Practical Data Science: Data Summarisation: Descriptive Statistics and Visualisation

Dr. Yongli Ren(yongli.ren@rmit.edu.au)

Computer Science & IT

School of Science

Outline

- Part 1: Overview
- Part 2: Descriptive Statistics and Visualisation
- Part 3: Assignment 1

Practical Data Science – COSC2670

PART 1: OVERVIEW

Data Summarisation

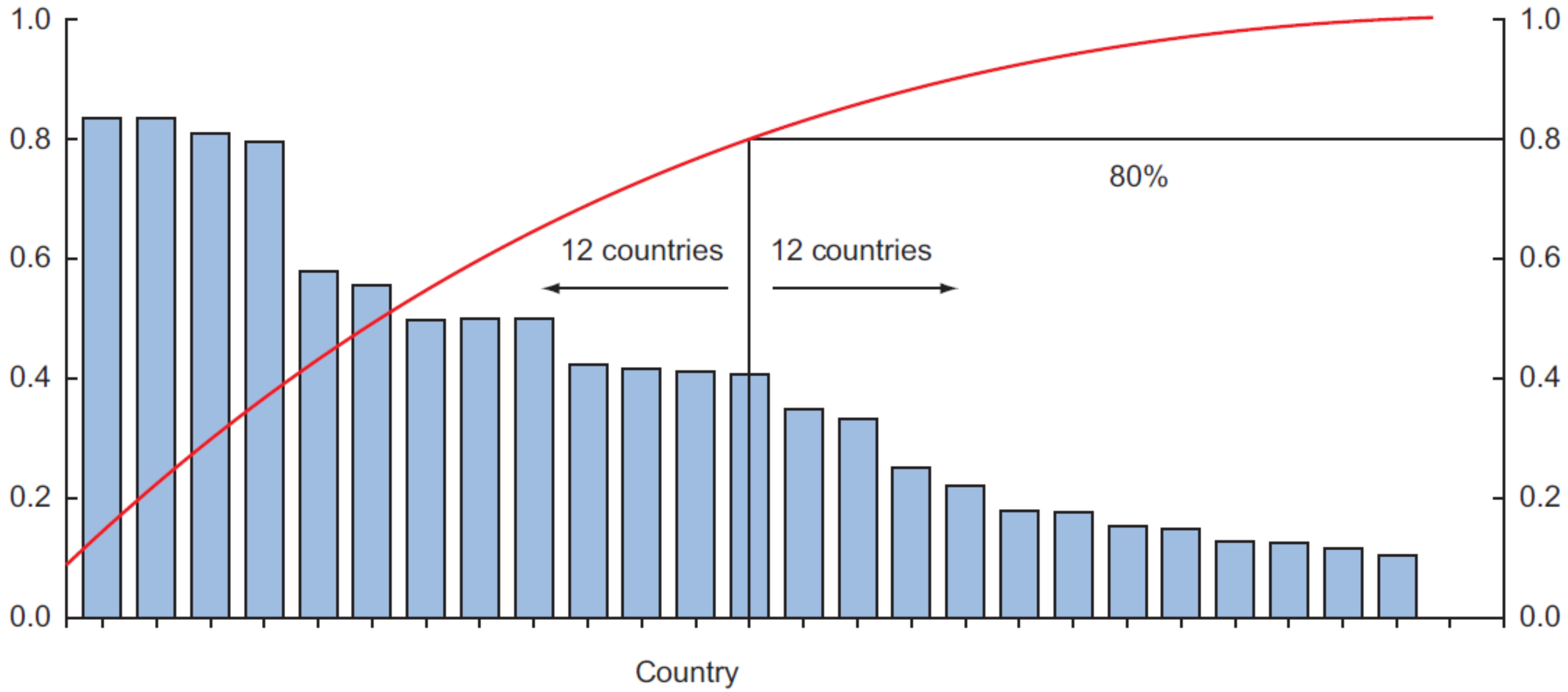
- During exploratory data analysis,
  - You take a deep dive into the data
  - Information becomes much easier to grasp when shown in a picture,
    - therefore you mainly use graphical techniques
      - to gain an understanding of
        - each feature and
        - the interactions between features.
  - This phase is about exploring data, so
    - keeping your *mind open* and *your eyes peeled*.
- The goal isn't to cleanse the data,
  - but it's common that you'll still discover anomalies you missed before, forcing you to take a step back and fix them.

Data Summarisation

- The visualization techniques you use in this phase range from
  - Simple graphs
    - E.g. bar graph, pie graph, line graph, histogram
  - Complex graphs
    - E.g. distribution graph, boxplot, scatter plot, cumulative distribution
- Sometimes it's useful to compose a composite graph from simple graphs to get even more insights into the data.

Data Summarisation

- Pareto Diagram
  - The values (Bar Chart) + Cumulative Distribution

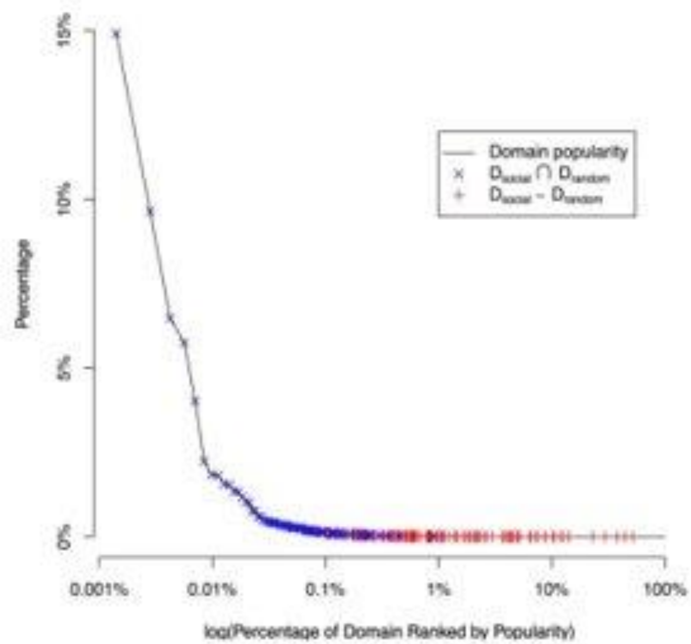


A Pareto diagram is a combination of the values and a cumulative distribution. It's easy to see from this diagram that the first 50% of the countries contain slightly less than 80% of the total amount. If this graph represented customer buying power and we sell expensive products, we probably don't need to spend our marketing budget in every country; we could start with the first 50%.

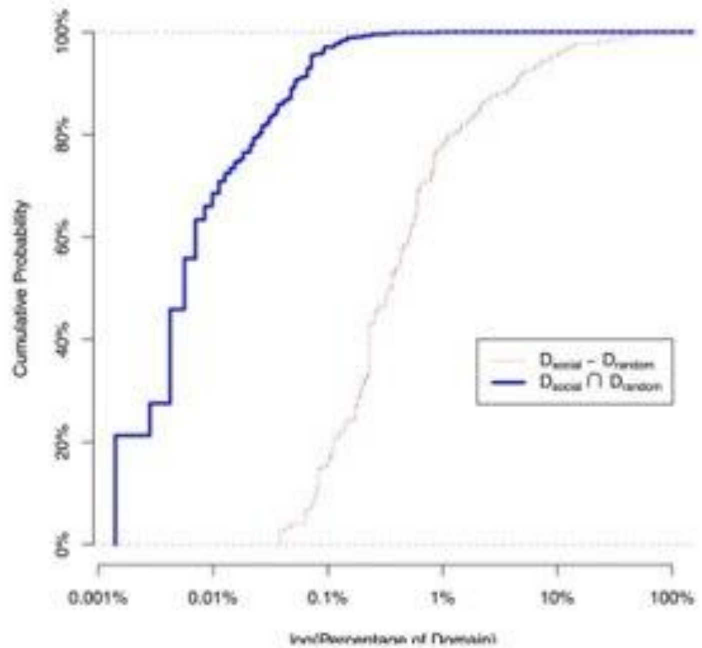
Data Summarisation

- Graphs can also be animated or interactive to make it easier to explore
  - This can also be more fun.
- An example of an interactive bubble graph
  - Across U.S. Companies, Tax Rates Vary Greatly
  - http://www.nytimes.com/interactive/2013/05/25/sunday-review/corporate-taxes.html?_r=0
  - Mike Bostock has interactive examples of almost any type of graph.
  - It's worth spending time on his website, though most of his examples are more useful for data presentation than data exploration.
- An example of animated and interactive visualisation
  - A shopping mall project
- We will focus on
  - How to draw the basic and complex graphs, and
  - how to use them to explore the data.

Data Summarisation



(a) the log plot of domain popularity and the distribution of $D_{social} \cap D_{random}$ and $D_{social} - D_{random}$ over it.



(b) the empirical CDFs of $D_{social} \cap D_{random}$ and $D_{social} - D_{random}$

Figure 6. The domain popularity and the relationship between D_{social} and D_{random}

Thus, $D_{social} \cap D_{random}$ reflects the domains that are commonly accessed by an indoor user regardless of whether they are accompanied or not, and $D_{social} - D_{random}$ reflects the domains that are shared among accompanying users but not non-accompanying users. Finally, we obtain $|D_{social}| = 208$, $|D_{random}| = 88$, $|D_{social} \cap D_{random}| = 70$ and $|D_{social} - D_{random}| = 138$.

Data Summarisation
- Would you survive the Titanic?

- The sinking of Titanic is one of the most infamous shipwrecks in history.
  - On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew.
- This sensational tragedy shocked the international community and led to better safety regulations for ships.
- One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew.
- Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others,
  - such as
    - women,
    - children,
    - and the upper-class.

https://www.kaggle.com/wiki/Leakage

Python Data Manipulation

- Before surviving Titanic, we need to learn some basic pre-processing routines
  - With the purposes to make it feasible for the next data science step:
    - Data Exploration

- Data Selection:
  - What if the source data document contains an *index* column?
  - How do you properly import it with pandas?
  - And then, can we actively exploit it to make our job simpler?
- For example, our Titanic dataset contains an index column: *PassengerID*
  - This is just a counter and not a feature.

Python Data Manipulation

- When trying to load the file in the classic way, you'll find yourself in a situation where you have *PassengerID* as a feature (or a column).

```
import pandas as pd
titanic_filename = 'titanic.csv'
titanic = pd.read_csv(titanic_filename,sep=',',decimal='.',header=0)
```

	PassengerID	PClass	Age	Sex	Survived
0	1	1st	29.00	female	1
1	2	1st	2.00	female	0
2	3	1st	30.00	male	0
3	4	1st	25.00	female	0
4	5	1st	0.92	male	1

- Nothing is practically incorrect
- But an *index* should not be used by mistake as a feature.
- So it is better to keep it separated.
- *If it is used during the learning phase of your model, you may possibly incur a case of "leakage",*
  - *which is one of the major sources of error in machine learning.*

Python Data Manipulation - Leakage

- One concrete example we've seen occurred in a *prostate cancer* dataset.
- Hidden among hundreds of variables in the training data was a variable named PROSSURG.
  - It turned out this represented whether the patient had received prostate surgery, an incredibly predictive but out-of-scope value.
- The resulting model was highly predictive of whether the patient had prostate cancer
  - but was useless for making predictions on new patients.

https://www.kaggle.com/wiki/Leakage

Python Data Manipulation

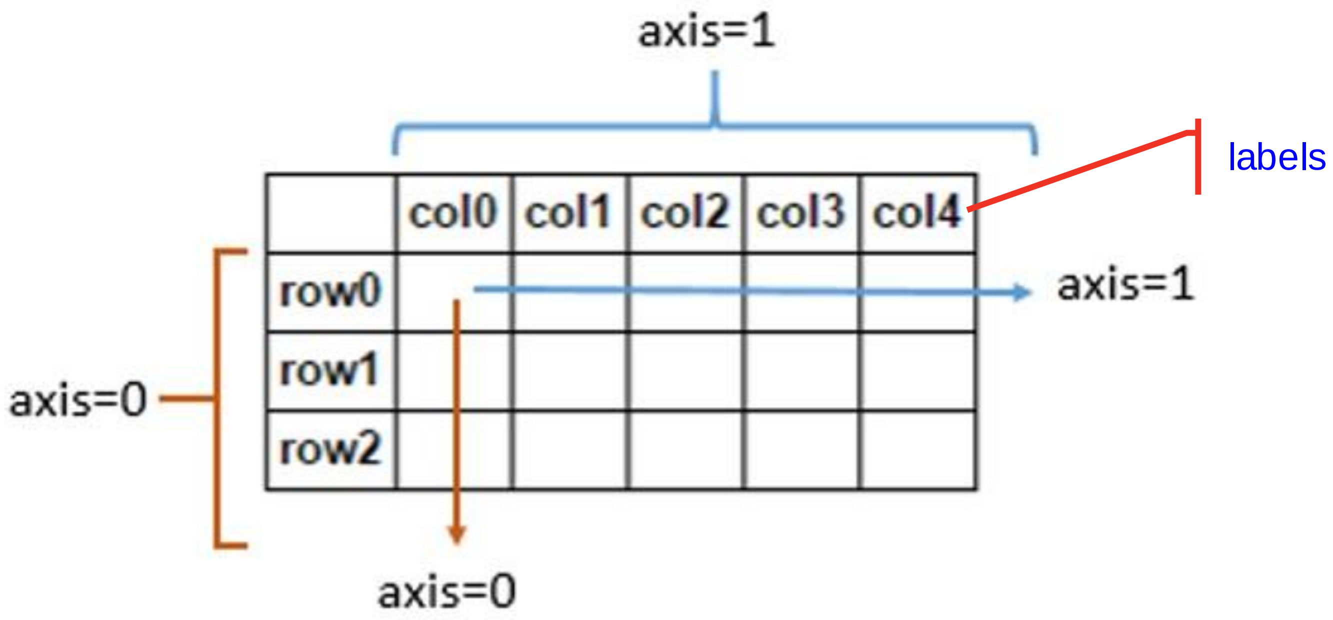
- In fact, if the *index* is a random number, no harm will be done to your model's usefulness.
- However, if the index contains progressive, temporal, or even informative elements
  - (for example, certain numeric ranges may be used for positive outcomes, and others for the negative ones),
  - you might incorporate leaked information into the model
    - then it will be impossible to replicate results when using your model on fresh data.
- Therefore, while loading such a dataset, we might want to specify that *PassengerID* is the index column.
- Since the index *PassengerID* is the first column, we can give the following command:

```
titanic = pd.read_csv('Titanic.csv', index_col=0)
```

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

Python Data Manipulation

- To access the value of a cell, we need to understand a special data structure: *DataFrame*
  - *Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labelled axes.*



http://stackoverflow.com/questions/25773245/ambiguity-in-pandas-dataframe-numpy-array-axis-definition

Python Data Manipulation

- These are a few ways to access the value of a cell.
- Let's list them one by one.
  - First, you can simply *specify the column then the line* (by using its index) you are interested in.
  - To extract the *Age* of the *fourth* line (indexed with *PassengerID*=4), you can give the following command:
    - titanic['Age'][4]

```
              PClass    Age     Sex  Survived
PassengerID
1                1st  29.00  female         1
2                1st   2.00  female         0
3                1st  30.00    male         0
4                1st  25.00  female         0
5                1st   0.92    male         1
```

Python Data Manipulation

- Do this operation carefully since it's not a matrix and you might be tempted to first input the row and then the column.
- Remember that it's actually a pandas *DataFrame*, and
- the [ ] operator works first on columns and then on the element of the resulting pandas Series.
  - Age_s = titanic['Age']
  - Age_s[4]

```
            PClass    Age     Sex  Survived
PassengerID
1              1st  29.00  female         1
2              1st   2.00  female         0
3              1st  30.00    male         0
4              1st  25.00  female         0
5              1st   0.92    male         1
```

Python Data Manipulation

- To have something similar to the preceding method of accessing data, you can use the .loc() method:
  - titanic.loc[4, 'Age']
  - where you should *first specify the index and then the columns* you're interested in.
- This solution is equivalent to the one provided by the .iloc() method.
  - .iloc() works with positions only,
    - titanic.ix[4, 1]

Python Data Manipulation

- If you need to apply a function to a limited section of rows,
  - you can create a ***mask***.
    - *A mask is a series of Boolean values (that is, True or False) that tells whether the line is selected or not.*
- Select data by masks
  - mask_survived = titanic['Survived'] == 1

PassengerID	PClass	Age	Sex	Survived
1	1st	29.00	female	1
2	1st	2.00	female	0
3	1st	30.00	male	0
4	1st	25.00	female	0
5	1st	0.92	male	1

mask_survived

```
True
False
False
False
True
```

Python Data Manipulation

- In the preceding simple example, we can immediately see which observations are *True* and which are not (*False*), and which fit the selection query.
- Now, we want to check the 'Age' of those survived.
  - titanic.loc[mask_survived,'Age'].

```
             PClass    Age     Sex  Survived
PassengerID
1               1st  29.00  female         1
2               1st   2.00  female         0
3               1st  30.00    male         0
4               1st  25.00  female         0
5               1st   0.92    male         1
```

mask_survived

```
 True
False
False
False
 True
```

```
PassengerID
1      29.00
5       0.92
```

Python Data Manipulation

- If you want to see some statistics about each feature,
  - you can group each column accordingly
  - titanic.groupby(['Survived']).mean()

- If you need to sort the observations using a function,
  - you can use the .sort() method, as follows:
  - titanic.sort_index(by='Age').head(10)

Practical Data Science – COSC2670

PART 2: DESCRIPTIVE STATISTICS AND VISUALISATION

Descriptive Statistics

- *Descriptive statistics are statistics that quantitatively describe or summarize features of a collection of information.* [ ]
- Some measures that are commonly used to describe a data set are
  - measures of *central tendency* and
  - measures of *variability* or *dispersion*.
- Measures of central tendency include
  - *The mean*, *median* and *mode*;
- Measures of variability include
  - the *standard deviation* (or *variance*),
  - the minimum and maximum values of the variables,
  - *kurtosis* and
  - *skewness*.

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

Descriptive Statistics
- Basic Statistics

- Count
- Mean
- Std
- Min
- Max
- Median
- 25^{th} percentile
- 50^{th} percentile
- 75^{th} percentile
- …

```
titanic.describe()
```

Descriptive Statistics - Basic Graphs

- Pie Chart
- Bar Graph
- Line Graph
- Histogram Graph
- Distribution Graph
- Scatter Plot
  - Of two features
  - Scatter matrix of all pairs of features
  - Hexagonal binning plots
- Boxplot
  - Boxplot of different features
  - Boxplot of one feature across groups
- *All graphs should be complete and informative in itself.*

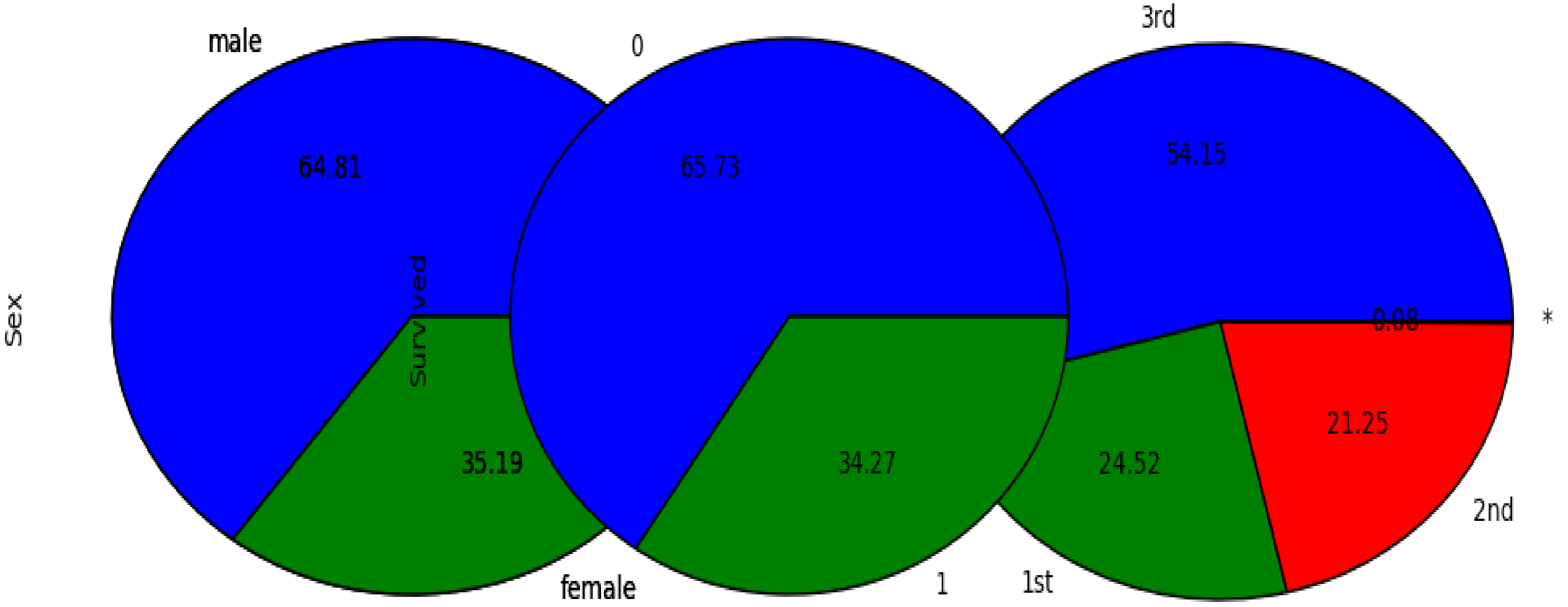
Pie Chart

- A pie chart (or a circle chart) is a circular statistical graphic
  - which is divided into slices to illustrate numerical proportion.
- In a pie chart, the arc length of each slice
  - (and consequently its central angle and area),
  - is proportional to the quantity it represents.

Pie Chart

All graphs should be complete and informative in itself.

- titanic['Sex'].value_counts().plot(kind='pie',autopct='%.2f')
- titanic['PClass'].value_counts().plot(kind='pie',autopct='%.2f')
- titanic['Survived'].dropna().value_counts().plot(kind='pie',autopct='%.2f')

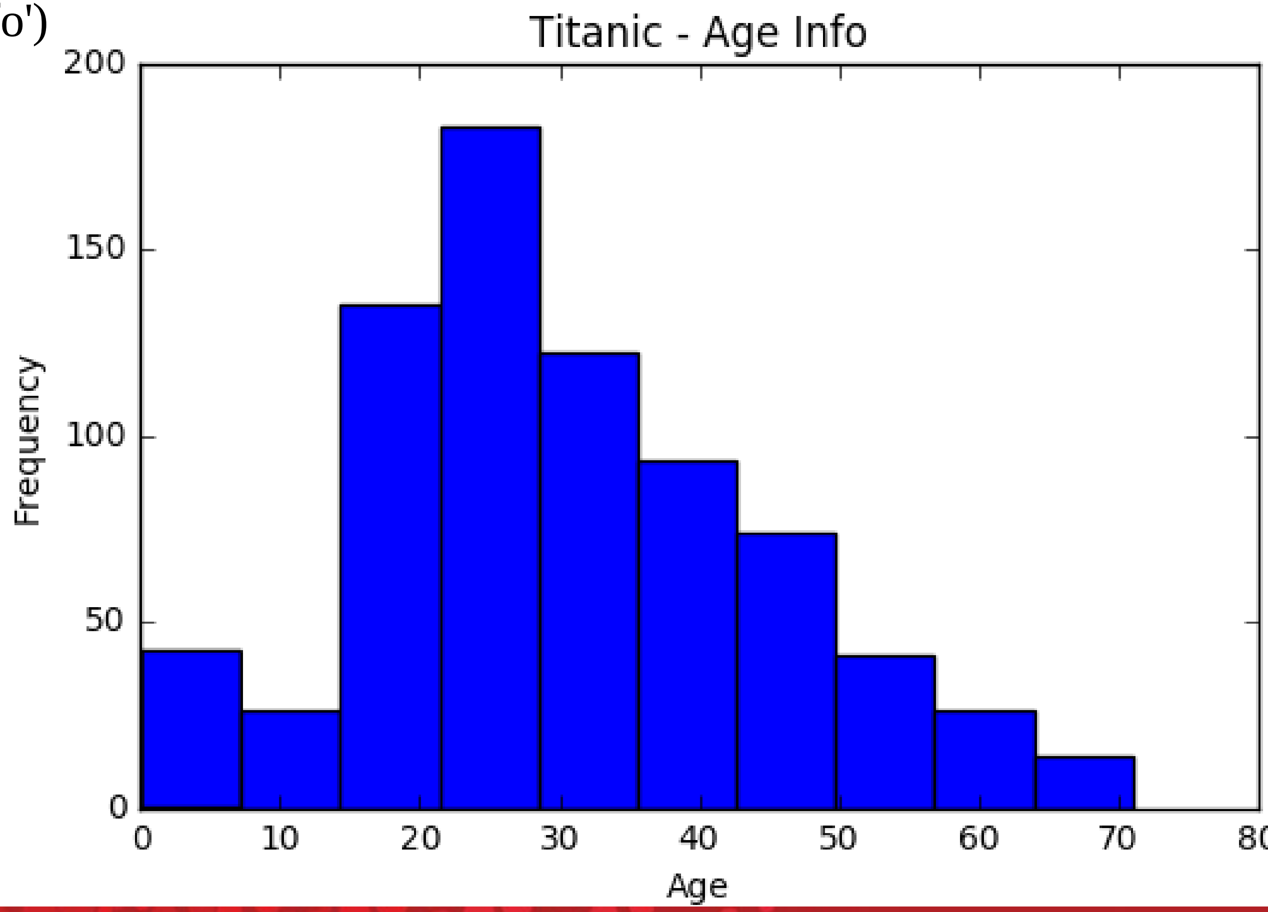


Histograms

All graphs should be complete and informative in itself.

- Histograms can effectively represent the distribution of a variable.

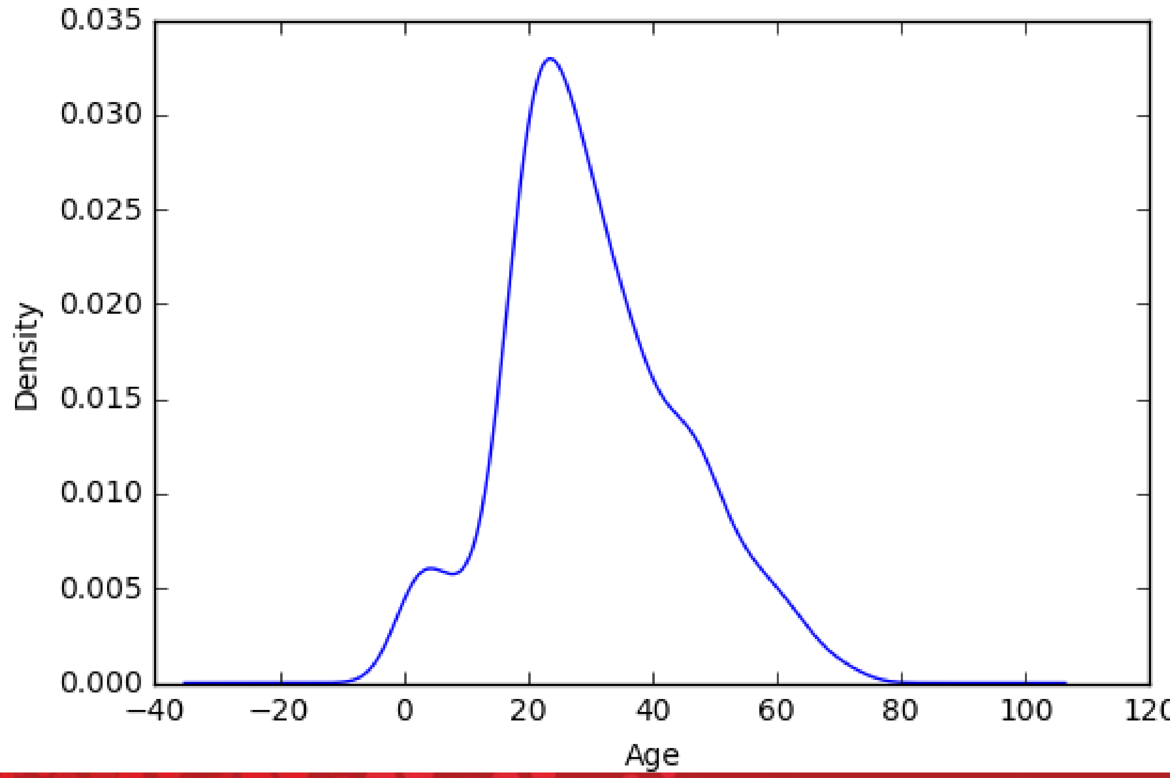
```
titanic['Age'].plot(kind='hist', bins = 10)
plt.title('Titanic - Age Info')
plt.xlabel('Age')
```



Density

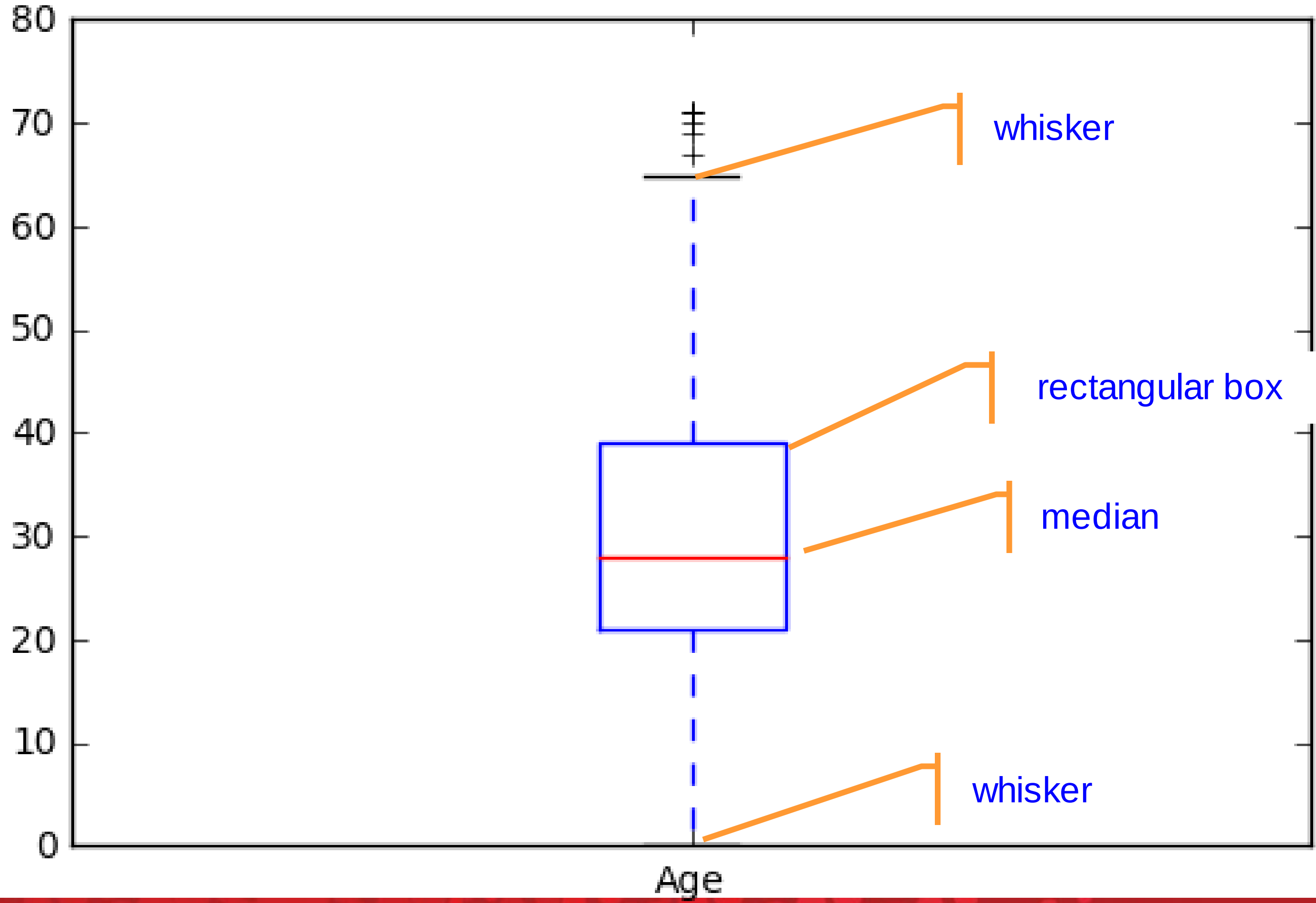
- Similar to histograms, the density plot can also figure out whether there are distributions peaks or valleys:

```
titanic['Age'].plot(kind='density')
plt.xlabel('Age')
```

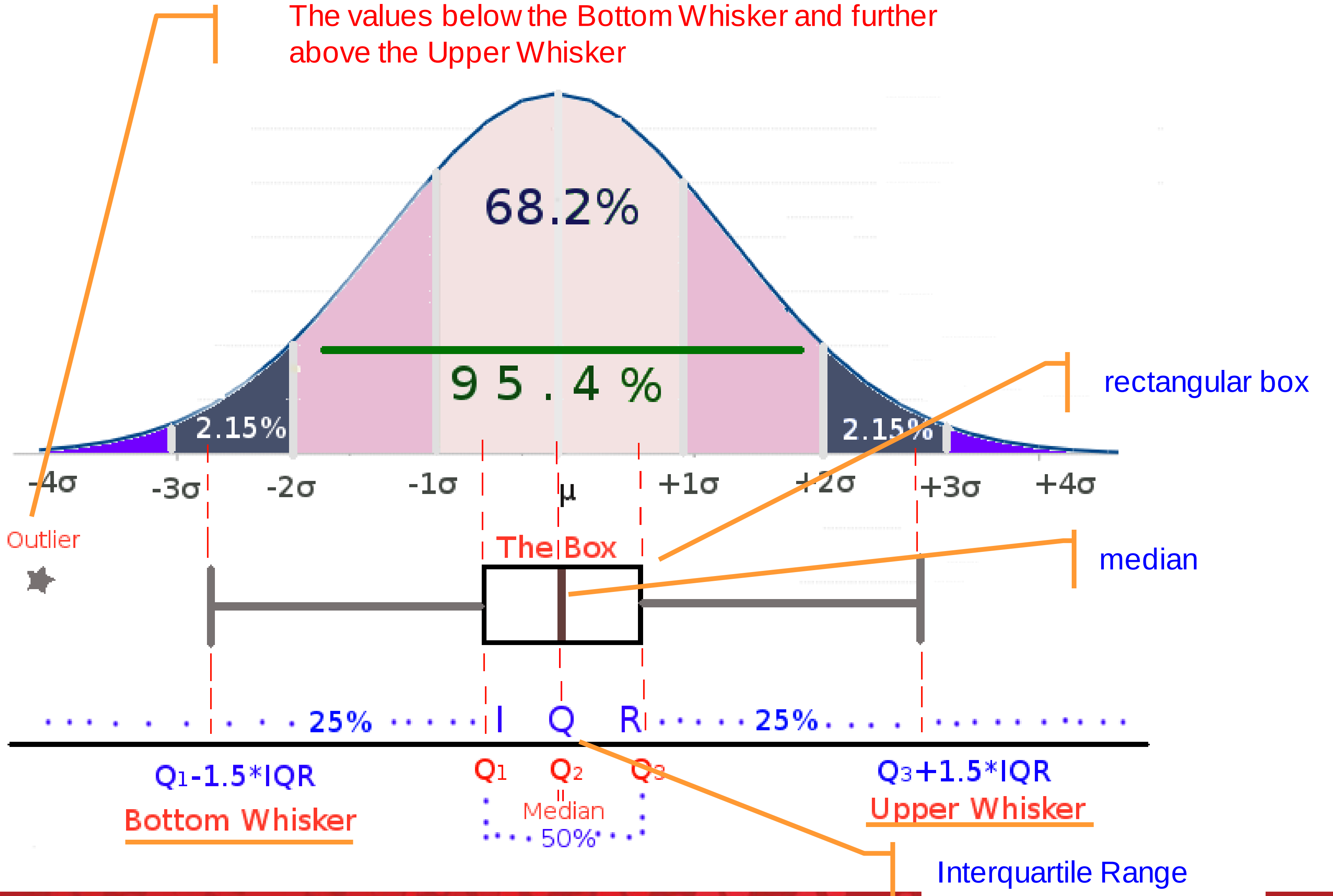


boxPlot

- Boxplots draft the key figures in the distribution and help you spot outliers.
  - titanic['Age'].dropna().plot(kind='box')



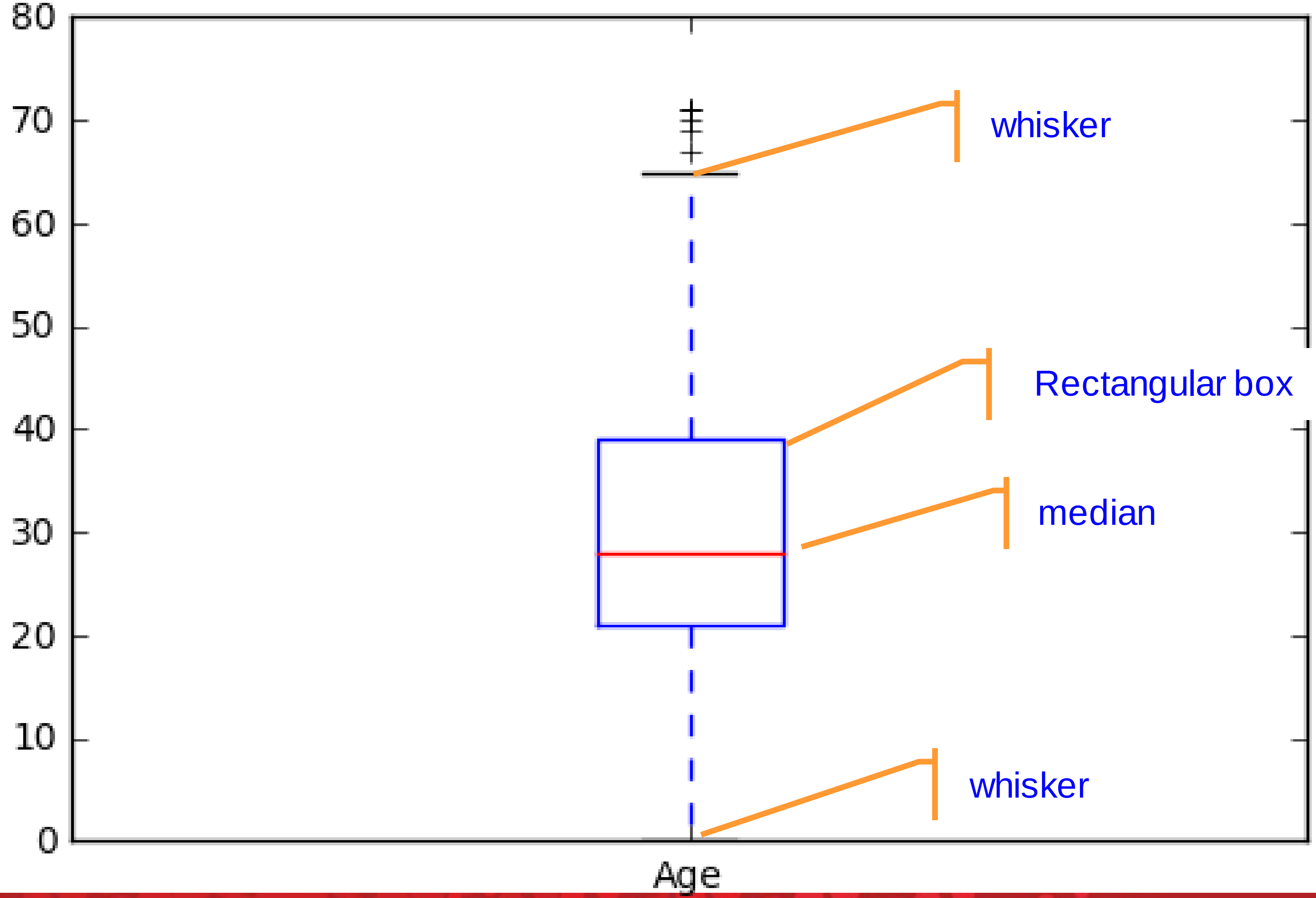
boxPlot



https://estatistics.eu/what-is-statistics-charts-boxplot/

boxPlot

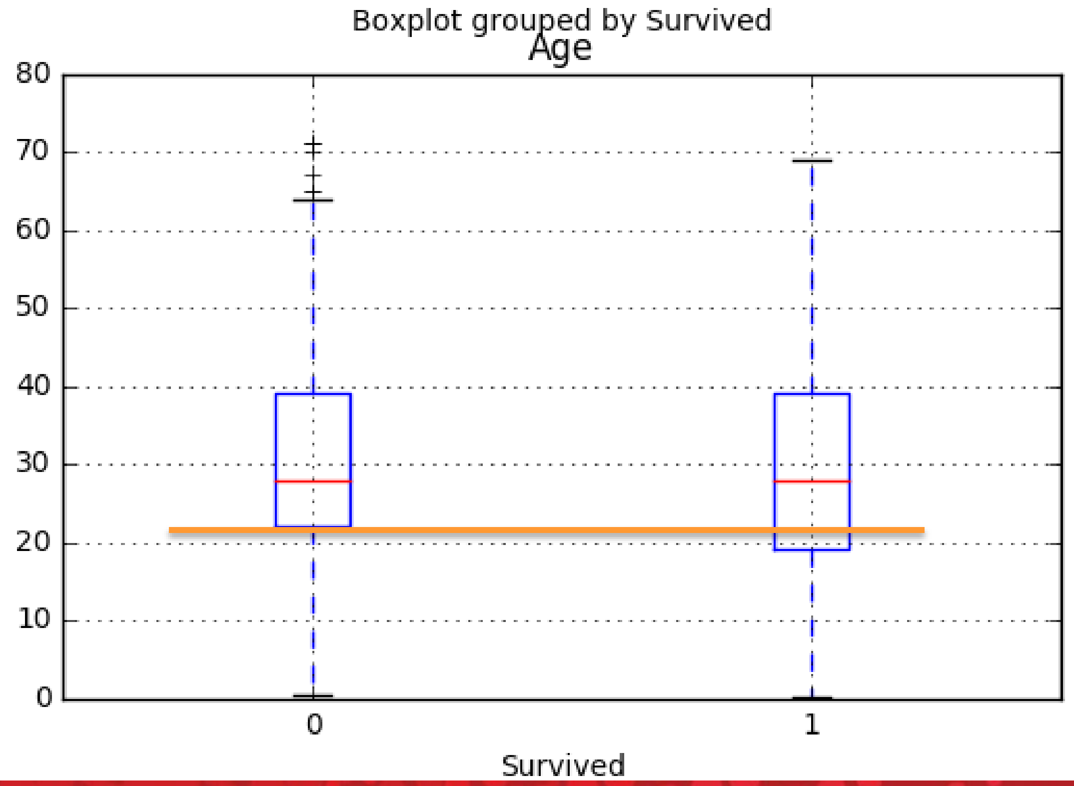
- Boxplots draft the key figures in the distribution and help you spot outliers.
  - titanic['Age'].dropna().plot(kind='box')



boxPlot

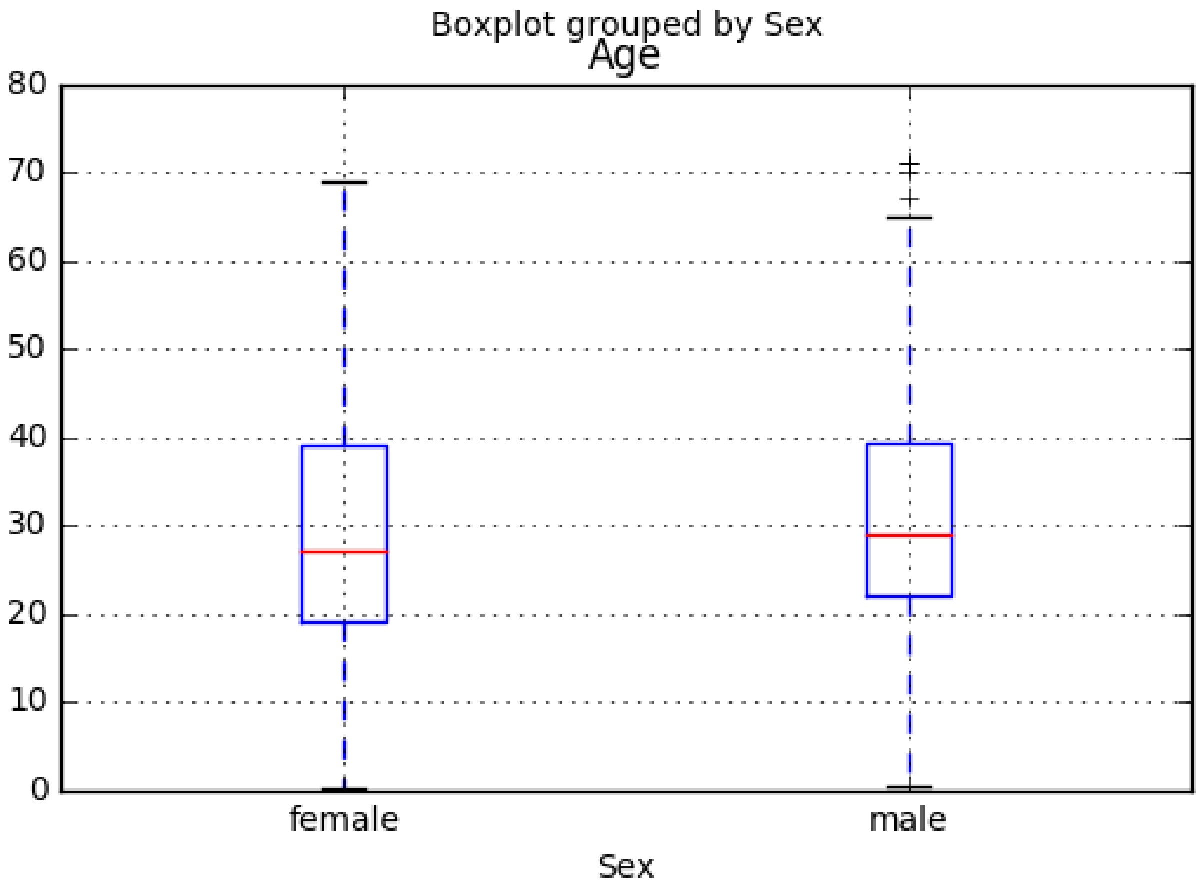
- by groups

- If there are groups in the data (from categorical variables),
  - just point out the variable for which you need the boxplot and specify that you need to have the data separated by the groups:
- titanic.dropna().boxplot(column='Age',by='Survived')



boxPlot
- by groups

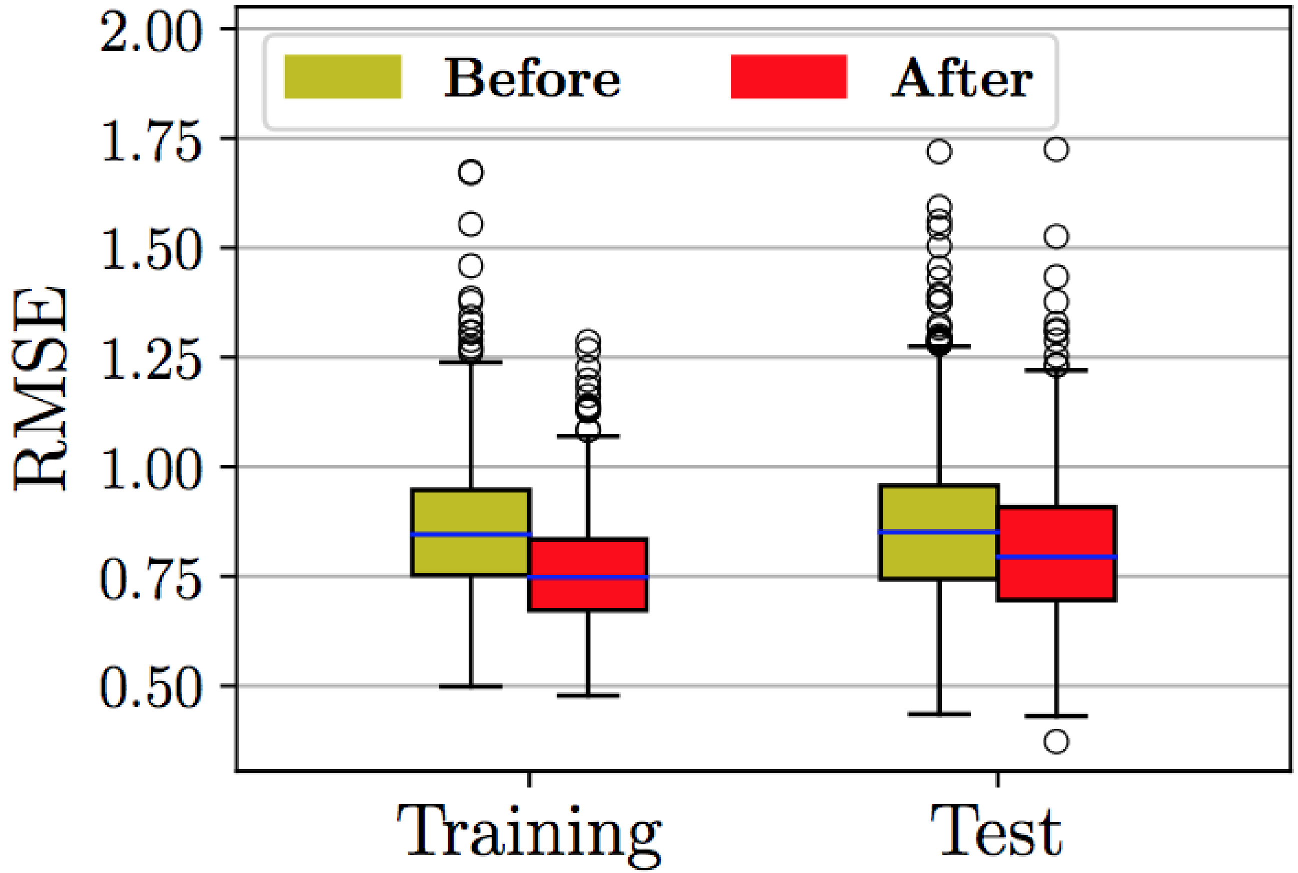
- titanic.dropna().boxplot(column='Age',by='Sex')



boxPlot

- by groups

- titanic.dropna().boxplot(column='Age',by='Sex')



Scatterplots

- Scatterplots can be used to effectively understand
  - whether the variables are in a nonlinear relationship,
  - and you can get an idea about their best possible transformations to achieve linearization.
- If you are using an algorithm based on linear combinations,
  - such as linear or logistic regression,
  - figuring out how to render their relationship more linearly will help you achieve a better predictive power

Scatterplots

```
import pandas as pd
import matplotlib.pyplot as plt
iris_filename = 'datasets-uci-iris.csv'
iris = pd.read_csv(iris_filename, sep=',', decimal='.', header=None,
names= ['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
'target'])

v = iris['target'].unique()
m_1 = iris['target'] == v[0]
m_2 = iris['target'] == v[1]
m_3 = iris['target'] == v[2]
iris.loc[m_1, 'target'] = 0
iris.loc[m_2, 'target'] = 1
iris.loc[m_3, 'target'] = 2
iris['target'].value_counts()

colors_palette = {0: 'red', 1:'green', 2:'blue'}

colors = [colors_palette[c] for c in iris['target']]
iris.plot(kind='scatter',x=0,y=1,c=colors)
plt.show()
```

Load packages and iris dataset

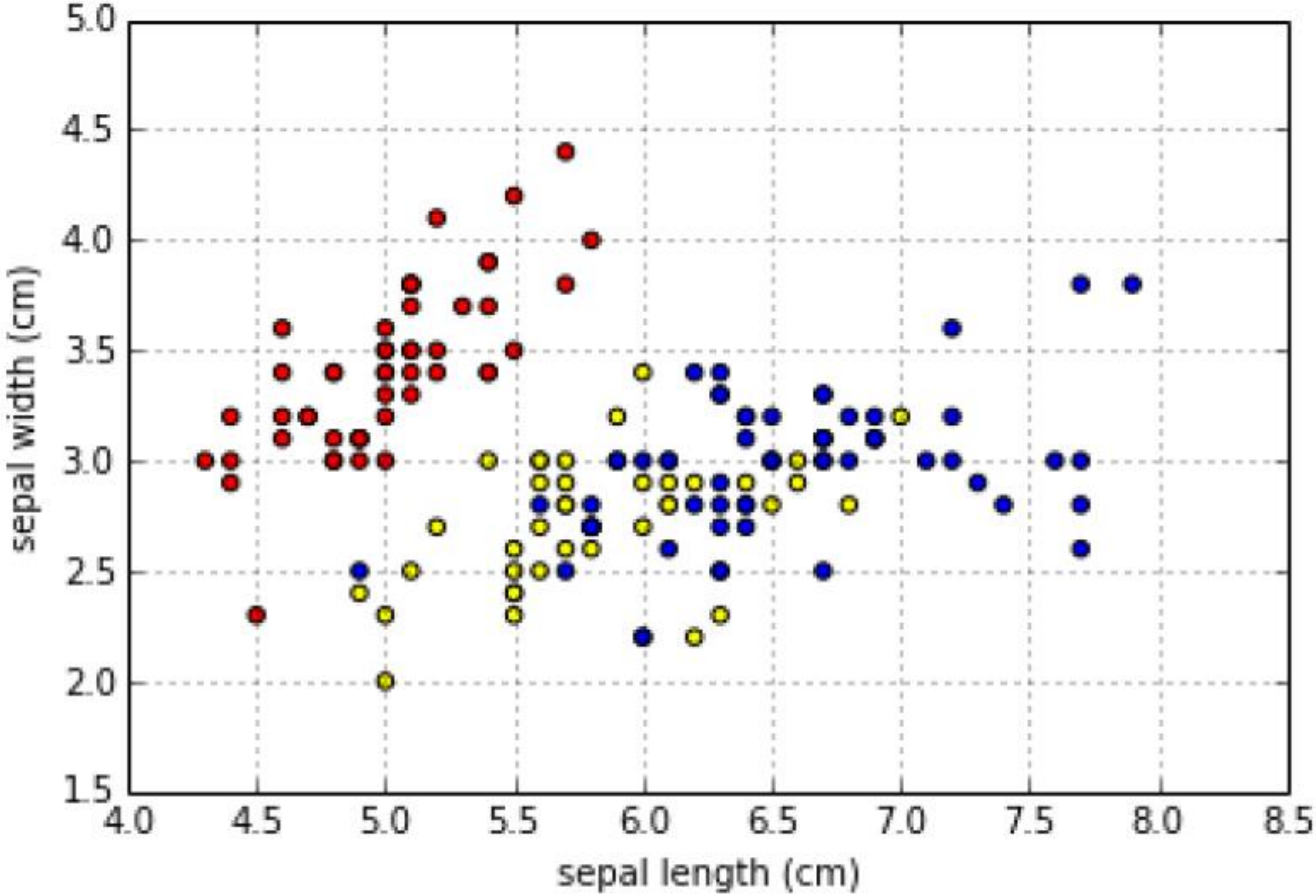
Get the unique values of 'target' column,
Then define three masks for each unique value.

Reassign each 'target' to a classLabel (0, 1, 2) by using the three masks.
These new coded class labels will be used as 'index' later

Check the revised 'target' values

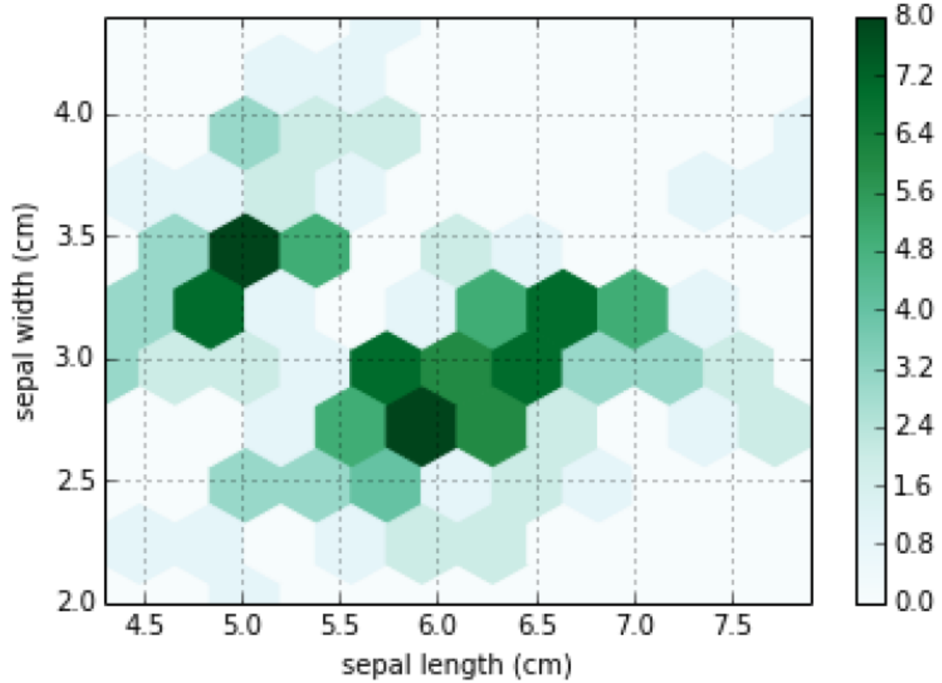
Define a dictionary: colors_palette
Define a colors list for each observation in iris according to its 'target' value, which is used an index to find the corresponding color in the dictionary
Finally, plot the scatter

Scatterplots



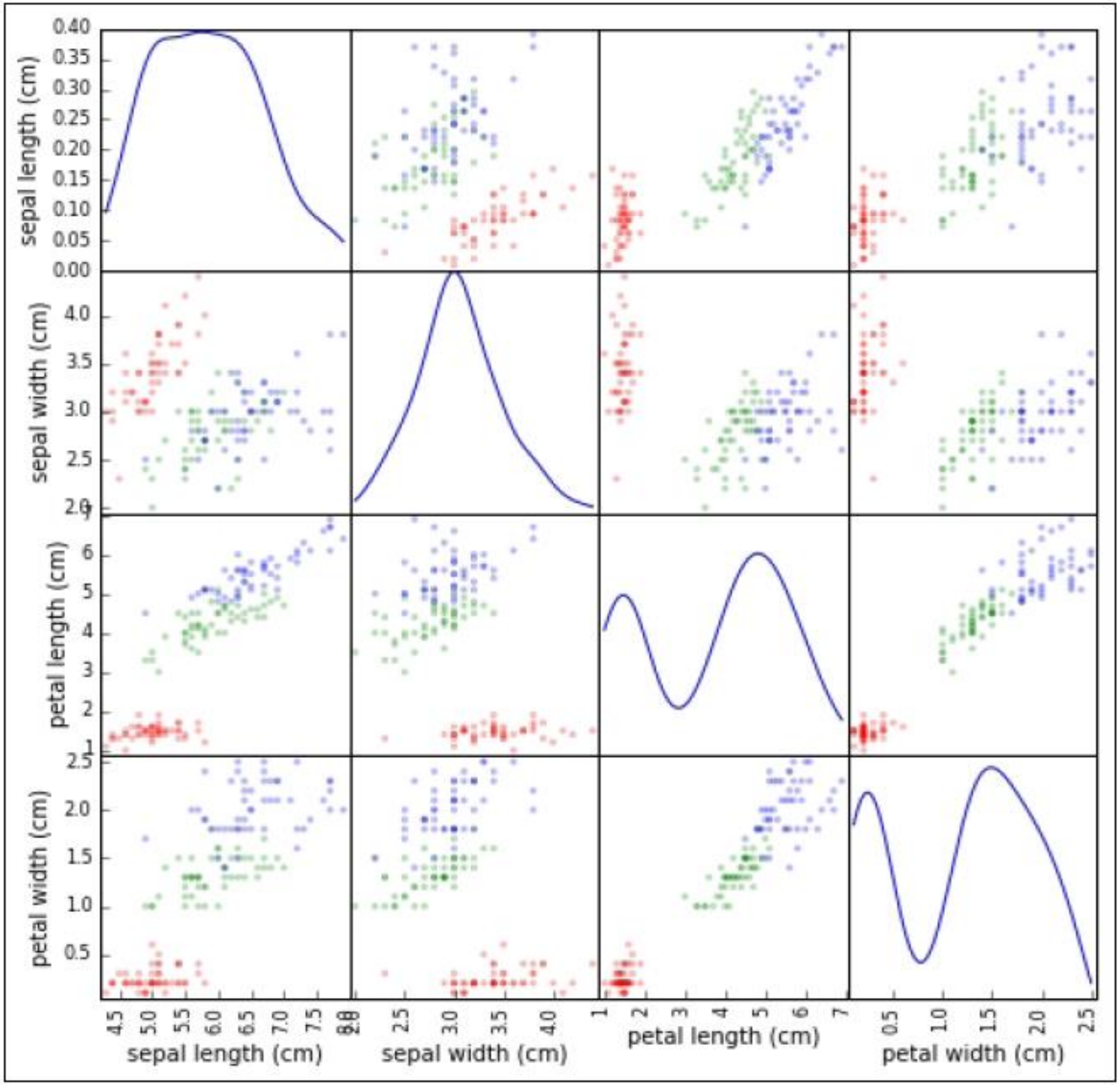
Scatterplots
- Hexbin

- Scatterplots can be turned into hexagonal binning plots.
- It helps you effectively visualize the point densities,
- Thus, revealing natural clusters hidden in your data by using some of the variables in the dataset:
- iris.plot(kind='hexbin', x=0,y=1,gridsize=10)



Scatterplots
- scatter matrix

- Scatterplots are bivariate.
- So, you'll require a single plot for every variable combination.
- If your variables are less in number,
  - a quick turnaround is to automatically place a command to draw a matrix of scatterplots.
  - (otherwise, the visualization will get cluttered)


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

Scatterplots

- scatter matrix

```
import pandas as pd
import matplotlib.pyplot as plt
iris_filename = 'datasets-uci-iris.csv'
iris = pd.read_csv(iris_filename, sep=',', decimal='.', header=None,
    names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'target'])

v = iris['target'].unique()
m_1 = iris['target'] == v[0]
m_2 = iris['target'] == v[1]
m_3 = iris['target'] == v[2]
iris.loc[m_1, 'target'] = 0
iris.loc[m_2, 'target'] = 1
iris.loc[m_3, 'target'] = 2
iris['target'].value_counts()

colors_palette = {0: 'red', 1: 'green', 2: 'blue'}
colors = [colors_palette[c] for c in iris['target']]

from pandas.plotting import scatter_matrix

scatter_matrix(iris, alpha=0.2, figsize=(16, 16), c=colors, diagonal='hist')
plt.show()
```

Load packages and iris dataset

Get the unique values of 'target' column,
Then define three masks for each unique value.

Reassign each 'target' to a classLabel (0, 1, 2) by using the three masks.
These new coded class labels will be used as 'index' later

Check the revised 'target' values

Define a dictionary: colors_palette
Define a colors list for each observation in iris according to its 'target' value, which is used an index to find the corresponding color in the dictionary

Load the appropriate package
Call the scatter_matrix function

Titanic Survival Rates by Sex

```
import pandas as pd
import matplotlib.pyplot as plt

titanic_filename = 'Titanic.csv'
titanic = pd.read_csv(titanic_filename, sep=',', decimal='.', index_col=0)

sex_counts = titanic.dropna()['Sex'].value_counts()

mask_sex_f = titanic['Sex'] == 'female'
mask_sex_m = titanic['Sex'] == 'male'

f_survive = titanic.dropna().loc[mask_sex_f, 'Survived'].value_counts()
m_survive = titanic.dropna().loc[mask_sex_m, 'Survived'].value_counts()

rate = [f_survive[1] / float(sex_counts['female']), m_survive[1] / float(sex_counts['male'])]
plt.bar(list(range(2)), rate, color='b', align='center')
plt.xticks(list(range(2)), ['female', 'male'])
plt.xlabel('Sex')
plt.ylabel('Propotion Survived')
plt.title('Titanic Survival Rates by Sex')
```

Load packages and dataset

Check the number of female/male passengers

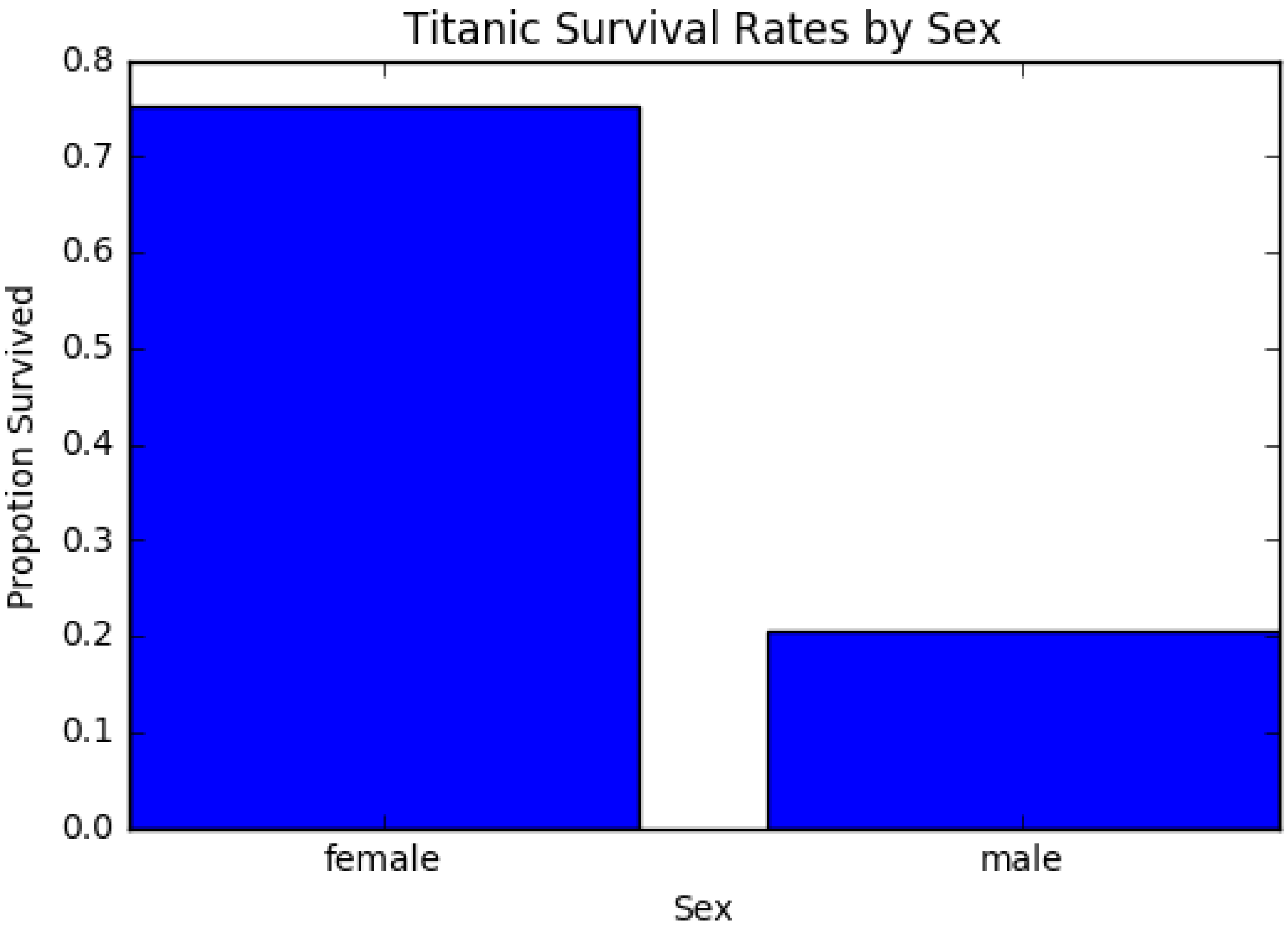
Create two masks for female and male

Use masks to select the survived female/male passengers

Calculate the survival rates for female and male

Plot the survival rates as bar graph
Specify the xticks, x/ylables and title of the graph

Titanic Survival Rates by Sex



Practical Data Science – COSC2670

PART 3:
ASSIGNMENT 1

Assignment 1

- The assignment 1 will be released today:
  - Data Cleaning and Summarising
  - Due: 23:59, Wednesday 15 April, 2020 (week 6)
  - This assignment is worth 15% of your overall mark.

- The specifications of this assignment will be under the
  - Canvas Course Shell -> Assignments

- Where to develop and test your code:
  - Jupyter Notebook on Lab PCs and Teaching Servers.

References and Further Reading

- A. Boschetti and L. Massaron, *Python Data Science Essentials*, Chapters 2 and 6
- D. Cielen and A. Meysman and M. Ali, *Introducing Data Science*, Chapter 2

Data Science

Thanks!