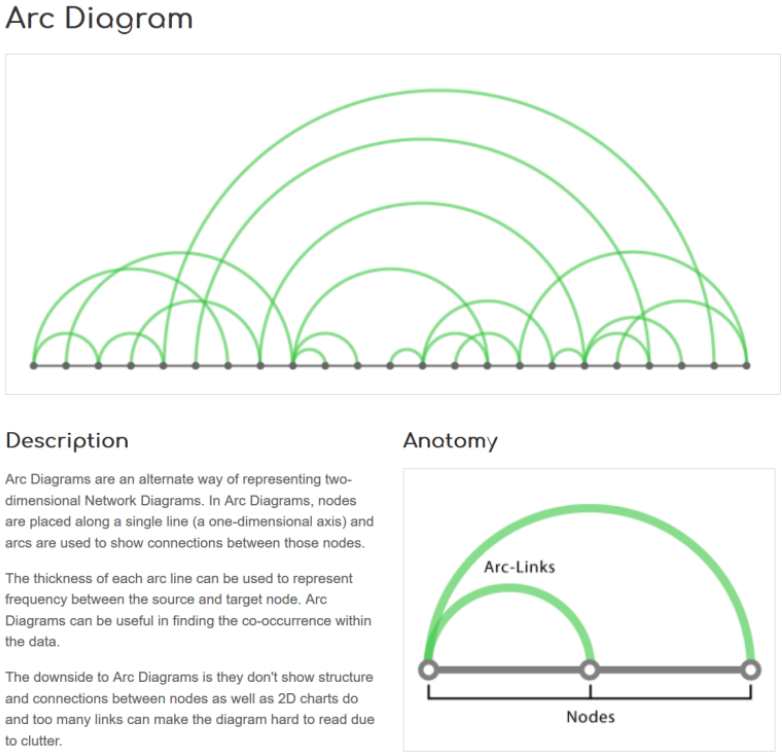
Data Visualization, Part XIII

- Network visualization is also relevant in many data science projects, and clever visualizations can be very helpful. Also, there are several good libraries for graph plotting.



Data Visualization, Part XIV

- In https://datavizcatalogue.com, for instance you have quite a complete catalog of views, each with a description, purpose, functioning, etc.



The End

Data Visualization Tools

Claudio Delrieux

Data Visualization Tools, Part I

- Python offers multiple visualization libraries packed with lots of cool features. These allow you to create highly customized plots and even add some actionability.
- Among the most popular we can mention are MatPlotLib, Seaborn, ggplot, plotly, and Pandas.

Data Visualization Tools, Part II

- **Matplotlib:** It's a low-level library providing lots of freedom, requires more coding and knowledge.
- **Pandas Visualization:** It's an easy to use, handy interface built on top of Matplotlib.
- **Seaborn:** It's another high-level interface focused on providing nice styles easily.
- **ggplot:** It's based on R's ggplot2 and uses a grammar of graphics.
- **Plotly:** It's similar to Seaborn, incorporating interactive visualizations.

Data Visualization Tools, Part III

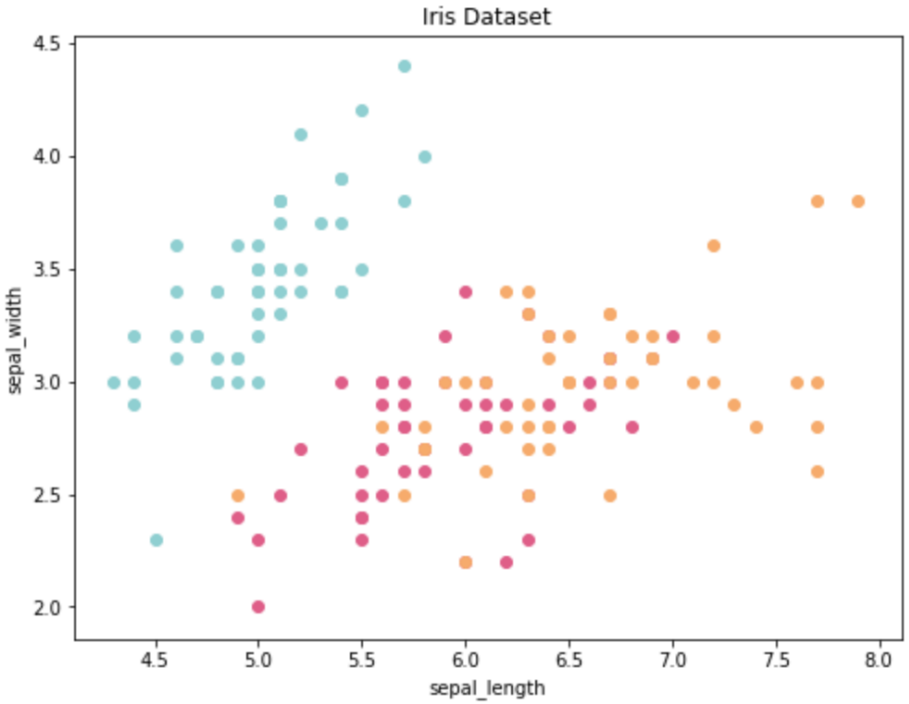
- We will see some examples using our Iris dataset.

```
import pandas as pd
import numpy as np
from sklearn import datasets
iris = pd.read_csv('https:// … /iris.csv')
```

Data Visualization Tools, Part IV

- Customized scatterplots are easy to create.

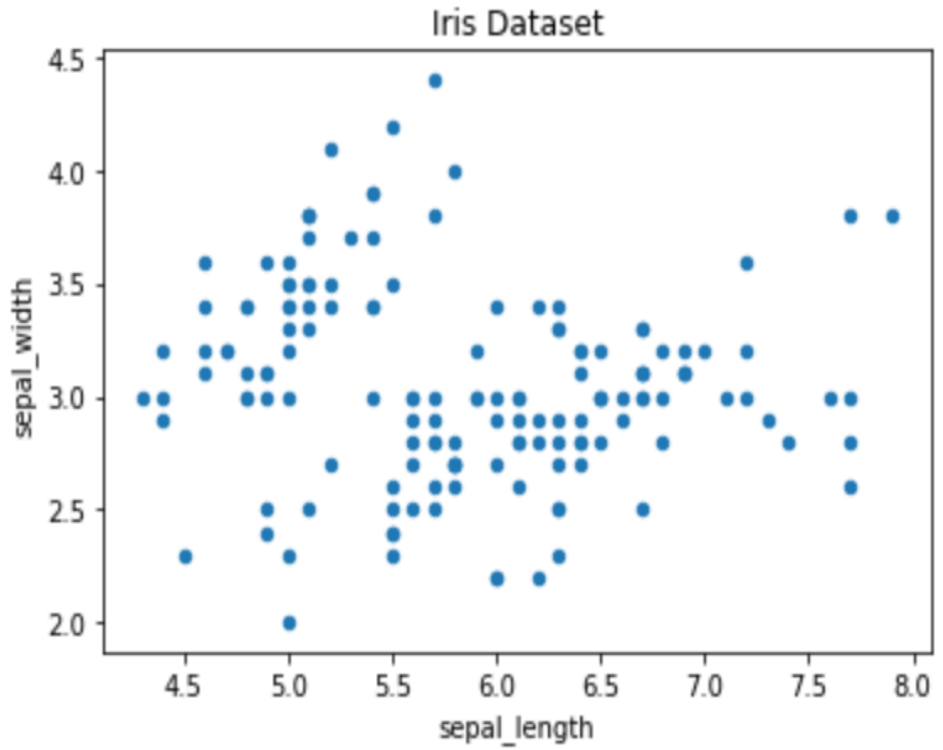
```
colors = {'Setosa':'#8fcfd1',
'Versicolor':'#df5e88',
'Virginica':'#f6ab6c'}
fig, ax = plt.subplots(figsize=(8,6))
for i in range(len(iris['sepal_length'])):
    ax.scatter(iris['sepal_length'][i],
    iris['sepal_width'][i],
    color=colors[iris['class'][i]] )
ax.set_title('Iris Dataset')
ax.set_xlabel('sepal_length')
ax.set_ylabel('sepal_width')
```



Data Visualization Tools, Part V

- In Pandas Visualization, it is more simple. Most information is inferred from the dataset.

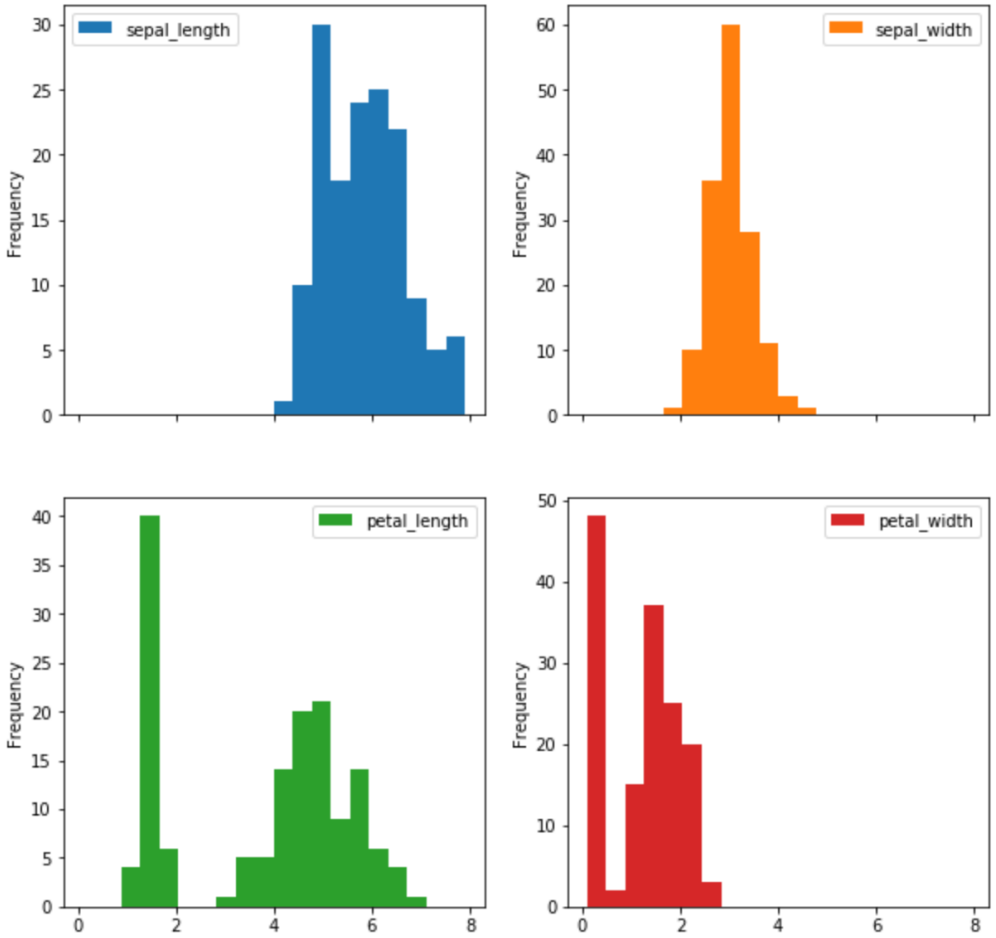
```
iris.plot.scatter(
 x='sepal_length',
 y='sepal_width',
 title='Iris Dataset')
```



Data Visualization Tools, Part VI

- It is also easy to plot arrays and lattices of plots.

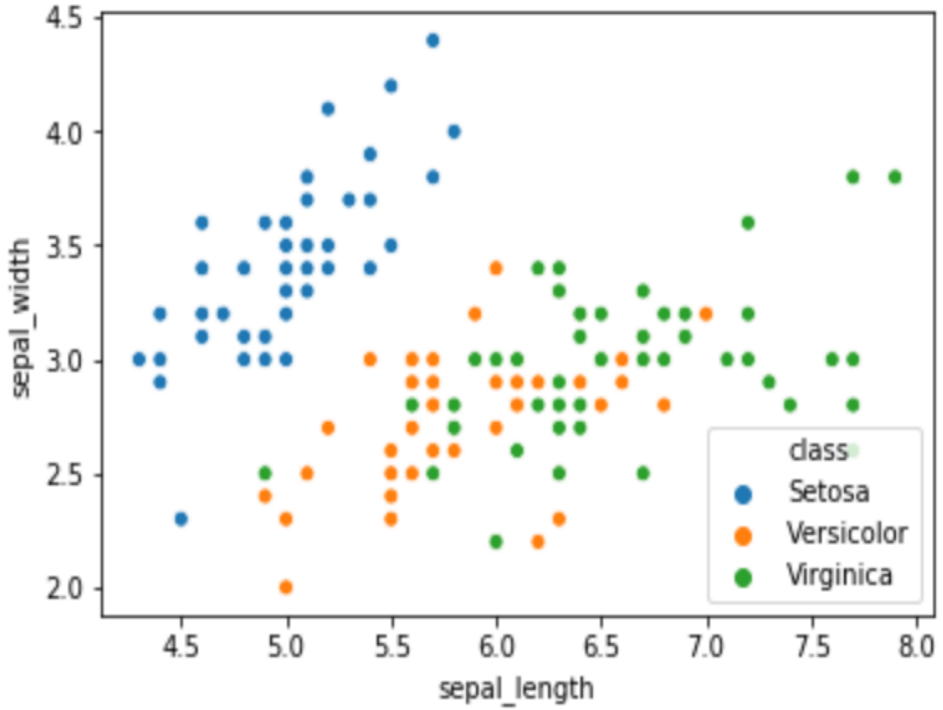
```
iris.plot.hist(
  subplots=True,
  layout=(2,2),
  figsize=(10, 10),
  bins=20)
```



Data Visualization Tools, Part VII

- Seaborn also provides lots of useful visualizations that can be deployed with simple scripts.

```
import seaborn as sns
sns.scatterplot(x='sepal_length',
                y='sepal_width',
               hue='class', data=iris
                )
```

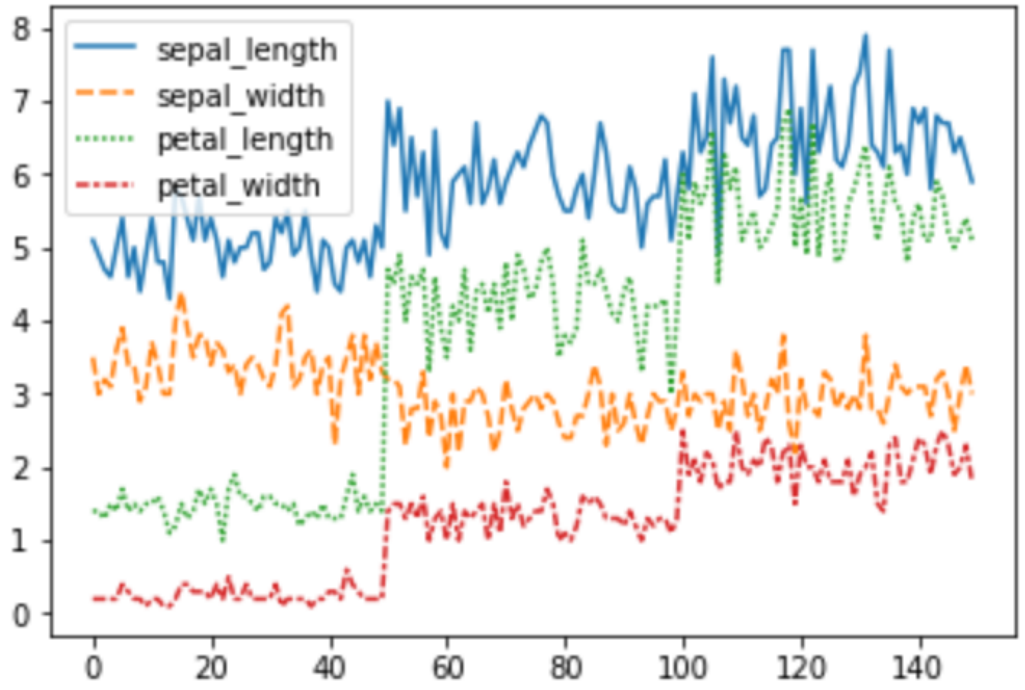


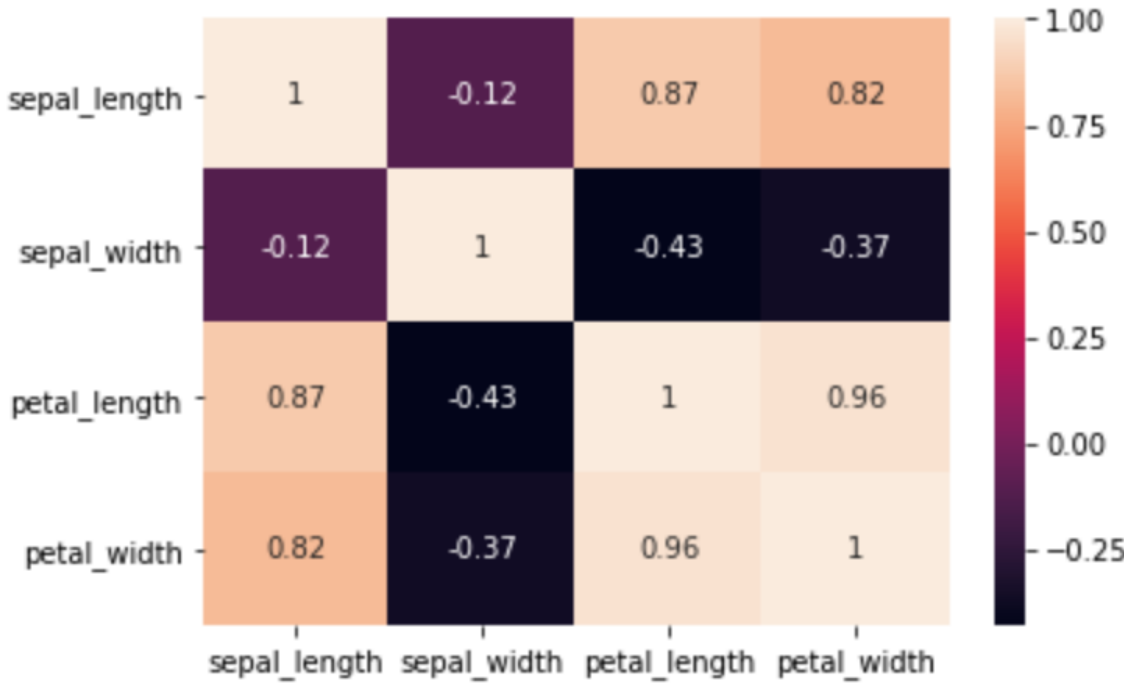
Note how many things are done by default.

Data Visualization Tools, Part VIII

- Other typical data views are also immediate.

```
sns.lineplot(data=iris.drop(['class'], axis=1))
sns.heatmap(iris.corr(), annot=True)
```

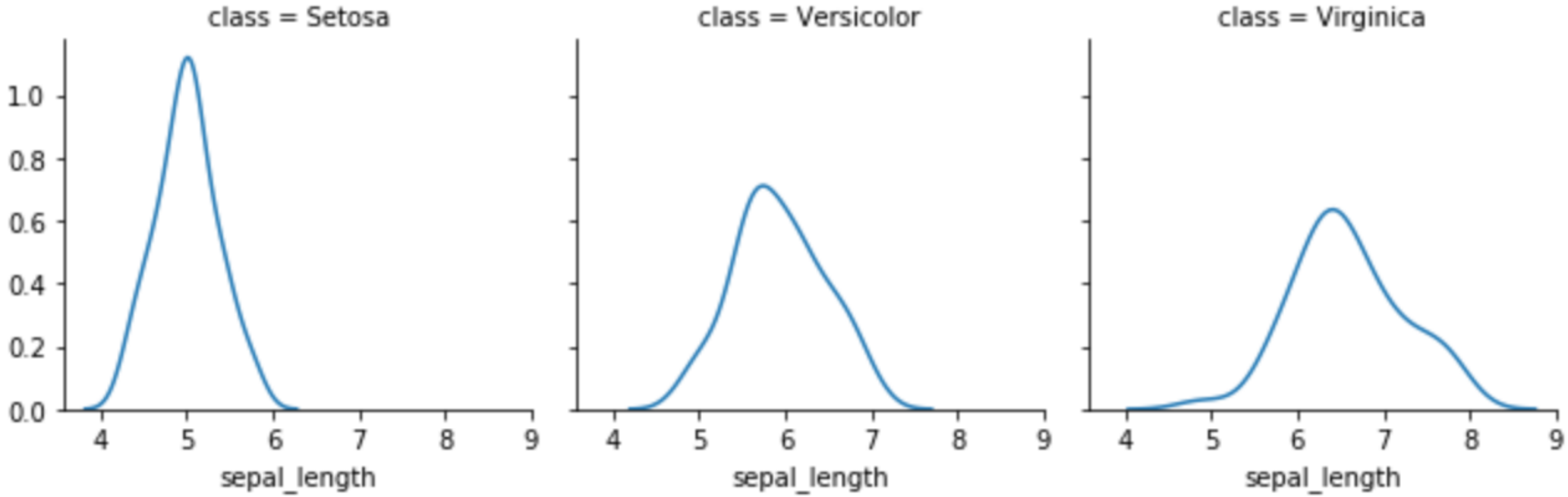




Data Visualization Tools, Part IX

- *Faceting* consists of breaking up data variables across subplots into a single figure.

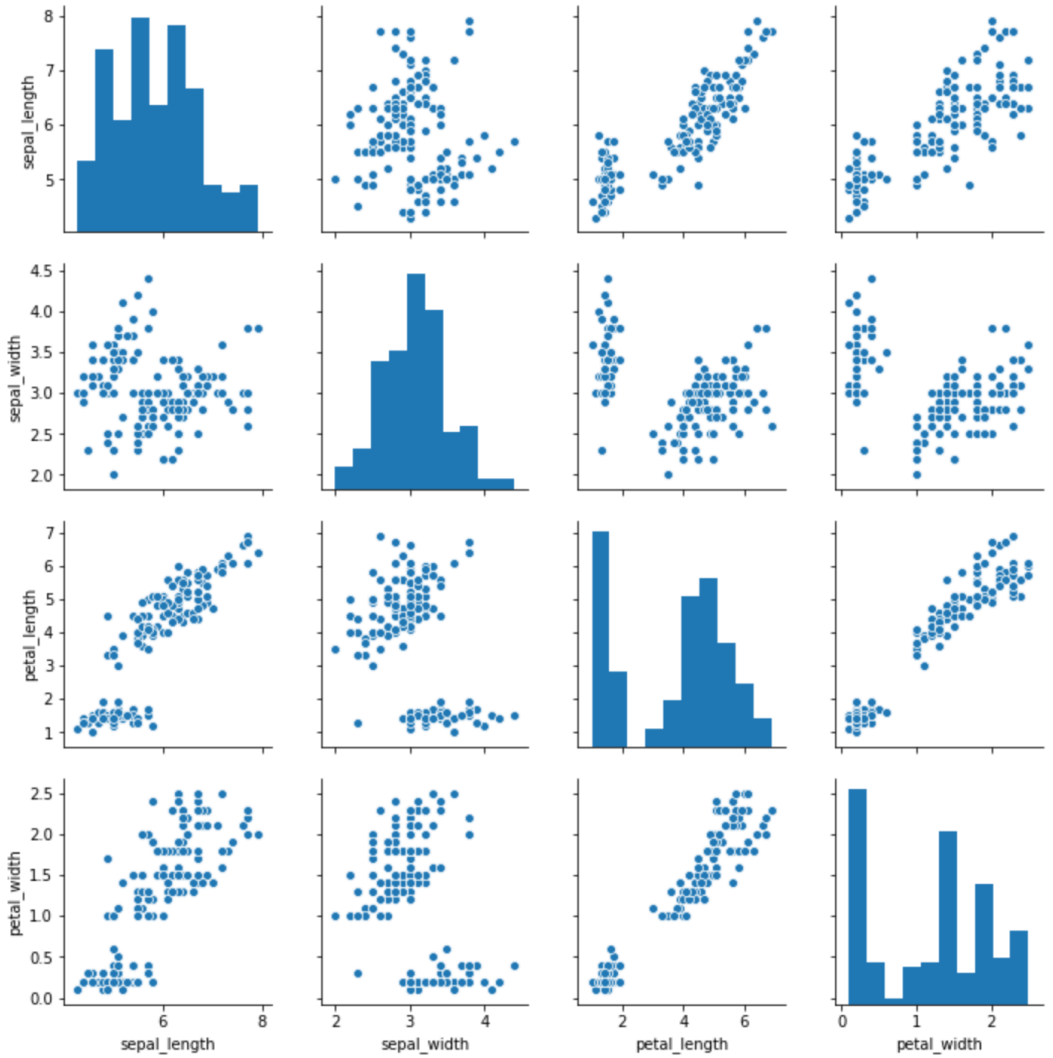
```
g = sns.FacetGrid(iris, col='class')
g = g.map(sns.kdeplot, 'sepal_length’)
```



Data Visualization Tools, Part X

- Finally, graph lattices are easy to deploy.

```
sns.pairplot(iris)
```



The End

Data Visualization Examples

Claudio Delrieux

Data Visualization Examples, Part I

We will use also yet another Python library for visualization: *plotly*. Plotly is an interactive, open-source plotting library that supports many chart types covering a wide range of statistical, financial, geographic, scientific, and three-dimensional use cases.

We will be exploring some advanced functionality in plotly, as well as map rendering and interactive features.

Data Visualization Examples, Part II

Plotly allows the creation of interactive maps. Here, we load a GeoJSON file with the geographic information for U.S. counties.

```
from urllib.request import urlopen
import json
with urlopen('https://raw.githubusercontent.com/plotly/datasets/
          master/geojson-counties-fips.json') as response:
          counties = json.load(response)
```

Data Visualization Examples, Part III

In this map data structure, the *features.id* field stores the reference FIPS code of the counties.

```
counties["features"][0]
```


```
{'geometry': {'coordinates': [[[-86.496774, 32.344437], [-86.717897,
32.402814], [-86.814912, 32.340803], [-86.890581, 32.502974], [-86.917595,
32.664169], [-86.71339, 32.661732], [-86.714219, 32.705694], [-86.413116,
32.707386], [-86.411172, 32.409937], [-86.496774, 32.344437]]], 'type':
'Polygon'}, 'id': '01001', 'properties': {'CENSUSAREA': 594.436, 'COUNTY':
'001', 'GEO_ID': '0500000US01001', 'LSAD': 'County', 'NAME': 'Autauga',
'STATE': '01'}, 'type': 'Feature'}
```

Data Visualization Examples, Part IV

Now, let's load unemployment data by county, which is also indexed by FIPS code.

```
import pandas as pd
df_unemployment =
  pd.read_csv("https://raw.githubusercontent.com/plotly/
  datasets/master/fips-unemp-16.csv",dtype={"fips": str})
```

Data Visualization Examples, Part V

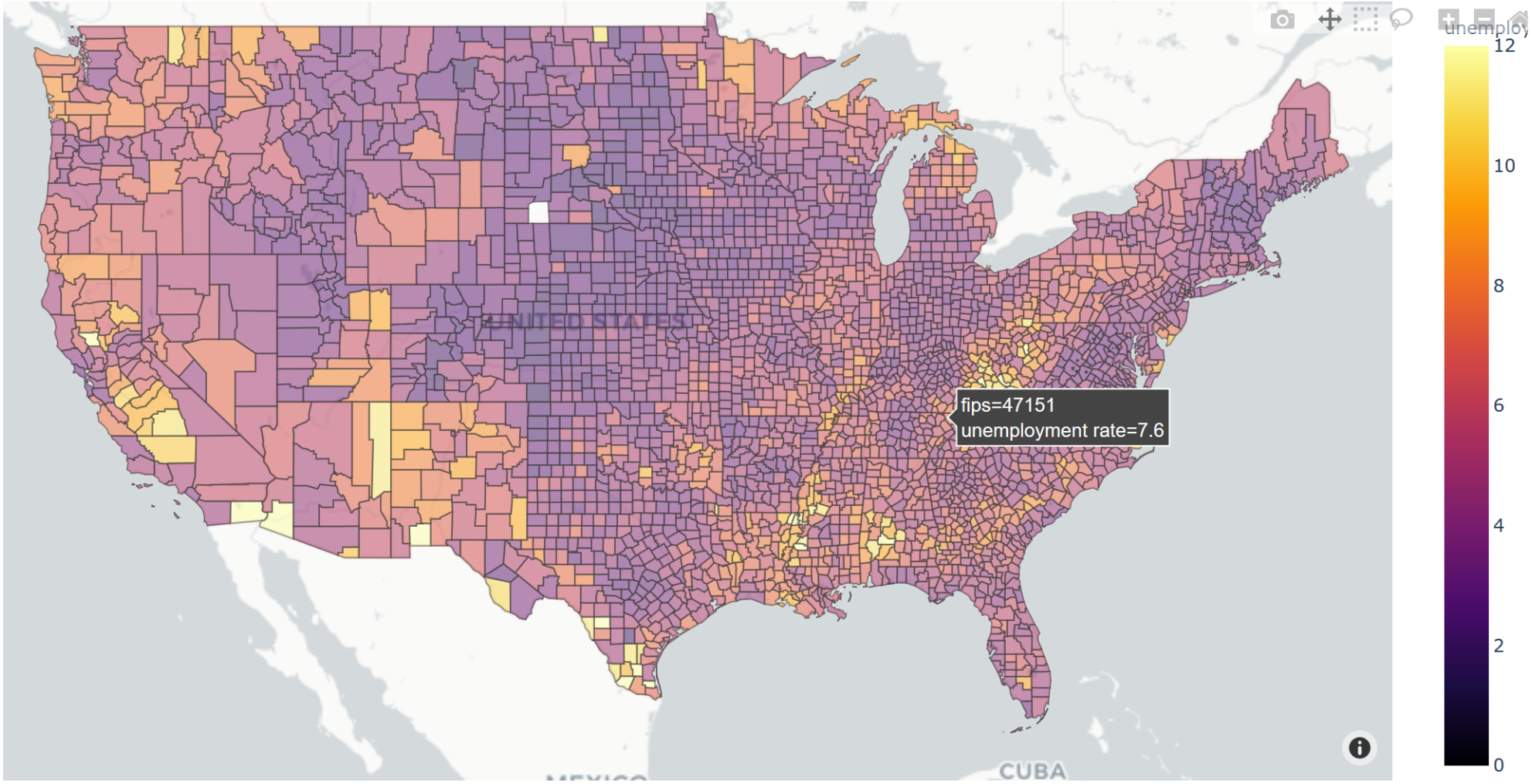
There are several ways to create maps combining this information on plotly. We will be using plotly.express, an easy-to-use, high-level interface to plotly, which operates on a variety of types of data and produces easy-to-style figures.

In particular, we are using choropleth_mapbox.

Data Visualization Examples, Part VI

```
import plotly.express as px
fig = px.choropleth_mapbox(df_unemployment, geojson=counties,
locations='fips', color='unemp',
    color_continuous_scale="Inferno",
    range_color=(0, 12),
    mapbox_style="carto-positron",
    zoom=3, center = {"lat": 37.0902, "lon": -95.7129},
    opacity=0.5,
    labels={'unemp':'unemployment rate'})
```

Data Visualization Examples, Part VII



Data Visualization Examples, Part VIII

Plotly also includes cool interactivity using graphic objects. In the accompanying notebook, you have worked examples using buttons, sliders, range sliders, etc. over different datasets (3D, time series, statistical info, and many others).

The End

Final Comments and Suggestions

Claudio Delrieux

Final Comments and Suggestions, Part I

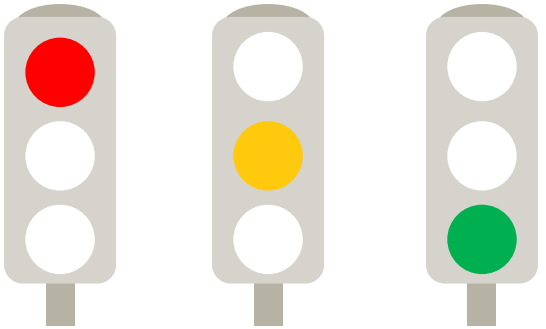
- In this week, we delved into a variety of aspects involving data preparation, analysis, and presentation.
- The additional material provides worked examples of all the specific tools and libraries that are required for the completion of your projects.
- Feel free to contact the instructor if any other issue requires further consideration.

Final Comments and Suggestions, Part II

- Our recommendation is that you focus first on properly understanding the goal of the projects and the expected deliverable. Then, proceed to the three main stages (preparation, analysis, and visualization). In this midterm, the datasets are more tidy and little wrangling will be needed (if at all).

Final Comments and Suggestions, Part III

- Also, read carefully the rubric regarding your type of project (“easy,” “intermediate,” or “difficult”) to fully grasp what is expected, and also the project presentation directives.



Final Comments and Suggestions

The End