DATA2001: Data Science, Big Data and Data Diversity

W2: Data Cleaning and Exploration with Python

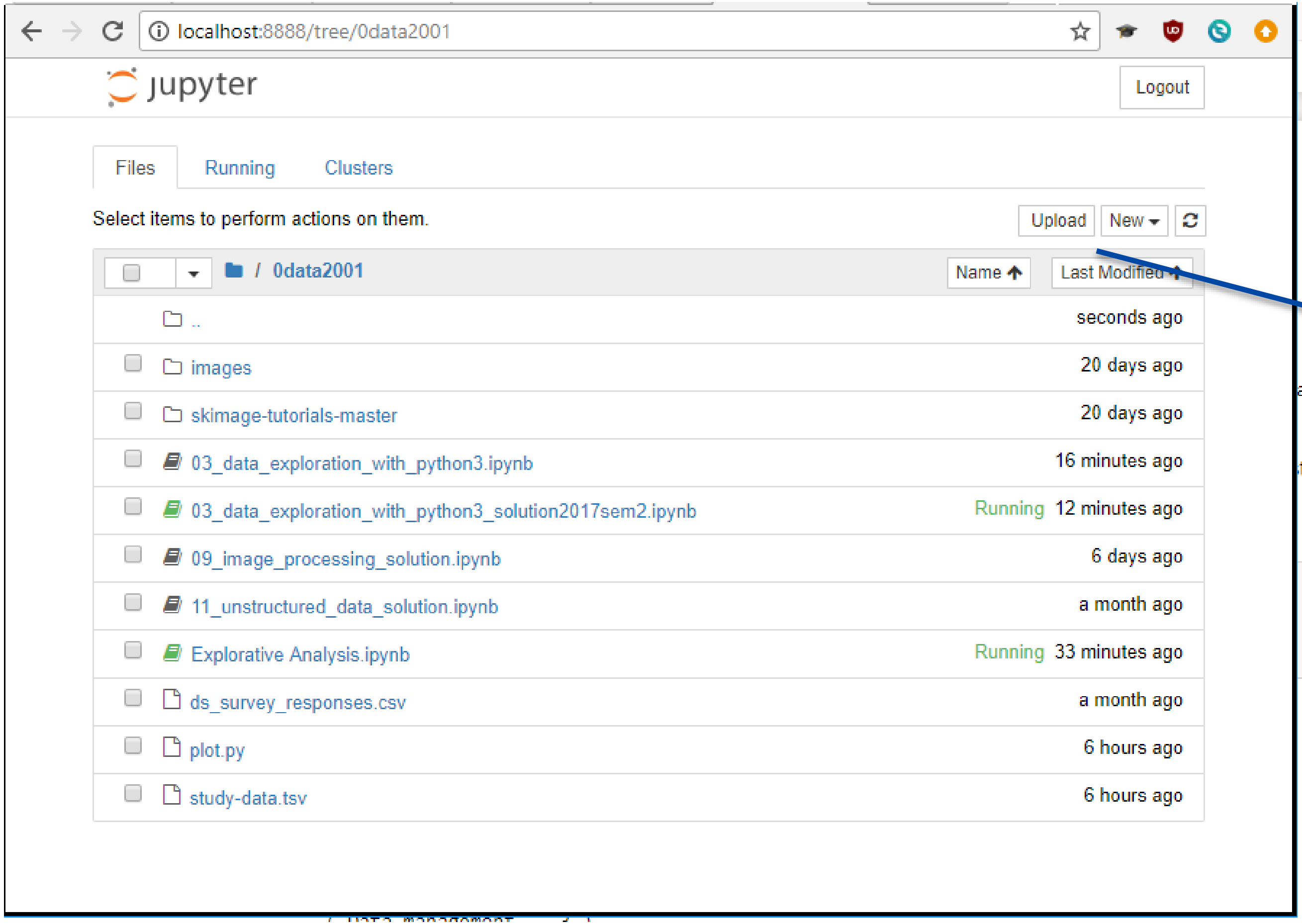
Presented by Dr Ali Anaissi
School of Computer Science

THE UNIVERSITY OF SYDNEY

Jupyter Notebooks:
The Python Environment in DATA2001

Jupyter Notebooks support interactive Data Science with Python

- IPython interactive command shell offers:
  - Introspection
  - Tab completion
  - Command history
- Jupyter runs in a browser and supports:
  - Sharing and documenting of live code
  - Data cleaning, visualisation, machine learning, …
  - Jupyter's gallery of interesting notebooks: https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks
- We provide Jupyter servers which run Python 3 https://ucpu0.ug.it.usyd.edu.au/ (remember you need to be using VPN, if off campus)



1. **Click here for file open dialogue**
2. **Click upload next to file name**

Installing Python and Jupyter using Anaconda

- You can use our Jupyter server
  - but this is a shared resource
- If you wish, you can install Python and Jupyter privately, eg using Anaconda Distribution, which includes Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science.

Python and Data Science Libraries

Python Background

- Students who did DATA1002: this should mostly be revision
  - If you didn't really master pandas, matplotlib before, do so now!
  - Also note the following key differences
    - More sophisticated ways to consider the kinds of data (not just numerical/castegorical)

- Students who learned Python elsewhere (eg INFO1110): you need to learn how to use particular libraries (pandas, matplotlib etc) from the examples here, and online resources

Python – General Concepts

- general program syntax
- variables and types
  - integer and float numbers, string types, type conversion
  - list, dictionary, tuple and set
- condition statements (if/elif/else)
- for loops, ranges
- functions
  - print(), len(), lower(), upper(), …
  - nesting of functions; example: print(len(*str*.upper()))

Data Preparation and Exploration with Python

Objective

Learn Python tools for exploring a new data set programmatically.

Lecture

- Data types, cleaning, preprocessing
- Descriptive statistics, e.g., median, quartiles, IQR, outliers
- Descriptive visualisation, e.g., boxplots, confidence intervals

Readings

- Data Science from Scratch: Ch 4-5

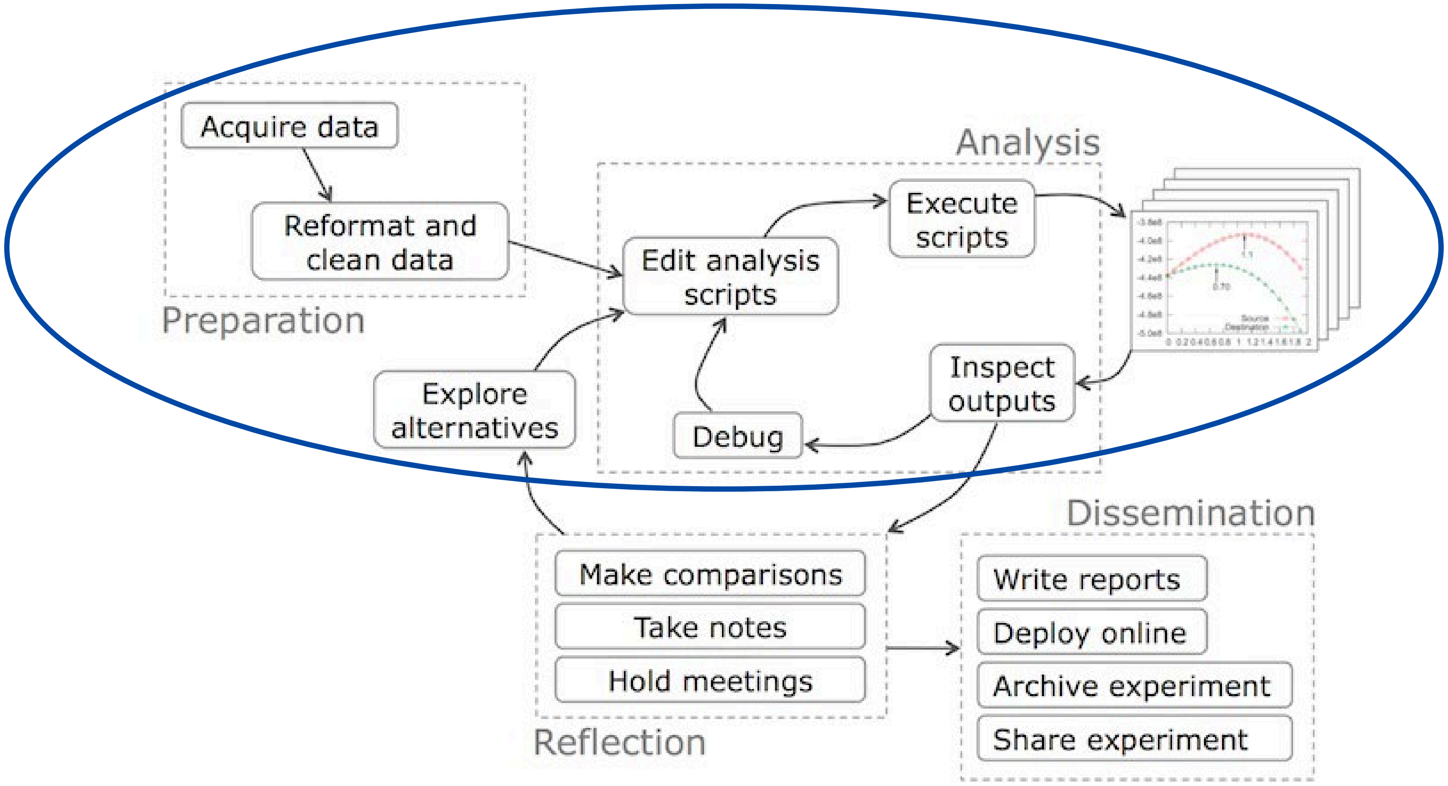
Exercises

- matplotlib: Visualisation
- numpy/scipy: Descriptive stats

TODO in W2/W3

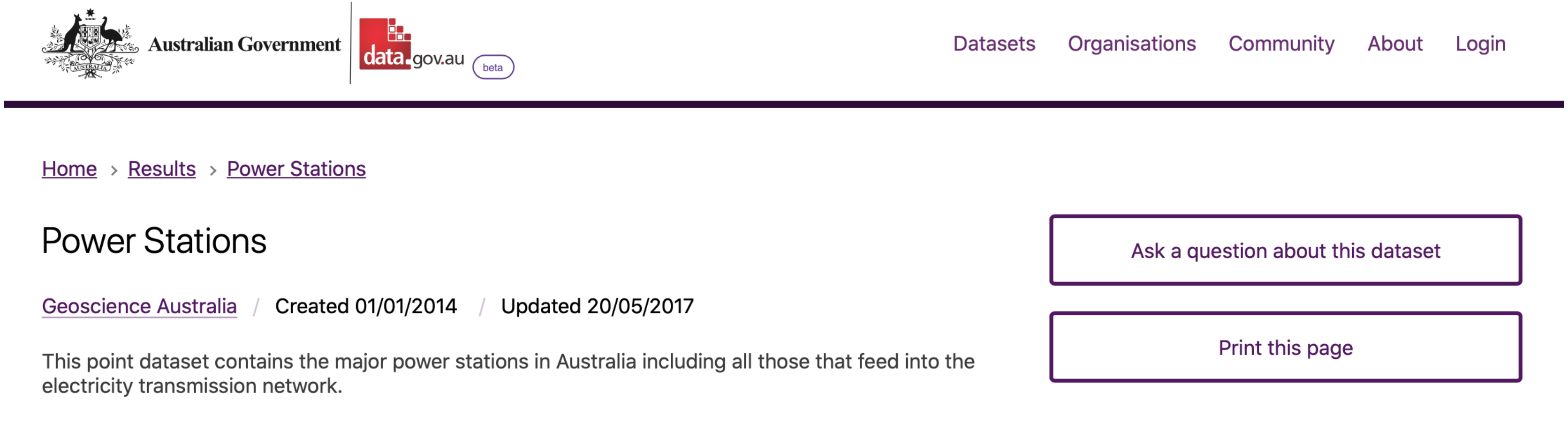
- Grok Python modules
- Explore the survey data

Exploratory Analysis Workflow



Example: Analysis of Major Power Stations in Australia

- dataset from data.gov.au



Source: https://data.gov.au/dataset/ds-ga-04661f51-82ee-144e-e054-00144fdd4fa6/details?q=power%20stations

- How can we load this data into Python?
- Which data preparation steps are needed?

Preliminaries:
Types of Data and Levels of Measurement

Level of Measurements and Type of Data

- Categorical
  - Nominal
    - Dichotomous
  - Ordinal
- Quantitative
  - Interval
  - Ratio

Other data types:

- Text
- Images, Video

Categorical Data

- A categorical variable is also known as a discrete or qualitative variable and can have two or more categories.
- It is further divided into two variants, **nominal** and **ordinal.**
  - These variables are sometimes coded as numerical values, or as strings.

Nominal Data

- This is an unordered category data. This type of variable may be "label-coded" in numeric form but these numerical values have no mathematical interpretation and are just labeling to denote categories.
  For example, colours: black, red and white can be coded as 1, 2 and 3.

What main industry have you worked in? *

Choose

What key experience do you have? *

- [ ] Relational databases
- [ ] NoSQL
- [ ] Information retrieval

- Values are names
- No ordering is implied
- Eg jersey numbers

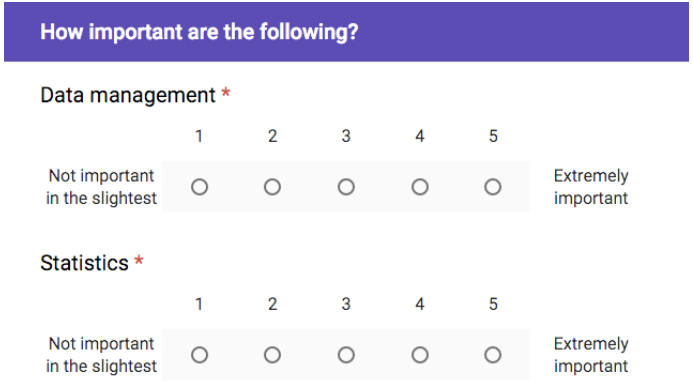
Dichotomous Data

- A dichotomous is a type of nominal data that can only have two possible values, e.g. true or false, or presence or absence. These are also sometimes referred as binary or Boolean variables.

- True (1) or false (0)
- Correct / Incorrect
- Student / Academic

Ordinal Data

- This is ordered categorical data in which there is strict order for comparing the values, so a labelling as numbers is not completely arbitrary. For example, human height (small, medium and high) can be coded into numbers small = 1, medium = 2, high = 3.



- Values are ordered
- No distance is implied
- Eg rank, agreement

Interval Data

It is a variable in which the interval between values has meaning and there is no true zero value.

- Values encode differences
- equal intervals between values
- No true zero
- Addition is defined
- e.g Celsius temperature scale
  - can't express "no temperature"
- e.g. dates

Ratio Data

It is variable that might have a true value of zero and represents the total absence of the variable being measured. For example, it makes sense to say a Kelvin temperature of 100 is twice as hot as a Kelvin temperature of 50 because it represents twice as much the thermal energy (unlike Fahrenheit temperatures of 100 and 50).

How many years professional experience do you have? *

Your answer

How many years programming experience do you have? *

Your answer

- Values encode differences
- Zero is defined
- Multiplication defined
- Ratio is meaningful
- Eg length, weight, pressure, income

Levels of Measurement

	Nominal	Ordinal	Interval	Ratio
Countable, equality defined	✓	✓	✓	✓
Order defined		✓	✓	✓
Difference defined (addition, subtraction)			✓	✓
Zero defined (multiplication, division)				✓

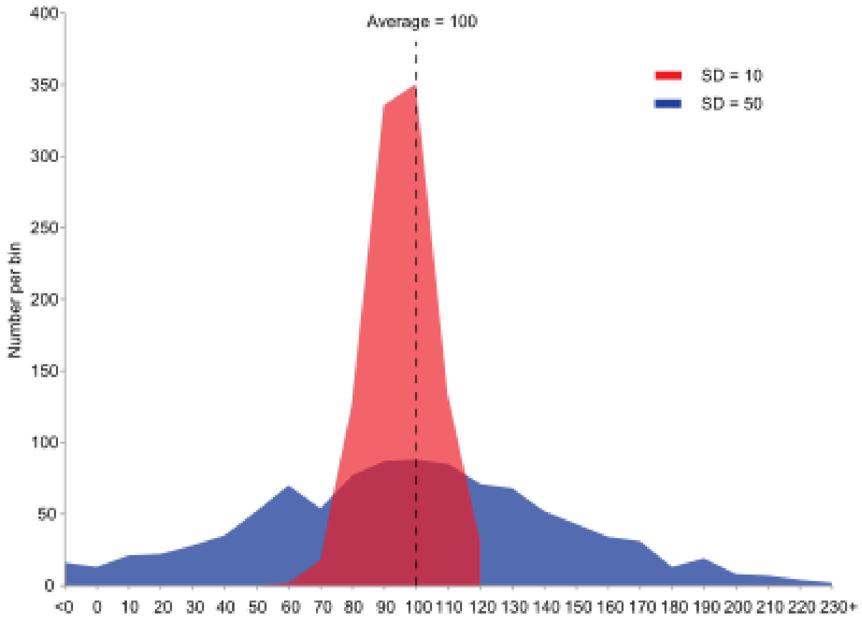
Measures of Central Tendency

	Nominal	Ordinal	Interval	Ratio
Mode	✓	✓	✓	✓
Median		✓	✓	✓
Mean			✓	✓

Measures of Dispersion

	Nominal	Ordinal	Interval	Ratio
Counts / Distribution	✓	✓	✓	✓
Minimum, Maximum		✓	✓	✓
Range		✓	✓	✓
Percentiles		✓	✓	✓
Standard deviation, Variance			✓	✓

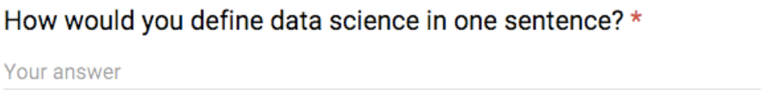
What does variance and standard deviation tell us?



Samples from two populations with the same mean but different variances. The red population has mean 100 and variance 100 (SD=10) while the blue population has mean 100 and variance 2500 (SD=50).

https://en.wikipedia.org/wiki/Variance

What about Text Data or Images?


How would you define data science in one sentence? *

Your answer

- Not defined as traditional data type in statistics
- Requires interpretation, coding or conversion
- More in future lectures…

Data Acquisition and Cleaning

Data Acquisition – Where does data come from?

- File Access
  - You or your organisation might already have a data set, or a colleagues provides you access to data.
  - Or: Web Download from an online data server (e.g. data.gov.au)
  - Typical exchange formats: CSV, Excel, sometimes also XML
- Programmatically
  - Scrapping the web (HTML)
  - or using APIs of Web Services (XML/JSON)
    -> Cf. textbook, Ch 9
- Database Access
- Collect data yourself, eg. via a survey

In tutorials this week: Using data from an online survey

Cleaning and Transforming Data

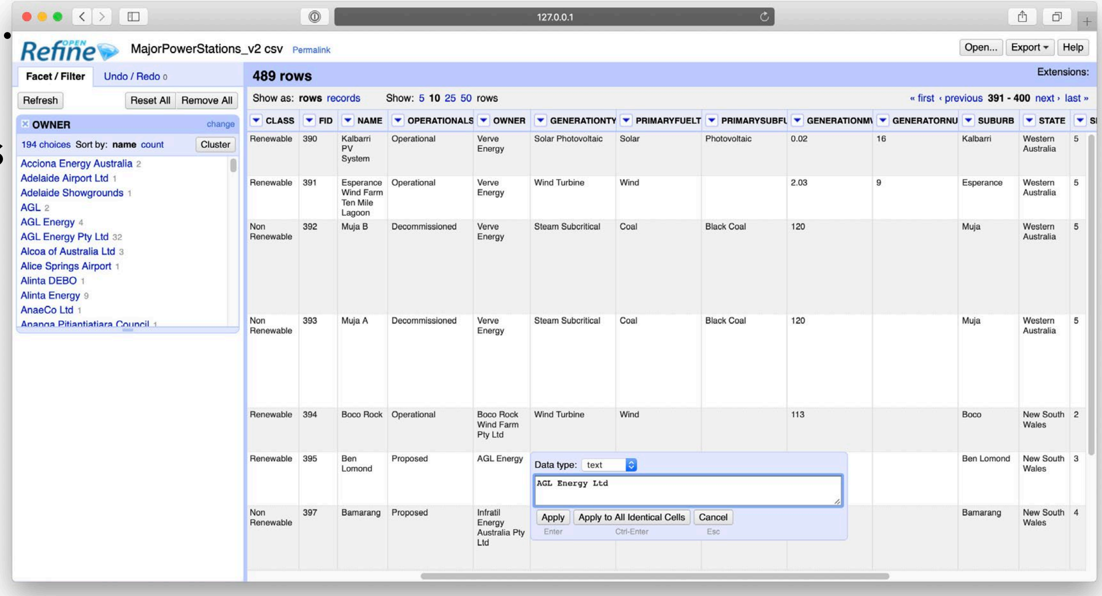
- Real data is often '*dirty*'
- Important to do some data cleaning and transforming first
- Typical steps involved:
  - type and name conversion
  - filtering of missing or inconsistent data
  - unifying semantic data representations
  - matching of entries from different sources
  - Handling of dates and time
- Later also:
  - **<u>Scaling</u>** and **normalization**, and optional dimensionality reduction

Approach 1: Specific Data Cleaning Tools

OpenRefine

A free, open source, powerful tool for working with messy data

- Open Source Example: **Open Refine**
  - Originally developed by Google, but also other commercial tools available
  - Allows to visually inspect and clean data with interactive user-interface
  - More advanced: Reconcile and match different data sets
  - Export to CSV, Excel, HTML, …
- Very helpful, especially for smaller data sets
  - But manual interaction required



Approach 2: Jupyter Notebooks and Python

- Write code with Python and its libraries, to check for, and deal with, dirty data
- Warning: always look at the data first, before running any functions
- Warning: always keep a copy of the original (before cleaning) data
  - Eg with version control system

Read data into Python using csv

- Python **csv** module
  - Reads/writes comma-separated values with escaping to handle cases where comma occurs within a field
    - csv.reader reads rows into arrays
    - csv.DictReader reads rows into ***dictionaries***
- However: Not much support for further data handling or output…
  - E.g. convoluted syntax and type conversions needed
  - E.g. **pprint** module needed for pretty print complex data structure
    - pprint formats a dictionary read by CSV so it's easier to read

```
import csv
import pprint
data = list(csv.DictReader(open('MajorPowerStations.csv')))
pprint.pprint(data[0])
```

Pandas - Python Data Analysis Library

- Open source library providing data import and analysis functionality to Python
- https://pandas.pydata.org/
  - optimised data structures for data analysis
    - Tabular data (DataFrame)
    - Time series data (Series)
    - Matrixes
  - configurable input/output file
  - support for handling missing data, cleaning data, descriptive stats
- API documentation: https://pandas.pydata.org/pandas-docs/stable/reference/index.html

Pandas – Reading data from a CSV file

- Pandas provides several reader functions for various file formats such as, for example, CSV
  - configurable with many options (cf. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html#pandas.read_csv)

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

Pandas – Data Structures

- Two main data structures:
  - **Series** (1-dimensional, labeled, homogenous typed)
  - **DataFrame** (2-dimensional, labeled, (potentially) heterogeneous columns)

- CSV reader imports a dataset as a **DataFrame**
  - Most Pandas functions also produce a DataFrame as output, hence multiple functions can be applied in sequence easily
  - http://pandas.pydata.org/pandas-docs/stable/reference/frame.html

```
data.axes
data.columns
data.dtypes
data['name'].count()
```

Pandas – Missing Data Handling

- Pandas provides various functions for handling missing/wrong data
  - Part of this already included in the input functions (cf. **csv_read()**) where missing values are automatically replaced with NA/NaN
  - Other examples:
    - DataFrame.dropna()   remove rows with missing values
    - DataFrame.fillna()   fill NA/NaN values using a specified method
    - DataFrame.replace()   replace values

```
data2 = data['numGen'].dropna()

data['numGen'].fillna(0, inplace=True)

data['numGen'].replace(to_replace='<Null>', value=0, inplace=True)
```

Pandas: Fix missing values during import

- Some datasets contain placeholders for missing values
  - such as ‘n/a’, ‘--’ or ‘null’
- Best to replace during import to avoid later problems

```
import pandas as pd

missing_values = [“--”,”<Null>”]
data = pd.read_csv(‘MajorPowerStations.csv’, na_values = missing_values)
data.head()
```

Cleaning data: convert to correct types

- The standard Python **csv** module reads everything as string types
- **Pandas** is a bit better, but still will fallback to string if it can't deduce the type from all values in a column
- This will give problems sooner or later with stat functions or plots
- Need to convert as appropriate (e.g., int, float, timestamp)
  - int() creates integer objects, e.g., -1, 101
  - float() creates floating point object, e.g., 3.14, 2.71
  - datetime.strptime() creates datetime objects from strings

Approach 1: Use Pandas

- astype() function on Data series to convert to new types
  - Careful: fails if any entry in the series violates the new type
  - For example: ints do not support NaN
  - Also: does not support special value handling

```
import pandas as pd
data = pd.read_csv('MajorPowerStations.csv')

data['numGenerator'] = data['numGenerator'].astype(int)
data['powerOutput']  = data['powerOutput'].astype(float)
```

Approach 2: Function to convert values in a given column

– We are more flexible by introducing our own *clean()* function

Use “not a number” (*nan*) as default value
numpy knows to ignore for some stats

```python
import numpy as np
DEFAULT_VALUE = np.nan

def clean(data, column_key, convert_function, default_value):
    special_values= {'1 year' : 1.0, '2years' : 2.0, '2 years': 2.0, 'Ten' : 10, 'Half a year': 0.5, '6 months': 0.5, '
    for row in data:
        old_value = row[column_key]
        new_value = default_value
        try:
            if old_value in special_values.keys():
                new_value = special_values[old_value]
            else:
                new_value = convert_function(old_value)
        except (ValueError, TypeError):
            print('Replacing {} with {} in column {}'.format(row[column_key], new_value, column_key))
        row[column_key] = new_value

clean(data, BACKGROUND_YEARS_PROFESSIONAL, float, DEFAULT_VALUE)
clean(data, BACKGROUND_YEARS_PROGRAMMING,  float, DEFAULT_VALUE)
```

A function to convert values in a given column

Define *clean* function that cleans given data

```python
import numpy as np
DEFAULT_VALUE = np.nan

def clean(data, column_key, convert_function, default_value):
    special_values= {'1 year' : 1.0, '2years' : 2.0, '2 years': 2.0, 'Ten' : 10, 'Half a year': 0.5, '6 months': 0.5, '
    for row in data:
        old_value = row[column_key]
        new_value = default_value
        try:
            if old_value in special_values.keys():
                new_value = special_values[old_value]
            else:
                new_value = convert_function(old_value)
        except (ValueError, TypeError):
            print('Replacing {} with {} in column {}'.format(row[column_key], new_value, column_key))
        row[column_key] = new_value

clean(data, BACKGROUND_YEARS_PROFESSIONAL, float, DEFAULT_VALUE)
clean(data, BACKGROUND_YEARS_PROGRAMMING,  float, DEFAULT_VALUE)
```

A function to convert/clean values in a given column

list of known strings
and their numerical equivalent

```
import numpy as np
DEFAULT_VALUE = np.nan

def clean(data, column_key, convert_function, default_value):
    special_values= {'1 year' : 1.0, '2years' : 2.0, '2 years': 2.0, 'Ten' : 10, 'Half a year': 0.5, '6 months': 0.5}
    for row in data:
        old_value = row[column_key]
        new_value = default_value
        try:
            if old_value in special_values.keys():
                new_value = special_values[old_value]
            else:
                new_value = convert_function(old_value)
        except (ValueError, TypeError):
            print('Replacing {} with {} in column {}'.format(row[column_key], new_value, column_key))
        row[column_key] = new_value

clean(data, BACKGROUND_YEARS_PROFESSIONAL, float, DEFAULT_VALUE)
clean(data, BACKGROUND_YEARS_PROGRAMMING,  float, DEFAULT_VALUE)
```

replace known strings
with valid number

A function to convert values in a given column

Get original value from row
Set new value to default

Attempt conversion catching errors

```python
import numpy as np
DEFAULT_VALUE = np.nan

def clean(data, column_key, convert_function, default_value):
    special_values= {'1 year' : 1.0, '2years' : 2.0, '2 years': 2.0, 'Ten' : 10, 'Half a year' : 0.5, '6 months' : 0.5, '
    for row in data:
        old_value = row[column_key]
        new_value = default_value
        try:
            if old_value in special_values.keys():
                new_value = special_values[old_value]
            else:
                new_value = convert_function(old_value)
        except (ValueError, TypeError):
            print('Replacing {} with {} in column {}'.format(row[column_key], new_value, column_key))
        row[column_key] = new_value

clean(data, BACKGROUND_YEARS_PROFESSIONAL, float, DEFAULT_VALUE)
clean(data, BACKGROUND_YEARS_PROGRAMMING,  float, DEFAULT_VALUE)
```

Cleaning float data

```
import numpy as np
DEFAULT_VALUE = np.nan

def clean(data, column_key, convert_function, default_value):
    special_values= {'1 year' : 1.0, '2years' : 2.0, '2 years': 2.0, 'Ten' : 10, 'Half a year': 0.5, '6 months': 0.5}
    for row in data:
        old_value = row[column_key]
        new_value = default_value
        try:
            if old_value in special_values.keys():
                new_value = special_values[old_value]
            else:
                new_value = convert_function(old_value)
        except (ValueError, TypeError):
            print('Replacing {} with {} in column {}'.format(row[column_key], new_value, column_key))
        row[column_key] = new_value

clean(data, BACKGROUND_YEARS_PROFESSIONAL, float, DEFAULT_VALUE)
clean(data, BACKGROUND_YEARS_PROGRAMMING,  float, DEFAULT_VALUE)
```

Call for professional and programming experience columns

Descriptive Statistics with Pandas

Descriptive Statistics with Pandas

– **DataFrame** supports a wide variety of data analysis functions

Sr.No.	Function	Description
1	count()	Number of non-null observations
2	sum()	Sum of values
3	mean()	Mean of Values
4	median()	Median of Values
5	mode()	Mode of values
6	std()	Standard Deviation of the Values
7	min()	Minimum Value
8	max()	Maximum Value
9	abs()	Absolute Value
10	prod()	Product of Values
11	cumsum()	Cumulative Sum
12	cumprod()	Cumulative Product

https://www.tutorialspoint.com/python_pandas/python_pandas_descriptive_statistics.htm

– Function application & GroupBy: groupby(), apply(), applymap()

– http://pandas.pydata.org/pandas-docs/stable/reference/frame.html

Examples of Descriptive Statistics on numerical data

```
import pandas as pd
data = pd.read_csv('MajorPowerStations.csv')

print( data['power'].min() )

print( data['power'].max() )

print( data['power'].mean() )

print( data['power'].median() )

print( data['power'].std() )
```

Examples of Descriptive Statistics: Mode

- Recall that the **mode** is the most frequent value
- Useful for categorial data where mean etc. are not defined

```
import pandas as pd
data = pd.read_csv('MajorPowerStations.csv')

# find most frequent class of power station
print( data['class'].mode() )

# find most frequent owner
print( data['owner'].mode() )
```

Filtering

- You can filter entries in a DataFrame using **loc[]**
  - Allows to specify a Boolean predicate where only those entries are selected in the DataFrame for which the predicate is True
  - Optionally also allows to select specific columns to keep in result

```
import pandas as pd
data = pd.read_csv('MajorPowerStations.csv')

# list of all photovoltaic solar power stations
solarStations = data.loc[ data[`type']=='Solar Photovoltaic' ]

# total power capacity of thermal solar stations
data.loc[ data[`type']=='Solar Thermal', 'power' ].sum()

# list of all large (>100 MW) wind power stations
largeWindParks = data.loc[(data[`type']=='Wind Turbine') & (data[`power']>100)]
```

Frequency distributions using groupby() and size()

– Entries in a Pandas DataFrame can be grouped by a column

```
import pandas as pd

data = pd.read_csv(‘MajorPowerStations.csv’)

classDistr = data.groupby(‘class’).size()
print(classDistr)
```

More descriptive Statistics with numpy

- Another useful Python library is **numpy** ('Numerical Python')
- **Numpy** provides various statistics for numeric data
  - essential tool for multi-dimensional, array-oriented computing
- Median, percentiles, mean, standard deviation, etc
- nan* versions calculate same statistics, ignoring NaN values

- Reference page for numpy statistics:
  http://docs.scipy.org/doc/numpy/reference/routines.statistics.html

Calculating central tendency and dispersion with numpy

Calculate stats for professional and programming experience

```
import numpy as np
for column_key in [BACKGROUND_YEARS_PROFESSIONAL, BACKGROUND_YEARS_PROGRAMMING]:
    v = [row[column_key] for row in data] # grab values
    print(column_key.upper())
    print("* Min..Max: {}..{}".format(np.nanmin(v), np.nanmax(v)))
    print("* Range: {}".format(np.nanmax(v)-np.nanmin(v)))
    print("* Mean: {}".format(np.nanmean(v)))
    print("* Standard deviation: {}".format(np.nanstd(v)))
    print("* Median: {}".format(np.nanmedian(v)))
    q1 = np.nanpercentile(v, 25)
    print("* 25th percentile (Q1): {}".format(q1))
    q3 = np.nanpercentile(v, 75)
    print("* 75th percentile (Q3): {}".format(q3))
    iqr = q3-q1
    print("* IQR: {}".format(iqr))
```

Calculate min/max, range, mean, standard deviation, median, 25/75th percentiles, inter-quartile range

Data Visualisation with Python

Visualising data with Pandas and matplotlib

- Matplotlib provides functionality for creating various plots
- Bar charts, line charts, scatter plots, etc
- Pandas offers some easy-to-use shortcut functions

- Reference page for Pandas plotting:
  https://pandas.pydata.org/pandas-docs/stable/reference/plotting.html
- Reference page for pyplot:
  http://matplotlib.org/api/pyplot_api.html
- Matplotlib Documentation:
  http://matplotlib.org/contents.html

Creating a Bar Chart

for nominal / categorial data

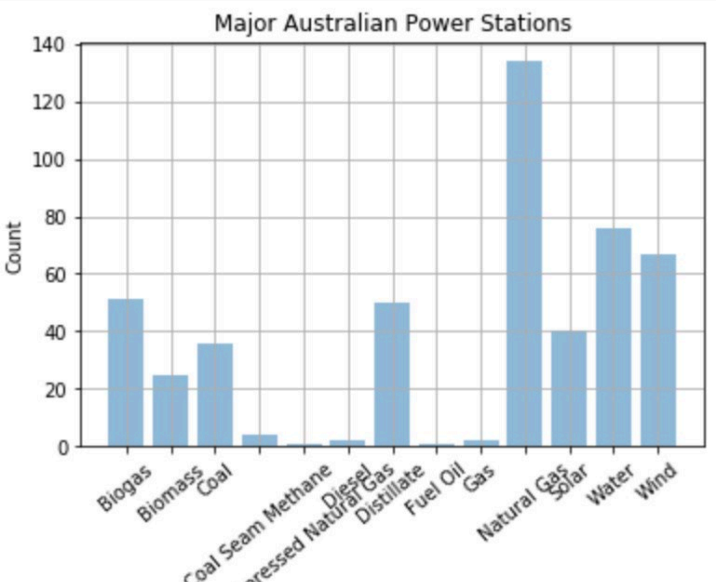
Use groupby() to get frequency distribution
Rename resulting counts as 'numStations'

```
%matplotlib inline
fuelTypeDistr = wrkData.groupby('fueltype').size().reset_index(name='numStations')

# Plot
plt.bar(fuelTypeDistr['fueltype'], fuelTypeDistr['numStations'], alpha=0.5, align='center')
plt.xticks(rotation=40)
plt.title('Major Australian Power Stations')
plt.xlabel('Primary Fuel Type')
plt.ylabel('Count')
plt.grid()
```

Configure plot using matplotlib

Resulting plot ->



Plotting a Histogram

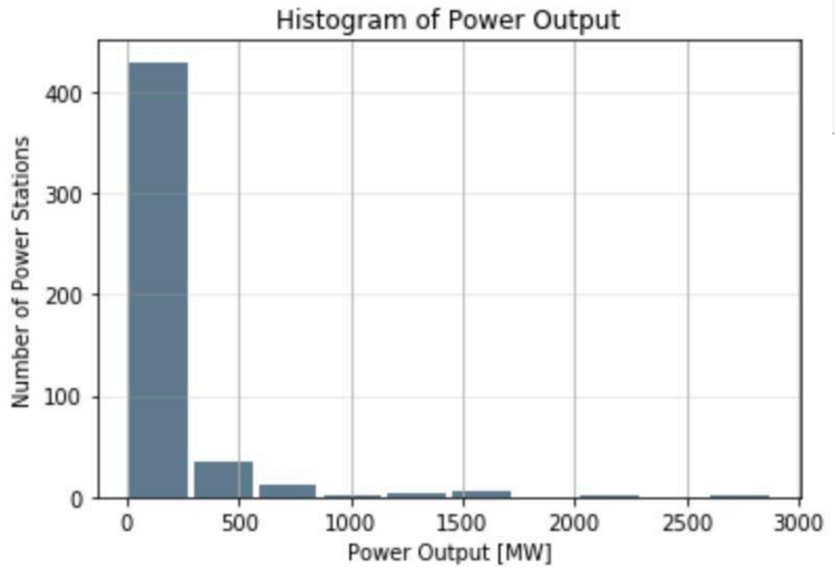
for continuous variables

Create histogram plot with 10 bins of ’power’ values

```
pyExpFreq = wrkData['power'].hist(bins=10, rwidth=0.9, color='#607c8e')
plt.title('Histogram of Power Output')
plt.xlabel('Power Output [MW]')
plt.ylabel('Number of Power Stations')
plt.grid(axis='y', alpha=0.25)
```

Configure plot

Resulting histogram ->



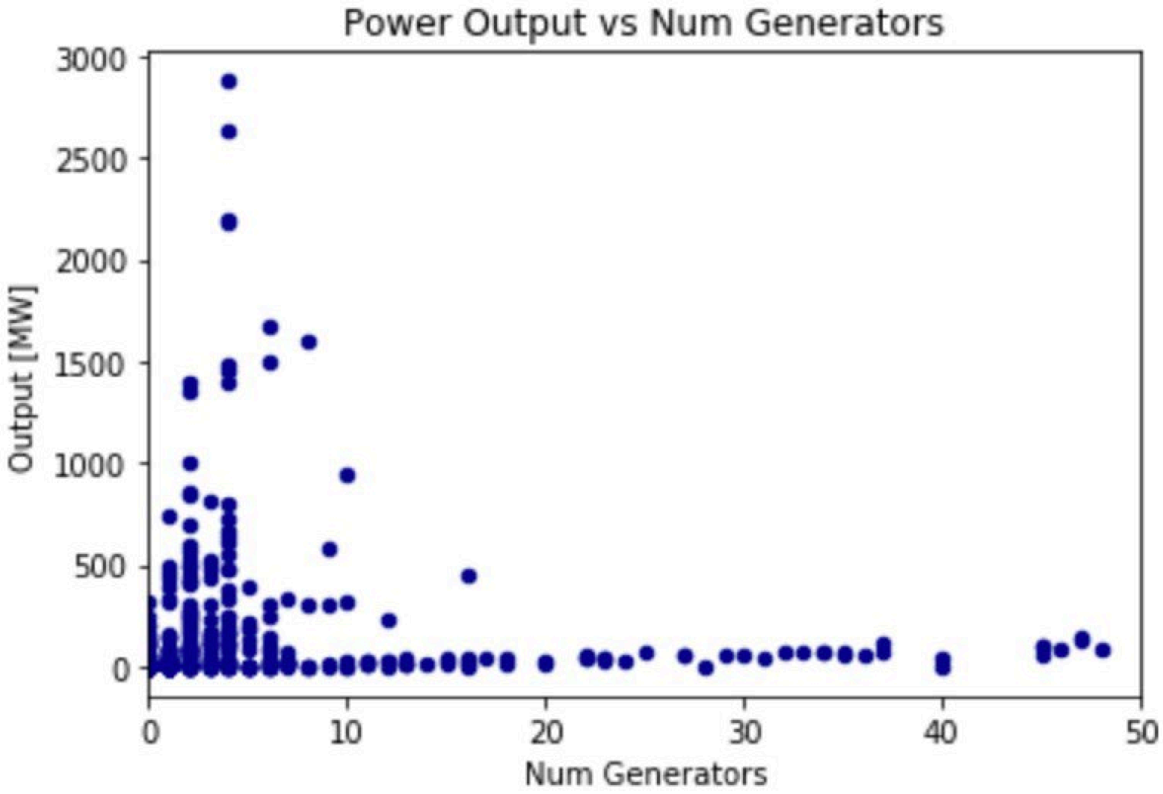
Creating a Scatter Plot

```
%matplotlib inline
import matplotlib.pyplot as plt

fig = plt.figure()
sub = plt.subplot()
wrkData.plot.scatter(x='numGen', y='power', c='DarkBlue', ax=sub)
sub.set_xlim(0,50)
plt.title('Power Output vs Num Generators')
plt.xlabel('Num Generators')
plt.ylabel('Output [MW]')
```

Create scatter plot

Scatter plot comparing Power Output vs. Generator Size



Creating a Scatter Plot with variable coloring/grayscale

```python
# assign colors to some selected fuel types
# the numbers and the order chosen are up-to you.
# we have chosen an order that works well with the color schemes used in the subsequent plots
wrkData['fuelEncoding'] = wrkData['fueltype'].map({
    'Biogas': 1,
    'Wind': 2,
    'Solar': 3,
    'Water': 4,
    'Natural Gas': 5,
    'Coal Seam Methane': 6,
    'Coal': 7
})
```

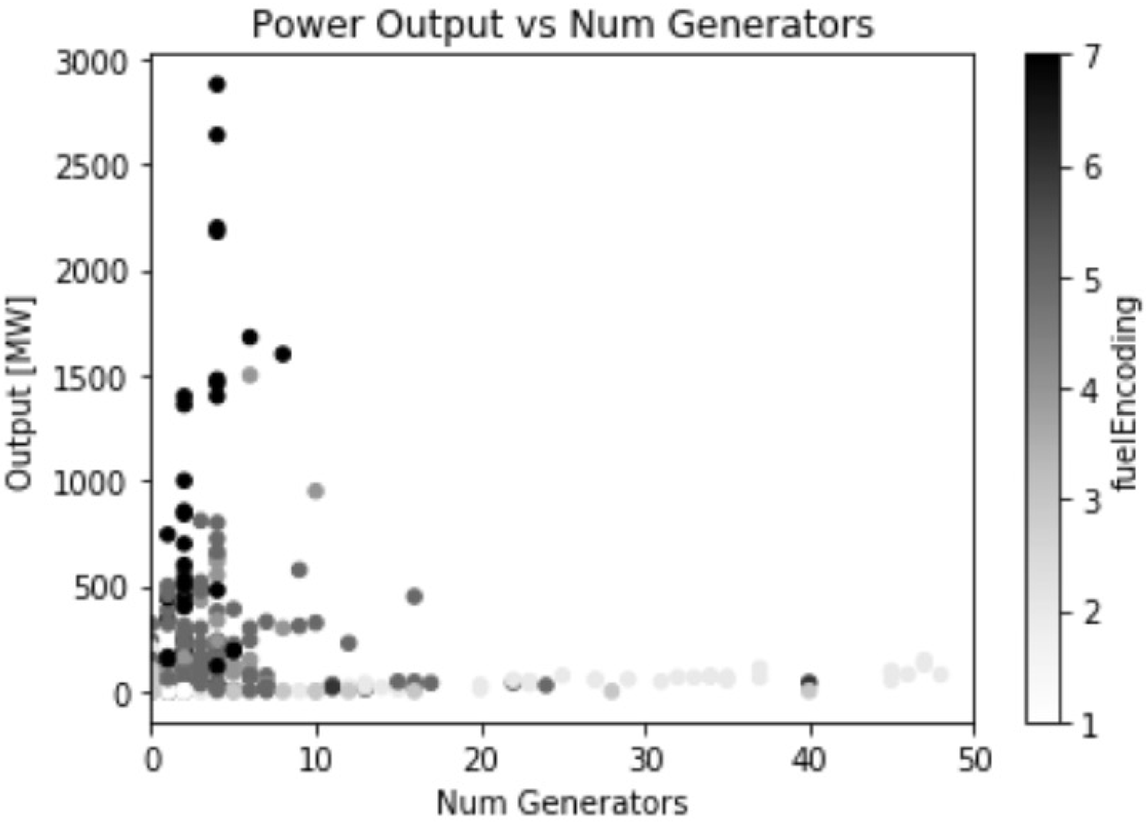
Encode fuel type into numerical values [1..7]

```python
# Now we can use this encoding column to color our plot
%matplotlib inline

fig = plt.figure()
sub = plt.subplot()
wrkData.plot.scatter(x='numGen', y='power', c='fuelEncoding', ax=sub)
sub.set_xlim(0,50)
plt.title('Power Output vs Num Generators')
plt.xlabel('Num Generators')
plt.ylabel('Output [MW]')
```

Color by encoding values

Scatter plot2 comparing Power Output vs. Generator Size



Coal

Coal Seam

Natural Gas

Water

Solar

Wind

Biogas

Creating a Scatter Plot with specific colormap

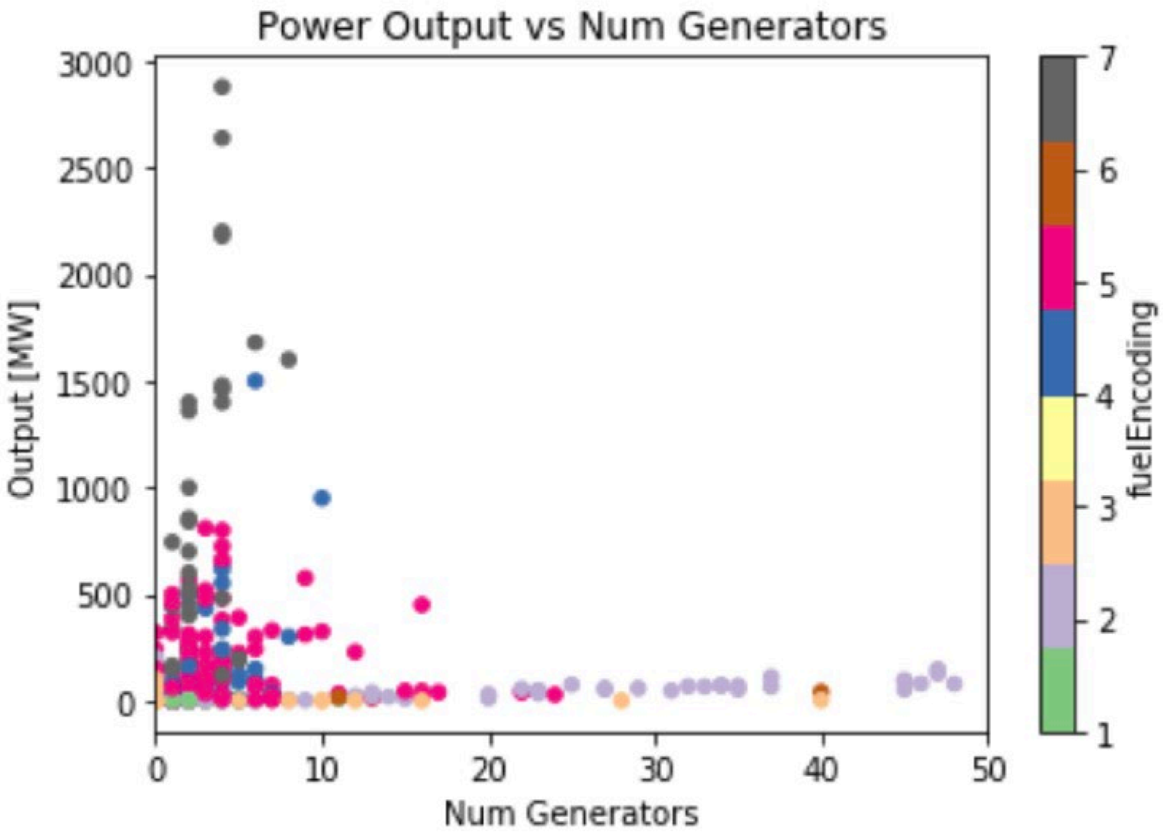
Using same encoding than before…

```
# the same plot as before, but using a more vivid color scheme (colormap='Accent')
# (for available colormaps from matplotlib, see https://matplotlib.org/3.1.0/tutorials/colors/colo
%matplotlib inline

fig = plt.figure()
sub = plt.subplot()
wrkData.plot.scatter(x='numGen',y='power',c='fuelEncoding',colormap='Accent',ax=sub)
sub.set_xlim(0,50)
plt.title('Power Output vs Num Generators')
plt.xlabel('Num Generators')
plt.ylabel('Output [MW]')
```

Color using matplotlib's 'Accent' colormap

Scatter plot2 comparing Power Output vs. Generator Size



Coal

Coal Seam

Natural Gas

Water

Solar

Wind

Biogas

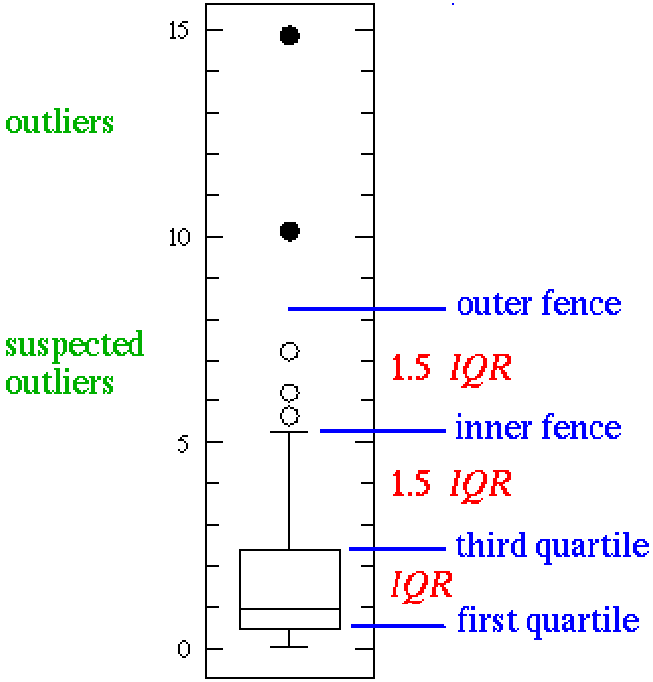
Box Plots and Correlation

Using box plots to compare distributions

For quantitative data

- Mean and stdev are not informative when data is skewed
- **Box plots** summarise data based on 5 numbers:
  - Lower inner fence – Q1–1.5*IQR
  - First quartile (Q1) – equivalent to 25^{th} percentile
  - Median (Q2) – equivalent to 50^{th} percentile
  - Third quartile (Q3) – equivalent to 75^{th} percentile
  - Upper inner fence – Q3+1.5*IQR
- Values outside fences are outliers
- Sometimes include outer fences at 3*IQR

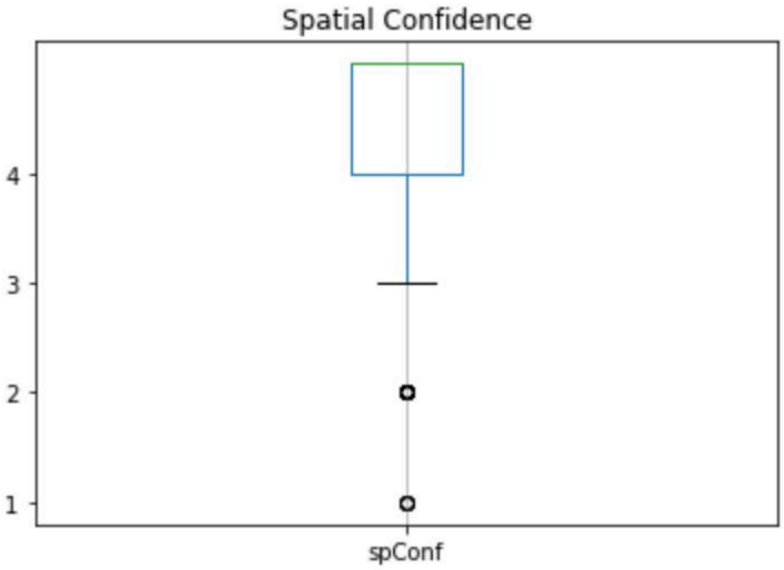
Box Plots illustrated



A box plot of the ‘spatial confidence’ values

```
%matplotlib inline

plt.yticks(np.arange(1, 5, 1.0))
fig = wrkData.boxplot(['spConf']).set_title('Spatial Confidence')
plt.grid(axis='y', alpha=0) # disable grid lines
```



spatial confidence on a likert scale of 1 to 5; 5 representing highest confidence in location data of power station

Using correlation statistics to measure dependence

- Scipy includes various correlation statistics
  - Parametric test
    - Pearson's *r* for two normally distributed variables
  - Non-Parametric test
    - Spearman's *rho* for ratio data, ordinal data, etc (rank-order correlation)
    - Kendall's *tau* for ordinal variables

- List of various scipy statistics including correlation coefficients: http://docs.scipy.org/doc/scipy-0.14.0/reference/stats.html

Calculating correlation

Since correlation is paired, grab values where both variables are defined

```
from scipy import stats
# only keep rows where both professional and programming experience are defined
prof, prog = [], []
for row in data:
    if row[BACKGROUND_YEARS_PROFESSIONAL] is np.nan:
        continue # ignore rows with no value for professional experience
    elif row[BACKGROUND_YEARS_PROGRAMMING] is np.nan:
        continue # ignore rows with no value for programming experience
    else:
        prof.append(row[BACKGROUND_YEARS_PROFESSIONAL])
        prog.append(row[BACKGROUND_YEARS_PROGRAMMING])
print("Pearson (r, p): {}".format(stats.pearsonr(prof, prog)))
print(stats.spearmanr(prof, prog))
```

Calculate Person's r and Spearman's rho

Review

Some Tips and Tricks

- Data cleaning important for any meaningful analysis
- Spreadsheet software is good for quick interactive analysis
  Need programmatic analysis for bigger/complex data
- Careful about which types of data allow what kind of measures & viz.
- Measures of central tendancy (e.g., mean) are not sufficient
  Always explore and communicate spread as well (e.g., stdev)
- Good visualisations help convey distributions and relationships
  - Label all plots and diagrams with readable and visible fonts
  - Use same axis bounds when comparing plots
  - Use meaningful axis bounds to convey effect size
    (50-55 on a 100 point scale over-sells small differences)
  - Design so comparison/effect is clear, include description of axes

Notes

- Python a good example of a scripting language for DS
- programmatic approaches allow for more powerful / flexible data preparation and analysis,
  - and more control on the visualisations
- Many useful support libraries available in the Python ecosystem
  - Pandas, numpy, scipy, matplotlib
- Hands-on practice is vital

Activities this week

- Tutorial:
  - Introduction to Pandas with Jupyter Notebooks
  - Data Loading and Cleaning with Python

Additional reading (not examinable)

- Pandas Documentation
  https://pandas.pydata.org/
- matplotlib API reference
  http://matplotlib.org/api/pyplot_api.html
- NumPy and SciPy documentation
  http://docs.scipy.org/doc/
- W. McKinney, Python for Data Analysis with Python (2nd ed) (O'Reilly)
  - Available online in Usyd library