Basic Data Manipulation with Pandas

Following are the common 4 steps to start a data analysis project.

- Data Exploration
- Data Filtering and Sorting
- Data Cleaning
- Data Transformation

In this exercise, we will learn how to use Pandas in these 4 steps.

Import `numpy` and `pandas`.

In [1]:

```
import numpy as np
import pandas as pd
```

1. Data Exploration

The loaded data may be too large to examine all of them. We check out following aspects of the data to understand it better.

- Number of rows and records
- Data types of columns
- View data samples
- Basic statistics of each columns
- Basic plotting

Load csv file `temperature-monthly-mean-daily-maximum.csv` in `data` folder.

In [91]:

```
df = pd.read_csv('data/temperature-monthly-mean-daily-maximum.csv')
```

Size of Data

The `dataframe.shape` attribute returns dimensions of the data.

In [92]:

```
df.shape
```

Out[92]:

```
(461, 2)
```

Dataframe Info

The `dataframe.info()` function is used to get a summary of the dataframe.

- Each column's name, data type and record counts, thus it contains any null data.
- Index type
- Memory usage

In [94]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 461 entries, 0 to 460
Data columns (total 2 columns):
month                  461 non-null object
temp_mean_daily_max    461 non-null float64
dtypes: float64(1), object(1)
memory usage: 7.3+ KB
```

Sample Data

The `head()` and `tail()` function returns first and last few rows of the data.

In [5]:

```
1 df.head()
```

Out[5]:

	month	temp_mean_daily_max
0	1982-01	29.8
1	1982-02	32.3
2	1982-03	31.7
3	1982-04	31.4
4	1982-05	31.7

In [6]:

```
1 df.tail(3)
```

Out[6]:

	month	temp_mean_daily_max
458	2020-03	32.9
459	2020-04	33.0
460	2020-05	32.2

Statistical Information

The `describe()` function provides some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values.

In [7]:

```
1 df.describe()
```

Out[7]:

	temp_mean_daily_max
count	461.000000
mean	31.525163
std	0.874877
min	28.800000
25%	31.000000
50%	31.500000
75%	32.100000
max	34.400000

Pandas provides many statistical functions.

- The `median()` function return the median of the values for the requested axis.
- The `mode()` function returns the most frequent values for the requested axis.

In [9]:

```
1 df.iloc[:,1].median()
```

Out[9]:

```
31.5
```

In [10]:

```
1 df['temp_mean_daily_max'].mode()
2 # df['temp_mean_daily_max'].value_counts()   # This function only available for pd.Seri
```

Out[10]:

```
0    31.7
dtype: float64
```

Rename columns

To make it easier for future exploration, we can rename some columns.

In [95]:

```
1 df.rename(columns={'temp_mean_daily_max':'temp_max'}, inplace=True)
```

Basic Plotting

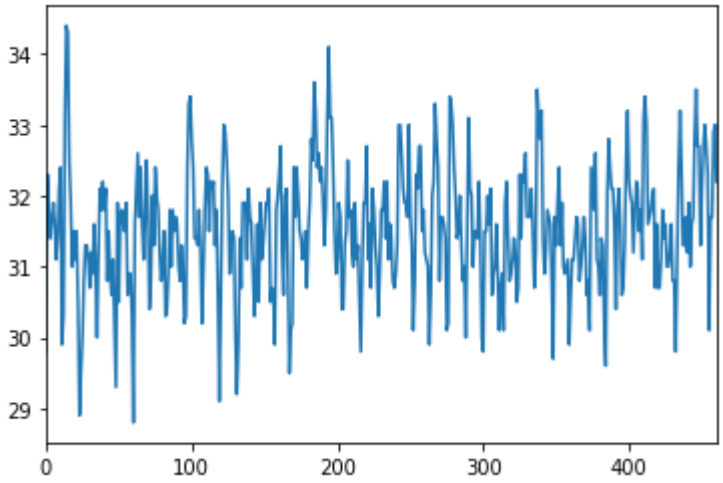
Line Graph

In [96]:

```
1 df['temp_max'].plot()
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cbad7aec18>
```



Histogram

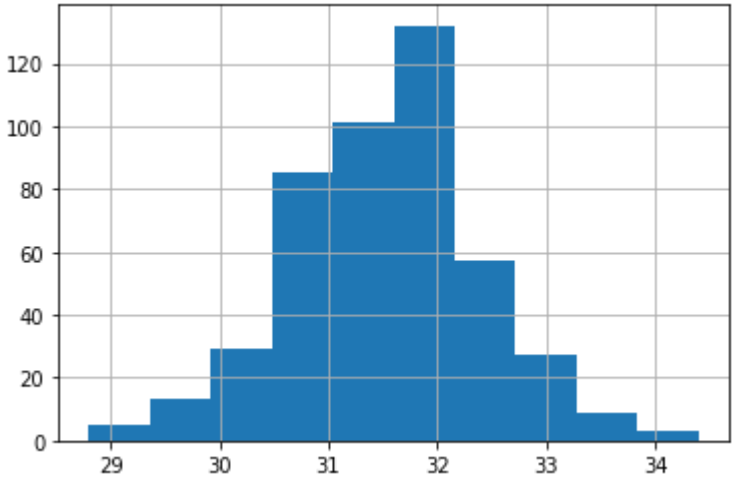
Histogram represents the frequency of occurrence within fixed intervals of values.

In [97]:

```
1 df['temp_max'].hist()
```

Out[97]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cbad9566a0>
```



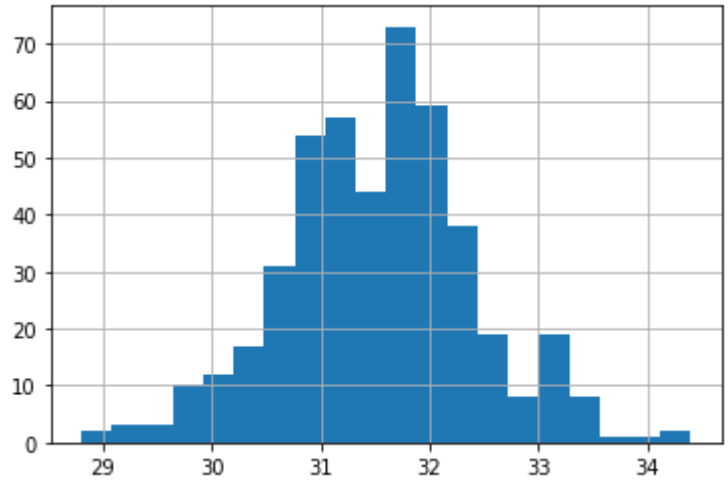
The parameter `bins` can be used to control the granularity of the charts.

In [13]:

```
1 df['temp_max'].hist(bins=20)
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cbad536e80>
```



2. Data Indexing, Filtering and Sorting

- Selecting row(s) and column(s) using index operator, `loc[]` and `iloc[]`
- Boolean filtering
- Assigning values with indexing
- Sorting

In [14]:

```
1 df = pd.read_csv('data/class1_test1.tsv', delimiter='\t')
```

In [15]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 4 columns):
name       9 non-null object
english    9 non-null int64
maths      9 non-null int64
science    9 non-null int64
dtypes: int64(3), object(1)
memory usage: 416.0+ bytes
```

In [16]:

```
1 df.head()
```

Out[16]:

	name	english	maths	science
0	Aaron	70	46	47
1	Adrian	72	40	95
2	Alby	49	65	64
3	Abner	86	40	96
4	Benett	50	98	69

Indexing and Data Selection

Indexing means selecting particular rows and columns of data from a DataFrame.

- Indexing can be used to select individual row, column or item.
- Indexing can also be used to perform **Subset Selection**.

Pandas uses indexers `[ ]`, `.loc[]` and `.iloc[ ]`.

- `Dataframe[]` : Used for columns selection. Also known as indexing operator.
- `Dataframe.loc[]` : Used for rows selection using **labels**.
- `Dataframe.iloc[]` : Used for rows selection using **positions**.

Select Columns using Indexing Operator `[]`

Select a single column using its label.

In [17]:

```
1 df['name']
```

Out[17]:

```
0     Aaron
1    Adrian
2      Alby
3     Abner
4    Benett
5     Brion
6    Collin
7     Cyril
8     Dylan
Name: name, dtype: object
```

Since single column selection returns a Series, additional `[]` can be added to select rows from resulting series.

In [18]:

```
df['name'][:4]
# df['name'][[4,2,0]]
```

Out[18]:

```
0     Aaron
1    Adrian
2      Alby
3     Abner
Name: name, dtype: object
```

To confirm that `dataframe[]` selects columns by labels, try on a dataframe whose column label is a integer value.

In [19]:

```
t = pd.DataFrame([['Alan'],['Bob'], ['Chris']], columns=[111], index=[2,3,2])
t
# t[111]
# t[111][2]
```

Out[19]:

	111
2	Alan
3	Bob
2	Chris

Slicing Rows using Indexing Operator `[]`

Indexing Operator can also be used to select multiple rows using their positions.

Select first 2 rows.

In [20]:

```
df[:2]
```

Out[20]:

	name	english	maths	science
0	Aaron	70	46	47
1	Adrian	72	40	95

Select Rows using `.loc[]`

This function selects rows and/or columns **by their labels**.

- `.loc[rows]` selects multiple rows of all columns

- `.loc[rows, cols]` select certain rows and columns
- `.loc[:, cols]` select multiple columns of all rows

Modify row index to alphabets.

In [21]:

```
idx = [name[:2] for name in df['name']]
idx
```

Out[21]:

```
['Aa', 'Ad', 'Al', 'Ab', 'Be', 'Br', 'Co', 'Cy', 'Dy']
```

Update index (row labels) of the dataframe.

In [22]:

```
df.index = idx
```

Select 2nd row.

In [23]:

```
df.loc['Ad']
```

Out[23]:

```
name       Adrian
english        72
maths          40
science        95
Name: Ad, dtype: object
```

Select row 0 and 2.

In [24]:

```
df.loc[['Aa', 'Al']]
```

Out[24]:

	name	english	maths	science
Aa	Aaron	70	46	47
Al	Alby	49	65	64

Select first row 0 and 2 in `name` and `english` columns.

In [25]:

```
df.loc[['Aa', 'Al'], ['name', 'english']]
```

Out[25]:

	name	english
Aa	Aaron	70
Al	Alby	49

Select all rows in `name` and `english` columns.

In [26]:

```
df.loc[:, ['name', 'english']]
```

Out[26]:

	name	english
Aa	Aaron	70
Ad	Adrian	72
Al	Alby	49
Ab	Abner	86
Be	Benett	50
Br	Brion	81
Co	Collin	45
Cy	Cyril	60
Dy	Dylan	72

Select Rows using `.iloc[]`

This function selects rows and/or columns **by their positions**.

- `.iloc[rows]` selects multiple rows of all columns
- `.iloc[rows, cols]` select certain rows and columns
- `.iloc[:, cols]` select multiple columns of all rows

Select 2nd row, i.e. row with name = `Adrian` .

In [27]:

```
df.iloc[1]
```

Out[27]:

```
name       Adrian
english        72
maths          40
science        95
Name: Ad, dtype: object
```

Select multiple rows.

In [28]:

```
df.iloc[[0,1,2]]
# df.iloc[:3]
```

Out[28]:

	name	english	maths	science
Aa	Aaron	70	46	47
Ad	Adrian	72	40	95
Al	Alby	49	65	64

Select all rows from columns `name` and `maths` .

In [29]:

```
df.iloc[:, [0,2]]
```

Out[29]:

	name	maths
Aa	Aaron	46
Ad	Adrian	40
Al	Alby	65
Ab	Abner	40
Be	Benett	98
Br	Brion	92
Co	Collin	83
Cy	Cyril	46
Dy	Dylan	90

Filtering using Boolean Expression

When series is evaluated in a boolean expression, it returns a Series of boolean values.

In [30]:

```
x = df['maths'] >= 50
print(type(x))
print(x)
```

```
<class 'pandas.core.series.Series'>
Aa    False
Ad    False
Al     True
Ab    False
Be     True
Br     True
Co     True
Cy    False
Dy     True
Name: maths, dtype: bool
```

Boolean values can be used to filter a dataframe.

- For example, to find out who has passed maths test.

In [31]:

```
df[x]
```

Out[31]:

	name	english	maths	science
Al	Alby	49	65	64
Be	Benett	50	98	69
Br	Brion	81	92	95
Co	Collin	45	83	45
Dy	Dylan	72	90	74

Exercise:

- List all who have passed all 3 tests.

In [32]:

```
y = (df['english']>=50) & (df['maths']>=50) & (df['science']>=50)
y
```

Out[32]:

```
Aa    False
Ad    False
Al    False
Ab    False
Be     True
Br     True
Co    False
Cy    False
Dy     True
dtype: bool
```

In [33]:

```
y2 = (df[['english', 'maths', 'science']]>=50).all(axis=1)
y == y2
```

Out[33]:

```
Aa    True
Ad    True
Al    True
Ab    True
Be    True
Br    True
Co    True
Cy    True
Dy    True
dtype: bool
```

In [34]:

```
df[y2]
```

Out[34]:

	name	english	maths	science
Be	Benett	50	98	69
Br	Brion	81	92	95
Dy	Dylan	72	90	74

Updating Values

Selection in dataframe returns a **view** to the original data. Thus any changes to values in the view will affects original data directly.

In [35]:

```
df1 = df.copy()
```

In [36]:

```
failed = df['english']<50
passed = ~failed
passed
```

Out[36]:

```
Aa     True
Ad     True
Al    False
Ab     True
Be     True
Br     True
Co    False
Cy     True
Dy     True
Name: english, dtype: bool
```

In [37]:

```
df1['english'][failed] = 'failed'
df1['english'][passed] = 'passed'
df1
```

```
C:\Users\zqi2\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykernel_
launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (http://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy)
  """Entry point for launching an IPython kernel.
```

Out[37]:

	name	english	maths	science
Aa	Aaron	passed	46	47
Ad	Adrian	passed	40	95
Al	Alby	failed	65	64
Ab	Abner	passed	40	96
Be	Benett	passed	98	69
Br	Brion	passed	92	95
Co	Collin	failed	83	45
Cy	Cyril	passed	46	74
Dy	Dylan	passed	90	74

Sorting

- sort_index()
- sort_values()

In [38]:

```
df1 = df.copy()
df1.set_index(['name'], inplace=True)
df1
```

Out[38]:

name	english	maths	science
Aaron	70	46	47
Adrian	72	40	95
Alby	49	65	64
Abner	86	40	96
Benett	50	98	69
Brion	81	92	95
Collin	45	83	45
Cyril	60	46	74
Dylan	72	90	74

Sort by Row Index

In [39]:

```
df1.sort_index(ascending=False, inplace=True)
df1
```

Out[39]:

name	english	maths	science
Dylan	72	90	74
Cyril	60	46	74
Collin	45	83	45
Brion	81	92	95
Benett	50	98	69
Alby	49	65	64
Adrian	72	40	95
Abner	86	40	96
Aaron	70	46	47

Sort by Column Index

In [40]:

```
df1.sort_index(axis=1, ascending=False, inplace=True)
df1
```

Out[40]:

name	science	maths	english
Dylan	74	90	72
Cyril	74	46	60
Collin	45	83	45
Brion	95	92	81
Benett	69	98	50
Alby	64	65	49
Adrian	95	40	72
Abner	96	40	86
Aaron	47	46	70

Sort by Value(s)

In [41]:

```
df1.sort_values(by='english')
```

Out[41]:

name	science	maths	english
Collin	45	83	45
Alby	64	65	49
Benett	69	98	50
Cyril	74	46	60
Aaron	47	46	70
Dylan	74	90	72
Adrian	95	40	72
Brion	95	92	81
Abner	96	40	86

Sorting by values can also be done on multiple columns.

In [42]:

```
df1.sort_values(by=['english', 'maths'], inplace=True)
df1
```

Out[42]:

name	science	maths	english
Collin	45	83	45
Alby	64	65	49
Benett	69	98	50
Cyril	74	46	60
Aaron	47	46	70
Adrian	95	40	72
Dylan	74	90	72
Brion	95	92	81
Abner	96	40	86

3. Data Cleaning

- Missing Data
- Outliers
- Duplicates
- Type Conversion

Load tsv file `class1_test1_cleaning.tsv` in `data` folder.

In [43]:

```
df = pd.read_csv('data/class1_test1_cleaning.tsv', delimiter='\t')
```

Missing Data

By examine returned values of `info()` , not all columns have same number of data.

That indicates that there are some missing data in the dataframe.

- Both `maths` and `science` columns have some missing data
- The `religion` column seems to have 0 data

In [44]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 5 columns):
name        12 non-null object
english     11 non-null float64
maths       10 non-null float64
science     10 non-null float64
religion    0 non-null float64
dtypes: float64(4), object(1)
memory usage: 608.0+ bytes
```

The `isnull()` returns `True` if the value is `NaN`. To find out which columns contain `NaN` value, use `any()` function.

In [45]:

```
1 df.isnull().any()
```

Out[45]:

```
name        False
english      True
maths        True
science      True
religion     True
dtype: bool
```

The `religion` column is empty.

In [46]:

```
1 df.isnull().all()
```

Out[46]:

```
name        False
english     False
maths       False
science     False
religion     True
dtype: bool
```

Handling Missing Data

Drop Column(s)

Drop `religion` column since it does not contain any data.

In [47]:

```
df1 = df.copy()
df1.drop(columns='religion', inplace=True)
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 4 columns):
name       12 non-null object
english    11 non-null float64
maths      10 non-null float64
science    10 non-null float64
dtypes: float64(3), object(1)
memory usage: 512.0+ bytes
```

In [48]:

```
df1
```

Out[48]:

	name	english	maths	science
0	Aaron	70.0	46.0	47.0
1	Adrian	72.0	40.0	95.0
2	Alby	49.0	65.0	NaN
3	Abner	86.0	40.0	96.0
4	Benett	50.0	98.0	69.0
5	Brion	81.0	NaN	95.0
6	Collin	45.0	83.0	45.0
7	Cyril	160.0	46.0	74.0
8	Dylan	72.0	90.0	74.0
9	Aaron	70.0	46.0	47.0
10	Dylan	72.0	90.0	74.0
11	Eva	NaN	NaN	NaN

Drop Row(s) with `NaN`

The `dropna()` function drops any rows contains null values.

- To drop any column with null value(s), supply a parameter `axis=1` .
- To drop any row or column with >= n number of null values, supply a parameter `thresh=n` .

In [49]:

```
1 df1.isnull().any(axis=1)
```

Out[49]:

```
0     False
1     False
2      True
3     False
4     False
5      True
6     False
7     False
8     False
9     False
10    False
11     True
dtype: bool
```

In [50]:

```
1 df1.dropna()
```

Out[50]:

	name	english	maths	science
0	Aaron	70.0	46.0	47.0
1	Adrian	72.0	40.0	95.0
3	Abner	86.0	40.0	96.0
4	Benett	50.0	98.0	69.0
6	Collin	45.0	83.0	45.0
7	Cyril	160.0	46.0	74.0
8	Dylan	72.0	90.0	74.0
9	Aaron	70.0	46.0	47.0
10	Dylan	72.0	90.0	74.0

In [51]:

```
1 df1.dropna(thresh=3)
```

Out[51]:

	name	english	maths	science
0	Aaron	70.0	46.0	47.0
1	Adrian	72.0	40.0	95.0
2	Alby	49.0	65.0	NaN
3	Abner	86.0	40.0	96.0
4	Benett	50.0	98.0	69.0
5	Brion	81.0	NaN	95.0
6	Collin	45.0	83.0	45.0
7	Cyril	160.0	46.0	74.0
8	Dylan	72.0	90.0	74.0
9	Aaron	70.0	46.0	47.0
10	Dylan	72.0	90.0	74.0

Replace `NaN` with a Value

You can replace `NaN` value with a value using `fillna()` function.

- Depends on application, sometimes it is logical to replace a missing value with mean, median or mode value of that column.

In [52]:

```
df1.fillna(0)
```

Out[52]:

	name	english	maths	science
0	Aaron	70.0	46.0	47.0
1	Adrian	72.0	40.0	95.0
2	Alby	49.0	65.0	0.0
3	Abner	86.0	40.0	96.0
4	Benett	50.0	98.0	69.0
5	Brion	81.0	0.0	95.0
6	Collin	45.0	83.0	45.0
7	Cyril	160.0	46.0	74.0
8	Dylan	72.0	90.0	74.0
9	Aaron	70.0	46.0	47.0
10	Dylan	72.0	90.0	74.0
11	Eva	0.0	0.0	0.0

Forward and Backward Filling

For missing value in some measurements or time series, it is logical to use previous or next value to replace missing values.

Set the parameter `method` of `fillna()` function to `ffill` to perform forward-filling, or `bfill` to perform backward-filling.

In [53]:

```
reading = pd.DataFrame([1,2,3,4,np.NaN,5,6,7], columns=['val'])
reading
```

Out[53]:

	val
0	1.0
1	2.0
2	3.0
3	4.0
4	NaN
5	5.0
6	6.0
7	7.0

In [54]:

```
1 reading.fillna(method='ffill')
```

Out[54]:

	val
0	1.0
1	2.0
2	3.0
3	4.0
4	4.0
5	5.0
6	6.0
7	7.0

In [55]:

```
1 reading.fillna(method='bfill')
```

Out[55]:

	val
0	1.0
1	2.0
2	3.0
3	4.0
4	5.0
5	5.0
6	6.0
7	7.0

Outliers

To detect outliers,

- Check basic statistic data of the dataframe
- Use basic plotting to detect outlier records.

In [56]:

```
1 df1.describe()
```

Out[56]:

	english	maths	science
count	11.000000	10.000000	10.000000
mean	75.181818	64.400000	71.600000
std	31.079956	23.552306	20.089798
min	45.000000	40.000000	45.000000
25%	60.000000	46.000000	52.500000
50%	72.000000	55.500000	74.000000
75%	76.500000	88.250000	89.750000
max	160.000000	98.000000	96.000000

Scatter Plots

A scatter chart shows the relationship between two different variables.

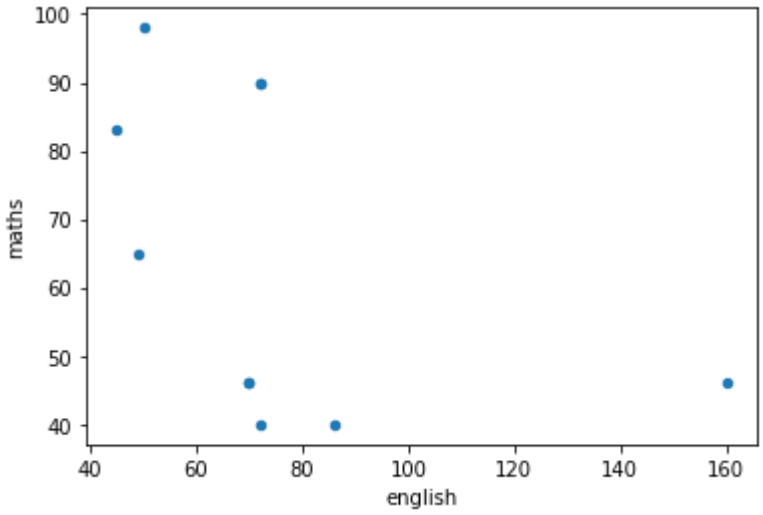
- It can reveal the distribution trends.
- It is used to highlight similarities in a data set.
- It is useful for understanding the distribution of your data.
- It is commonly used to find outliers.

In [57]:

```
1 df1.plot.scatter(x='english', y='maths')
```

Out[57]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cbad615b70>
```



Box Plots

Box Plot is the visual representation of groups of numerical data through their quartiles.

- Boxplot summarizes a sample data using 25th, 50th and 75th percentiles.
- It captures the summary of the data efficiently with a simple box and whiskers.
- It allows us to compare easily across groups.
- It is commonly used to detect the outlier in data set.

A box plot consist of 5 things.

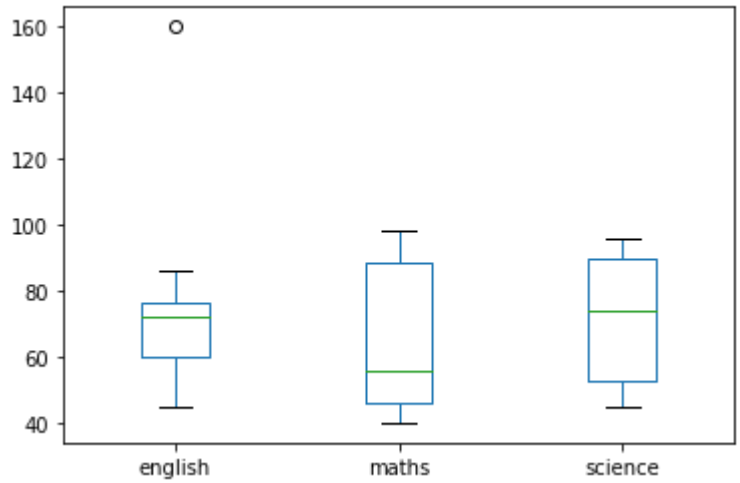
- Minimum
- First Quartile or 25%
- Median (Second Quartile) or 50%
- Third Quartile or 75%
- Maximum

In [58]:

```
1 df1.plot.box()
```

Out[58]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cbad69eac8>
```



In [59]:

```
df1['english']
```

Out[59]:

```
0      70.0
1      72.0
2      49.0
3      86.0
4      50.0
5      81.0
6      45.0
7     160.0
8      72.0
9      70.0
10     72.0
11      NaN
Name: english, dtype: float64
```

Cap outliers' value.

In [60]:

```
df1['english'][df1['english']>100] = 100
df1
```

```
C:\Users\zqi2\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykernel_
launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (http://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy)
  """Entry point for launching an IPython kernel.
```

Out[60]:

	name	english	maths	science
0	Aaron	70.0	46.0	47.0
1	Adrian	72.0	40.0	95.0
2	Alby	49.0	65.0	NaN
3	Abner	86.0	40.0	96.0
4	Benett	50.0	98.0	69.0
5	Brion	81.0	NaN	95.0
6	Collin	45.0	83.0	45.0
7	Cyril	100.0	46.0	74.0
8	Dylan	72.0	90.0	74.0
9	Aaron	70.0	46.0	47.0
10	Dylan	72.0	90.0	74.0
11	Eva	NaN	NaN	NaN

Duplicate Values

In [61]:

```
1 df1.duplicated()
```

Out[61]:

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9      True
10     True
11    False
dtype: bool
```

In [62]:

```
1 df1[df1.duplicated()]
```

Out[62]:

	name	english	maths	science
9	Aaron	70.0	46.0	47.0
10	Dylan	72.0	90.0	74.0

Drop duplicates in dataframe directly.

In [63]:

```
1 df1.drop_duplicates(inplace=True)
```

In [64]:

```
1 df1.duplicated().any()
```

Out[64]:

```
False
```

Type Conversion

Some marks columns contains null values. In Pandas, only `float` and `object` types can contain null values. Thus to convert marks columns to `int` , we need to fix missing data.

In [65]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10 entries, 0 to 11
Data columns (total 4 columns):
name       10 non-null object
english    9 non-null float64
maths      8 non-null float64
science    8 non-null float64
dtypes: float64(3), object(1)
memory usage: 400.0+ bytes
```

In [66]:

```
df2 = df1.dropna()
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7 entries, 0 to 8
Data columns (total 4 columns):
name       7 non-null object
english    7 non-null float64
maths      7 non-null float64
science    7 non-null float64
dtypes: float64(3), object(1)
memory usage: 280.0+ bytes
```

In [67]:

```
df2[['english', 'maths','science']] = df2[['english', 'maths','science']].astype(int)
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7 entries, 0 to 8
Data columns (total 4 columns):
name       7 non-null object
english    7 non-null int32
maths      7 non-null int32
science    7 non-null int32
dtypes: int32(3), object(1)
memory usage: 196.0+ bytes

C:\Users\zqi2\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\cor
e\frame.py:3509: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (http://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy)
  self[k1] = value[k2]
```

Update Index

In [68]:

```
1 df1.head()
```

Out[68]:

	name	english	maths	science
0	Aaron	70.0	46.0	47.0
1	Adrian	72.0	40.0	95.0
2	Alby	49.0	65.0	NaN
3	Abner	86.0	40.0	96.0
4	Benett	50.0	98.0	69.0

In [69]:

```
1 df2 = df1.set_index('name')
2 df2.head()
```

Out[69]:

	english	maths	science
name			
Aaron	70.0	46.0	47.0
Adrian	72.0	40.0	95.0
Alby	49.0	65.0	NaN
Abner	86.0	40.0	96.0
Benett	50.0	98.0	69.0

In [70]:

```
1 idx = [name[:2] for name in df2.index]
2 idx
```

Out[70]:

```
['Aa', 'Ad', 'Al', 'Ab', 'Be', 'Br', 'Co', 'Cy', 'Dy', 'Ev']
```

In [71]:

```
df2['id'] = idx
df2.head()
```

Out[71]:

name	english	maths	science	id
Aaron	70.0	46.0	47.0	Aa
Adrian	72.0	40.0	95.0	Ad
Alby	49.0	65.0	NaN	Al
Abner	86.0	40.0	96.0	Ab
Benett	50.0	98.0	69.0	Be

In [72]:

```
df2.reset_index(inplace=True)
df2
```

Out[72]:

	name	english	maths	science	id
0	Aaron	70.0	46.0	47.0	Aa
1	Adrian	72.0	40.0	95.0	Ad
2	Alby	49.0	65.0	NaN	Al
3	Abner	86.0	40.0	96.0	Ab
4	Benett	50.0	98.0	69.0	Be
5	Brion	81.0	NaN	95.0	Br
6	Collin	45.0	83.0	45.0	Co
7	Cyril	100.0	46.0	74.0	Cy
8	Dylan	72.0	90.0	74.0	Dy
9	Eva	NaN	NaN	NaN	Ev

In [73]:

```
df2.set_index('id', inplace=True)
df2.head()
```

Out[73]:

id	name	english	maths	science
Aa	Aaron	70.0	46.0	47.0
Ad	Adrian	72.0	40.0	95.0
Al	Alby	49.0	65.0	NaN
Ab	Abner	86.0	40.0	96.0
Be	Benett	50.0	98.0	69.0

4. Basic Data Transformation

- Maths Operations
- Function Applications

In [74]:

```
df = pd.DataFrame(np.ones([2,3]), columns=['a','b', 'c'])
df
```

Out[74]:

	a	b	c
0	1.0	1.0	1.0
1	1.0	1.0	1.0

Maths Operations with Scalar Value

You can apply maths operation to all items in the dataframe.

In [75]:

```
df = df * 2
df
```

Out[75]:

	a	b	c
0	2.0	2.0	2.0
1	2.0	2.0	2.0

In [76]:

```
df.iloc[0] = 3
df
```

Out[76]:

	a	b	c
0	3.0	3.0	3.0
1	2.0	2.0	2.0

Subtract Same Value from Columns

In [77]:

```
m_col = df.mean()
m_col
```

Out[77]:

```
a    2.5
b    2.5
c    2.5
dtype: float64
```

In [78]:

```
df - m_col
```

Out[78]:

	a	b	c
0	0.5	0.5	0.5
1	-0.5	-0.5	-0.5

Subtract Same Value from Rows

In [79]:

```
m_row = df.mean(axis=1)
m_row
```

Out[79]:

```
0    3.0
1    2.0
dtype: float64
```

In [80]:

```
df.sub(m_row*2, axis=0)
```

Out[80]:

	a	b	c
0	-3.0	-3.0	-3.0
1	-2.0	-2.0	-2.0

Operations between DataFrames

In [81]:

```
df2 = pd.DataFrame(np.ones([3,2]), columns=['c', 'd'])
df2
```

Out[81]:

	c	d
0	1.0	1.0
1	1.0	1.0
2	1.0	1.0

In [82]:

```
df + df2
```

Out[82]:

	a	b	c	d
0	NaN	NaN	4.0	NaN
1	NaN	NaN	3.0	NaN
2	NaN	NaN	NaN	NaN

Function Applications

- `apply()` to apply a function to column or row (with `axis=0`)
- `applymap()` to apply a function to every cell

In [83]:

```
df
```

Out[83]:

	a	b	c
0	3.0	3.0	3.0
1	2.0	2.0	2.0

In [84]:

```
df.apply(lambda col: col.mean())
```

Out[84]:

```
a    2.5
b    2.5
c    2.5
dtype: float64
```

In [85]:

```
df.apply(lambda col: col.mean(), axis=1)
```

Out[85]:

```
0    3.0
1    2.0
dtype: float64
```

In [86]:

```
df.applymap(lambda x: x+.1)
```

Out[86]:

	a	b	c
0	3.1	3.1	3.1
1	2.1	2.1	2.1