MEME19403: Exploratory Data Analysis and Visualisation

Dr Liew How Hui

June 2021

Outline

Revision

In the previous topic, we learn

- Basic data structures in Python
- Python containers
- Python Pandas DataFrame: Develop because Python's integer data structure is slow, Python containers are slow, lack of
- to load "structured" data from different formats (CSV, HTML Table, JSON) to DataFrame or list of DataFrames.

Revision (cont)

Useful modules mentioned last week:

- import numpy as np
- import pandas as pd (optionally depends on some Python Excel libraries mentioned last week)
- import matplotlib.pylab as plt
- import seaborn as sns
- import statsmodels.api as sm
- from sklearn import appropriate modules.

Data Programming

Things we need to know when programming with data:

- Integers and Floating point numbers are not using the same representations in computer.
- Categorical data can be unordered (e.g. Sex) or ordered (e.g. height: short, medium, tall, very tall, etc.)
- Ordered data are usually encoded with corresponding “integers”.
- Social network data are mostly unstructured.
- Business network data are dominated by structured (SQL) data.

In this course, we learn how to process structured data.

Data Programming (cont)

Now, we want to learn to use programming to

- convert data into the correct format — numeric, categorical, etc.
- merge multiple sources of data;
- learn the statistics of the data;
- identify anomalies and impute missing data.

Frameworks: ETL (extract-transform-load) & ELT (for cloud network)

Data Programming (cont)

Summary of DataFrame (2D structured data representation in Python):

- df.shape
- df.index
- df.columns (not in 1D data)
- df.dtypes
- .astype('category')
- df.info() (not in 1D data)
- df.describe()
- df.describe(include='category')

Note: 1D structured data → pd.Series

Indexing Data

Example: Load ISLR Default.csv data (try on the data from the assignment as well). Beware of data without column names.

Identify index:

- Row index: df.index
- Column index: df.columns

Getting data:

- Column: df['columnname']
- Row: df.iloc[row,:]
- Cell/element: df.iloc[row, col]

Indexing Data (cont)

Column Indexing:

- Adding column: df[newcol] = ...
- Removing column: `data=data.drop(['col.name'],axis=1)`
- Selecting column(s): df.col1, df[col1], df[[col1, col2, ...]]
- Modifying column:
  - df.apply
  - data = data[data['species'] != 'Chinstrap'] → Boolean indexing

Indexing Data (cont)

Row Indexing (Not a good idea):

- Adding row: Possible but discourage → join tables
  - df.append(pd.DataFrame('a' : [5, 6], 'b' : [5.5, 5.6]), ignore_index=True)
  - slice and concat / merge
- Removing row(s): df.drop(df.index[2:5], inplace=True)
- Selecting row(s): df.iloc[row,:]
- Modifying row?: Not a good idea. df.iloc[row,:] = ...

Cell Processing (Never a good idea).

Merging Data using Index

- Merging by column (need to have same index): pd.merge(df1,df2,how='outer')
- Merging by rows (need to have same columns): pd.concat([df1,df2,df3])
- Sorting: sorted_df = unsorted_df.sort_values(by=['col1','col2'])

Note: df1.join(df2) requires two Data Frames to have the same row indexing.

Merging Data Example

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index=[0, 1, 2, 3])
df2 = pd.DataFrame({'A': ['A6', 'A7'], 'B': ['B6', 'B7'],
                    'C': ['C6', 'C7'], 'D': ['D6', 'D7']},
                    index=[6, 7])
df3 = pd.DataFrame({'A': ['A11'], 'B': ['B11'],
                    'C': ['C11'], 'D': ['D11']}, index=[11])
df4 = pd.DataFrame({'D': ['D2', 'D3', 'D6', 'D7'],
                    'F': ['F2', 'F3', 'F6', 'F7']},
                    index=[2, 3, 6, 7])
df5 = pd.merge(df1,df4,how='outer')  # to preserve everythi
df6 = pd.concat([df3,df2,df1])
df6.sort_values('A')
```

Data Programming (cont)

Convert Data Frame to other formats:

- .values
- to_numpy()
-

```
.to\_csv(path_or_buf=None, sep=',', na_rep='', float_format=None,
 columns=None, header=True, index=True, index_label=None,
 mode='w', encoding=None, compression='infer', quoting=None,
 quotechar='"', line_terminator=None, chunksize=None,
 date_format=None, doublequote=True, escapechar=None, decimal='.')
```

- .to_excel(): There is no formatting
- .to_html()
- .to_json()
- .to_sql()

Outline

Imperative Programming

Imperative Programming (Year 1 Programming):

- if statement : decision
- for loop : going through a list
- while loop : indefinite loop
- break and continue
- defining functions

Imperative Programming (cont)

Example: Simple linear regression:

$$y_i = ax_i + b, \quad i = 1, \ldots, n.$$

The estimate for a and b are as follows (given in SPM and university year 1 statistics):

$$a = (n\sum x_i y_i - \sum x_i \sum y_i)/(n\sum x_i^2 - (\sum x_i)^2)$$
$$b = (\sum x_i^2 \sum y_i - (\sum x_i y_i)(\sum x_i))/(n\sum x_i^2 - (\sum x_i)^2)$$

Imperative Programming (cont)

Implementation using Year 1 Programming Technique in Python:

```
1  import pandas as pd
2  Default = pd.read_excel("Default.xlsx", index_col=0)  # Need openpyxl
3  def convert(x): return 0 if x=="No" else 1
4  Y = Default.default.apply(convert).values
5  X = Default.balance.values
6
7  if X.shape != Y.shape:
8      print(f"Error length(X)={len(X)} != length(Y)={len(Y)}")
9  n = Y.shape[0]; D = range(n)
0  SX = SY = SX2 = SXY = 0.0
1  for i in D: SX  += X[i]
2  for i in D: SY  += Y[i]
3  for i in D: SX2 += X[i]**2
4  for i in D: SXY += X[i]*Y[i]
5  denom = n * SX2 - SX*SX
6  a = (n * SXY - SX*SY)/denom
7  b = (SX2*SY - SXY*SX)/denom
8  print("a=",a,"\nb=",b)
```

Imperative Programming (cont)

Implementation using Year 1 Programming Technique in C++:

```
1  #include <iostream>
2  // https://www.mlpack.org/doc/mlpack-3.0.4/doxygen/formatdoc.html
3  #include <mlpack/core.hpp> // g++ -fopenmp default1a.cpp -lmlpack
4
5  int main() {
6      arma::mat Default;
7      mlpack::data::Load("Default.csv", Default);
8      std::cout << Default.n_rows << " x " << Default.n_cols << "\n";
9      arma::vec Y = arma::conv_to<arma::vec>::from(Default.row(1)); // default
0      arma::vec X = arma::conv_to<arma::vec>::from(Default.row(3)); // balance
1      //X.print();
2
3      int n = Y.n_elem, i;
4      double SX = 0.0, SY = 0.0, SX2 = 0.0, SXY = 0.0;
5      for(i=0; i<n; i++) { SX  += X[i];      }
6      for(i=0; i<n; i++) { SY  += Y[i];      }
7      for(i=0; i<n; i++) { SX2 += X[i]*X[i];   }
8      for(i=0; i<n; i++) { SXY += X[i]*Y[i]; }
9      double denom = n * SX2 - SX*SX;
20     double a = (n * SXY - SX*SY)/denom;
21     double b = (SX2*SY - SXY*SX)/denom;
22     std::cout << "a=" << a << "\nb=" << b << "\n";
23 }
```

Imperative Programming (cont)

Implementation using Year 2 Object-Oriented Programming Python:

```
1  import pandas as pd
2  Default = pd.read_excel("Default.xlsx", index_col=0)  # Need openpyxl
3  print(Default.head())
4  print(Default.tail())
5  def convert(x): return 0 if x=="No" else 1
6  Y = Default.default.apply(convert).values
7  X = Default.balance.values
8
9  n = Y.shape[0]
0  denom = n*(X**2).sum()-X.sum()**2
1  a = (n * (X*Y).sum() - X.sum()*Y.sum())/denom
2  b = ((X**2).sum()*Y.sum() - (X*Y).sum()*X.sum())/denom
3  print("a=",a,"\n","b=",b)
```

Imperative Programming (cont)

Implementation using Data Processing Python:

```
1  import pandas as pd
2  Default = pd.read_excel("Default.xlsx", index_col=0)  # Need openpyxl
3  print(Default.dtypes)
4  def convert(x): return 0 if x=="No" else 1
5  Y = Default.default.apply(convert).values
6  X = Default.balance.values
7
8  import statsmodels.api as sm
9  X = sm.add_constant(X)
0  model = sm.OLS(Y, X)  # Create Ordinary Least Square object
1  linreg = model.fit()  # Use the method .fit() to fit data
2  print(linreg.params)  # print the 'public' values
3  print(linreg.summary())  # Call the summary() method
4
5  import matplotlib.pylab as plt, numpy as np
6  b, a = linreg.params
7  plt.plot(X, Y, '+')
8  Xrange = np.linspace(X.min(),X.max(), 100)
9  plt.plot(Xrange, a*Xrange+b,'-')
0  plt.plt.show()
```

Imperative Programming (cont)

Example: Process WOS Journal List

The Excel approach: Can be fast for small PDF documents.

The Python approach: Can be painful for someone who is not familiar with Python programming:

- Extract Text from PDF document
- Convert Text (unstructured data) to structured data (e.g. Python list)
- Programming involve: for loop & append list
- Check for ‘outliers”

code/wos_journals.py

Outline

Descriptive Statistics

There are two ways to explore / inspect data:

- using statistical measures → descriptive statistics
- using diagrams → data visualisation

They are by no means isolated, e.g. histogram is an essential tool in descriptive statistics.

Outline

Descriptive Statistics

Classification of Univariate Data in Statistics:

- Categorical
- Ordinal & Integral
- Numeric

Basic Statistical Summary:

- Find min, max, mean, variance: df.min(axis=0), df.max(axis=1), df.mean(), df.var(), axis=0 or 1 for columns and rows respectively.
- `df.describe()`: Numeric data only
- `df.describe(include=['object'])`
- `df.describe(include='all')`

Frequency Table

For categorical data and integers (ordinal data), frequency table is useful:

- Particular column: df[col].value_counts()
- All columns: [df[c].value_counts() for c in list(df.select_dtypes(include=['O']).columns)]

Numeric data can be converted to categorical data using 'cut':

- pd.cut(df[col], bins=5).value_counts(sort=False)
- df[col].value_counts(bins=5,sort=False)

Histogram, etc.

Histogram:

- No “stem and leaf plot”
- df[col].plot.hist() → calls to matplotlib
- plt.hist(df[col]).
  For density, add “density=True”

Cummulative plot:

- plt.hist(df[col], cumulative=True)
- plt.hist(df[col], cumulative=True, density=True)

Kernel Density Estimates (KDE)

To form a KDE, we place a smooth strongly peaked function at K the position of each data point and then add up the contributions from all the functions to obtain a smooth curve called the ***kernel density estimator***

$$D_h(x; \{x_i\}) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h}). \tag{1}$$

where $(x_1, x_2, \cdots, x_n)$ are assumed to be a univariate independent and identically distributed sample drawn from some distribution with an unknown density f.

KDE (cont)

A typically used function is the *Gaussian kernel* (`scipy.stats.gaussian_kde`) and it is used in Matplotlib:

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2). \tag{2}$$

- df[col].plot.kde()
  For density KDE, add “density=True”.
- sns.kdeplot(df[col])

Descriptive Statistics (cont)

Examples of Clean Data:

- flame.txt → simple nonlinear
- Default.csv → simulated data from ISLR

Examples of 'Bad' Data:

- penguins_size.csv → contains missing values

Example: code/p_stat_flame.py

QQ Plot

To confirm that a given set of points is distributed according to some specific distribution such as Gaussian (or some other) distribution, we could compare a histogram or KDE of the data set directly against the theoretical density function, but it is notoriously difficult to compare distributions that way, especially out in the tails (because they are small and it is difficult differentiate)

Answer: QQ plot $\rightarrow$ "F^{-1}" vs "G^{-1}".

Here F and G are the cumulative probability distribution functions.

QQ Plot (cont)

- scipy.stats.probplot(df[col], dist=”norm”, plot=plt)
- statsmodels.api.qqplot(df[col],line=’45’)

Note: Need to normalise the data before applying QQ-plot

Finding Relations Between Data

- Cross Tabulation
- Covariance Matrix ???
- Correlation Analysis

Cross Tabulation

Also known as contingency table.

Use to comparing “categorical data”:

- pd.crosstab(a, [b, c], rownames=[’a’], colnames=[’b’, ’c’])

Related: df.groupby, df.pivot_table (for relations between categorical data & numeric data?)

Correlation Analysis

Covariance: $\text{Cov}(x, y) = \dfrac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{n-1}$.

- df.cov() $\rightarrow$ covariance matrix

(Pearson) Correlation: $\text{Cor}(x, y) = \dfrac{\text{Cov}(x, y)}{sd(x) \times sd(y)}$.

- corrMatrix = df.corr() $\rightarrow$ correlation matrix
- sns.heatmap(corrMatrix, annot=True)

Other methods: kendall, spearman

Correlation Analysis (cont)

Data Type	Categorical.out	Numeric.out
Categorical.in	χ^2, mutual information	Kendall, ANOVA
Numeric.in	Kendall, ANOVA	Pearson (normal), Spearman (non-normal)

- scipy.stats.chi2_contingency, sklearn.metrics.mutual_info_score
- scipy.stats.kendalltau, ...
- scipy.stats.pearsonr, scipy.stats.spearmanr

ANOVA

An ANOVA test is a way to find out if survey or experiment results are significant. In other words, they help you to figure out if you need to reject the null hypothesis or accept the alternate hypothesis.

Basically, you're testing groups to see if there's a difference between them.

ANOVA (cont)

Examples of when you might want to test different groups:

A group of psychiatric patients are trying three different therapies: counseling, medication and biofeedback. You want to see if one therapy is better than the others. A manufacturer has two different processes to make light bulbs. They want to know if one process is better than the other.

Students from different colleges take the same exam. You want to see if one college outperforms the other.

Outline

Data Visualisation

Some good practices in designing charts:

- Keep the graph simple with appropriate labels

Some popular data visuals:

- histogram (mentioned earlier)
- Bar chart, compound (bar) chart
- pie chart and doughnut diagram
- radar chart
- line chart
- scatter plot, bubble chart
- maps

Vis: Bar Chart, Histogram

Bar chart: For categorical data.

```
langs = ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')
y_pos = np.arange(len(langs))
performance = [10,8,6,4,2,1]
sr = pd.Series(performance, index=langs)

### Pandas plotting (rely on Matplotlib)
sr.plot.bar(y='Usage')  # For horizontal bar plot: sr.plot.barh(y='Usage'

### Using Matplotlib straight away
plt.rcdefaults()
plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, langs)
plt.ylabel('Usage')
plt.title('Programming language usage')
plt.show()
```

Histogram: For numeric data, mentioned earlier.

Vis: Bar Chart, Histogram (cont)

Compound Bar Charts = Stacked Bar Graph

```
import numpy as np, matplotlib.pylab as plt
N = 5
menMeans   = (20, 35, 30, 35, 27)
womenMeans = (25, 32, 34, 20, 25)
menStd     = (2, 3, 4, 1, 2)
womenStd   = (3, 5, 2, 3, 3)
ind = np.arange(N) # the x locations for the groups
width = 0.35       # the width of the bars: can also be len(x) sequence

p1 = plt.bar(ind, menMeans, width, yerr=menStd)
p2 = plt.bar(ind, womenMeans, width,
             bottom=menMeans, yerr=womenStd)

plt.ylabel('Scores')
plt.title('Scores by group and gender')
plt.xticks(ind, ('G1', 'G2', 'G3', 'G4', 'G5'))
plt.yticks(np.arange(0, 81, 10))
plt.legend((p1[0], p2[0]), ('Men', 'Women'))

plt.show()
```

Vis: Pie Chart

```
1  import matplotlib.pylab as plt
2  fig = plt.figure()
3  ax = fig.add_axes([0,0,1,1])
4  ax.axis('equal')
5  langs = ('Python', 'C++', 'Java', 'Perl',
6    'Scala', 'Lisp')
7  performance = [10,8,6,4,2,1]
8  ax.pie(performance, labels=langs,
9    autopct='%1.2f%%')
0  plt.show()
```

Vis: Doughnut Diagram/Chart

```
1  import matplotlib.pylab as plt, numpy as np
2  langs = ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')
3  performance = [10,8,6,4,2,1]
4  fig, ax = plt.subplots(figsize=(6, 3), subplot_kw=dict(aspect="equal"))
5  wedges, texts = ax.pie(performance, wedgeprops=dict(width=0.5),
6                         startangle=-40)
7  bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
8  kw = dict(arrowprops=dict(arrowstyle="-"),
9            bbox=bbox_props, zorder=0, va="center")
0  for i, p in enumerate(wedges):
1      ang = (p.theta2 - p.theta1)/2. + p.theta1
2      y = np.sin(np.deg2rad(ang))
3      x = np.cos(np.deg2rad(ang))
4      horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
5      connectionstyle = "angle,angleA=0,angleB={}".format(ang)
6      kw["arrowprops"].update({"connectionstyle": connectionstyle})
7      ax.annotate(langs[i] + " (" + str(performance[i]) + ")", xy=(x, y),
8                  xytext=(1.35*np.sign(x), 1.4*y),
9                  horizontalalignment=horizontalalignment, **kw)
20
21 ax.set_title("Matplotlib doughnut chart")
22 plt.show()
```

Vis: Radar / Spider / Star Chart

"Polar" coordinate plot

Simple example:

vis_radar.py

Advanced example:

`https://matplotlib.org/3.1.0/gallery/specialty_plots/radar_chart.html`

Vis: Line Chart

A *line plot* or a *line chart* can be used to show the relationship between two data and time series. It's a good indicator of trends, accumulations, decreases, and changes.

Application: time series.

```
df = pd.DataFrame({'x': np.arange(-2*np.pi,2*np.pi,0.1)})
df['sin'] = np.sin(df['x'])
df['zer'] = df['x']*0
df.plot(x='x',y=['zer','sin'])  # call matplotlib to plot
```

Vis: Area Chart

An *area plot* (or area chart, area graph) displays "time series" data visually using "area" instead of "line" and is widely used in because it is visually appealing.
Example: MySejahtera Weekly Statistics.

It can illustrate accumulated totals using numbers or percentages by area underneath the line over time.

```
1  import pandas as pd
2  import matplotlib.pylab as plt
3  df = pd.DataFrame({
4      'sales': [3, 2, 3, 9, 10, 6],
5      'signups': [5, 5, 6, 12, 14, 13],
6      'visits': [20, 42, 28, 62, 81, 50],
7  }, index=pd.date_range(start='2018/01/01',
8                         end='2018/07/01', freq='M'))
9  ax = df.plot.area(); plt.show()
```

Vis: Scatter Plot

```
1  # https://pythonspot.com/matplotlib-scatterplot/
2  import numpy as np, matplotlib.pyplot as plt
3
4  # Create data
5  N = 500
6  x = np.random.rand(N)
7  y = np.random.rand(N)
8  colors = (0,0,0)
9  area = np.pi*3
0
1  plt.scatter(x, y, s=area, c=colors, alpha=0.5)
2  plt.title('Scatter plot pythonspot.com')
3  plt.xlabel('x')
4  plt.ylabel('y')
5  plt.show()
```

Scatter plot (x is unordered) vs Line / Area plot (x is ordered).

Vis: Pair plot

```
1  # https://stackoverflow.com/questions/56188305/matplotlib-
2  import pandas as pd, numpy as np, matplotlib.pyplot as plt
3  import seaborn as sns
4
5  df = pd.DataFrame({
6      'ozone': [1.0, 0.3483, 0.6985, -0.6129],
7      'radiation': [0.3483, 1.0, 0.2941, -0.1274],
8      'temperature':[0.6985, 0.2941, 1.0, -0.4971],
9      'wind': [-0.6129, -0.1274, -0.4971, 1.0]})
0
1  g = pd.plotting.scatter_matrix(df, figsize=(10,10),
2    marker='o', hist_kwds={'bins': 10}, s=60, alpha=0.8)
3
4  ### A 'better' alternative?
5  #sns.pairplot(df, diag_kws={'bins': 10})
6
7  plt.show()
```

Vis: Bubble Plot

A **bubble plot** is very close to a scatterplot. With Matplotlib, we will construct them using the same scatter function. However, we will use the ‘s’ argument to map a third numerical variable to the size of the marker. This will give us an additive information on the same graphic!

```
1  # https://python-graph-gallery.com/270-basic-bubble-plot/
2  import matplotlib.pyplot as plt, numpy as np
3
4  x = np.random.rand(40)
5  y = np.random.rand(40)
6  z = np.random.rand(40)
7
8  plt.scatter(x, y, s=z*1000, alpha=0.5)
9  plt.show()
```

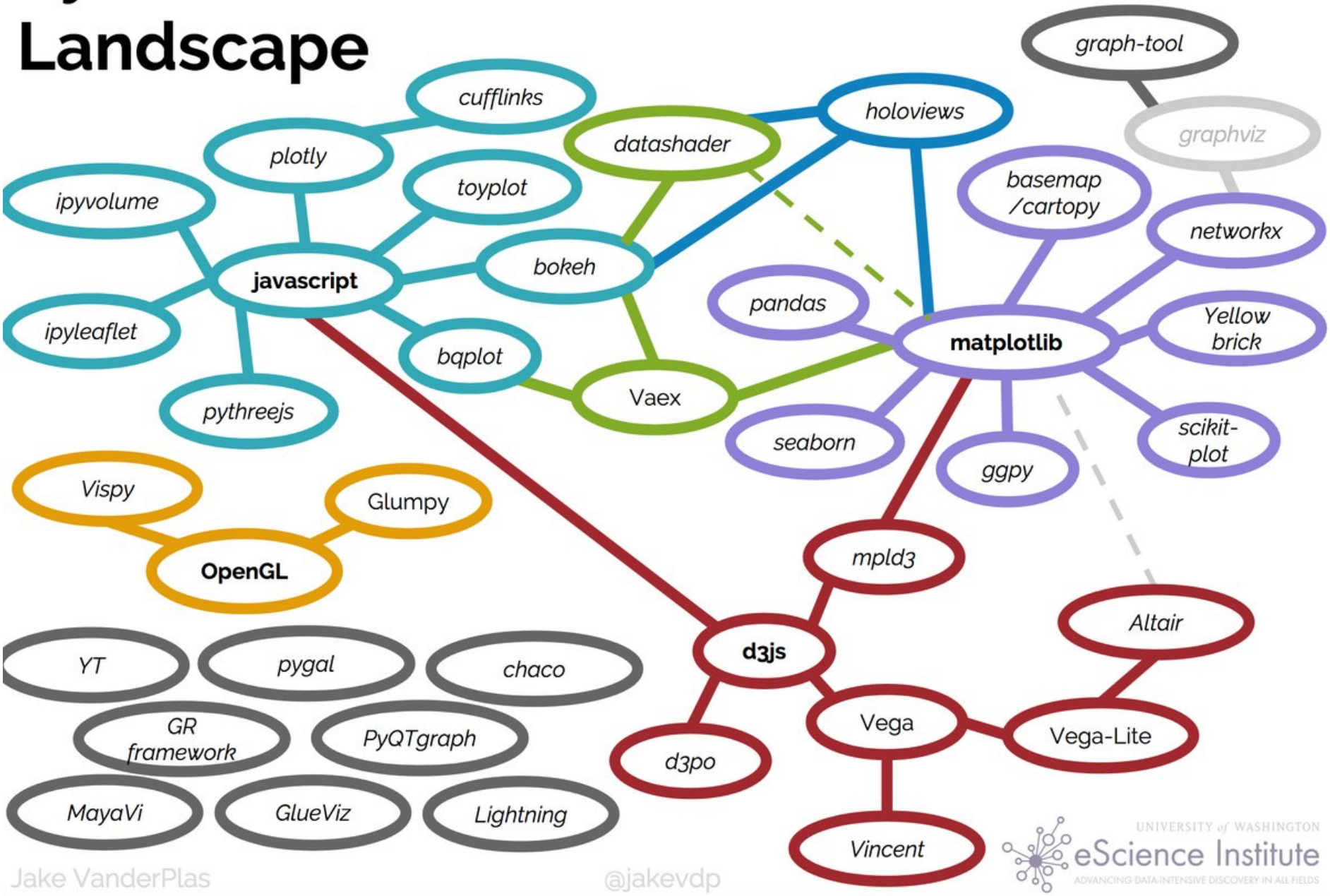
Vis: Maps

Following are a series of examples that illustrate how to use Basemap instance methods to plot your data on a map. More examples are included in the examples directory of the basemap source distribution. There are a number of Basemap instance methods for plotting data:

- contour(): draw contour lines.
- contourf(): draw filled contours.
- imshow(): draw an image.
- pcolor(), pcolormesh(): draw a pseudocolor plot.
- plot(): draw lines and/or markers.
- scatter(): draw points with markers.
- quiver(): draw vectors.
- barbs(): draw wind barbs.
- drawgreatcircle(): draw a great circle.

Outline

Python's Visualization Landscape



Dashboards

JavaScript-based:

- D3.js A JavaScript library for manipulating documents based on data.
- C3.js D3-based reusable chart library.
- https://www.chartjs.org/docs/master/
- https://pyecharts.org/
- PyXley Python helpers for building dashboards using Flask and React

Dashboards (cont)

Web + JavaScript-based:

- Dash: Based on Flask, Plotly, etc.
- Metabase dashboard
- Apache Superset (incubating) is a modern, enterprise-ready business intelligence web application

Json + JavaScript-based:

- Vega: A Visualization Grammar (https://vega.github.io/vega/)

Dashboards (cont)

Desktop GUI-based:

- Matplotlib is intended for use in mathematics / scientific / engineering applications. It is possible to use Panel to turn Matplotlib into a dashboard, see `https://coderzcolumn.com/tutorials/data-science/how-to-create-dashboard-using-python-matplotlib-panel`
- PyQtGraph is a pure-python graphics and GUI library built on PyQt4 / PySide and numpy. It is intended for use in mathematics / scientific / engineering applications.
- Enthought Tool Suite (ETS): For scientific visualisation.

Outline

Data Cleaning

- Removing / Isolating outliers?
- Removing / Imputing missing values?

Missing values

- Find missing values:
  - df[col] = df[col].replace(special value, np.nan)
  - df.isnull().sum() → count NA in each column
  - df.isnull().sum(axis=1) → count NA in each row
- Handling missing values:
  - Removing: df = df.dropna()
  - Imputation:
    - Categorical: df[col][df[col].isnull()] = df[col].value_counts().idxmax()
    - Numeric: df[col][df[col].isnull()] = df[col].mean()