COMP5310: Principles of Data Science

W3: Data Exploration with Python

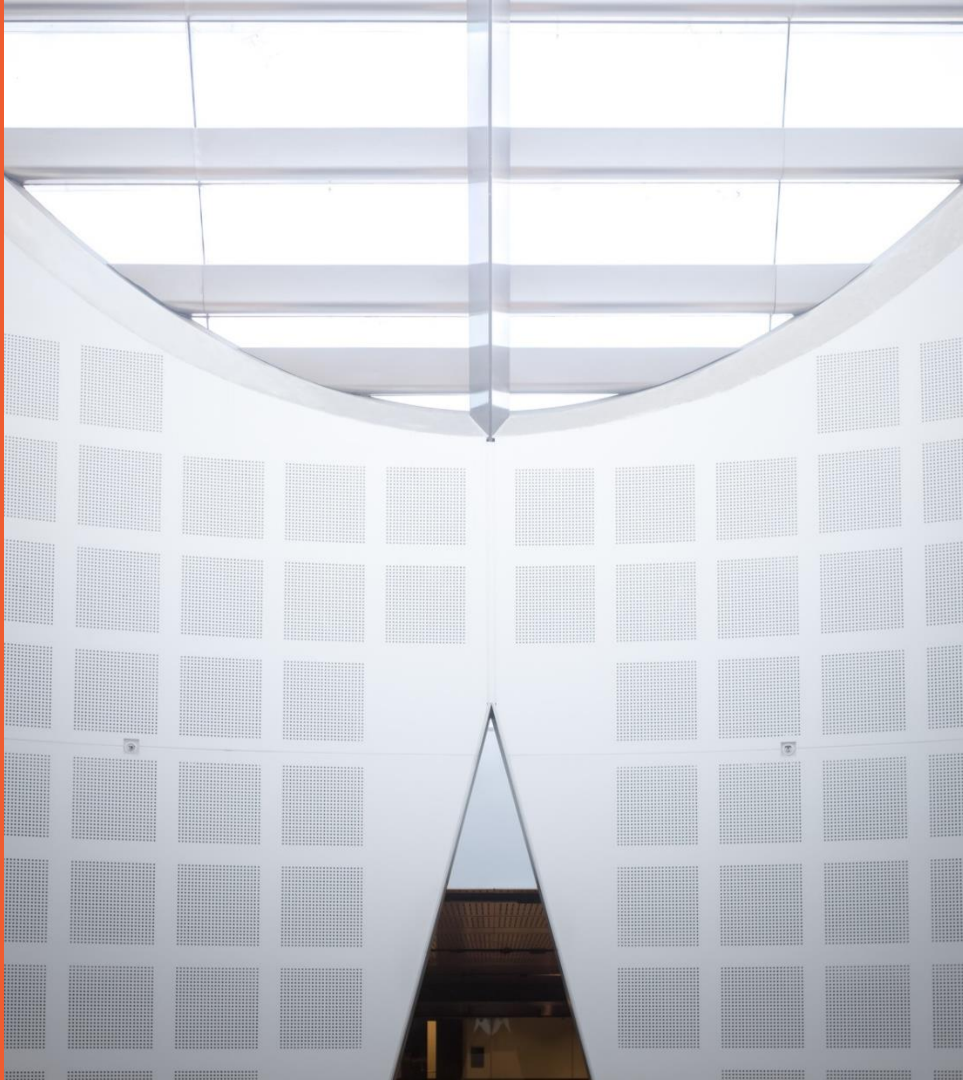
Presented by

Maryam Khanian

School of Computer Science

Based on slides by previous lecturers of this unit of study

THE UNIVERSITY OF SYDNEY



Last week: Data cleaning and exploration (via spreadsheet)

Objective

- Use interactive tools to explore a new data set quickly.

Lecture

- Data types, cleaning, preprocessing.
- Descriptive statistics, e.g., mean, stdev, median.
- Descriptive visualisation, e.g., scatterplots, histograms.

Readings

- Introduction to Data Mining: Ch 2.1.1
- Data Science from Scratch: Ch 2-3.

Exercises

- Spreadsheets: Visualisation.
- Spreadsheets: Descriptive stats.

TO-DO in W2

- Ed Lessons Python modules 4-6.
- Ed Lessons SQL modules 16-17.
- Explore project data.

PYTHON AND JUPYTER NOTEBOOKS

THE UNIVERSITY OF SYDNEY

Python is great for prototyping

- **Interpreted**: direct execution without compilation.
- **Dynamically-typed**: don’t have to declare a static type.
- **Readable**: easy-to-understand syntax.
- **Deployable**: easy to incorporate in applications.

Python Recap

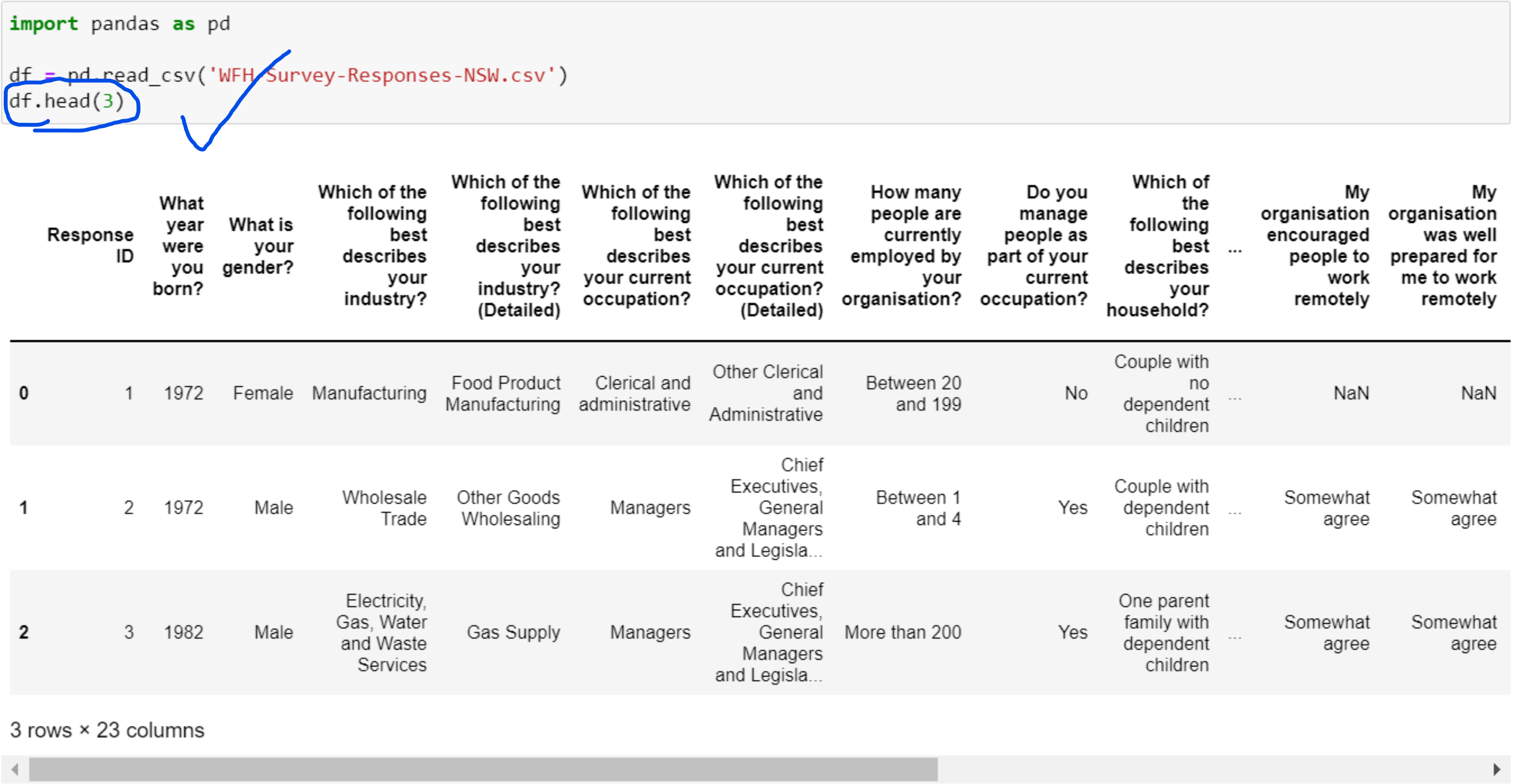
- General program syntax.
- Variables and types.
  - Integer (int) and float numbers, string types, type conversion.
  - List of values (list, array).
- Condition statements (if/elif/else/while).
- For loops, ranges (for x in range(n)).
- Functions.
  - E.g., input(), print(), len(), lower(), upper(), …
  - Nesting of functions; example: print(len(*str*.upper()))

Python import system

- Ed Lessons, so far, concentrated on *built-in* functions.
- Additional functionality available via **import** statement.
  - Gives access to classes and functions from various 3rd party modules.
  - Examples:
    - **pandas:** We will use comma-separated file format in pandas.
    - **numpy:** multi-dimensional arrays and matrices and high-level mathematical functions support.
    - **matplotlib:** creating static, animated, and interactive visualizations.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Read data using Pandas in Jupyter notebook



```
import pandas as pd

df = pd.read_csv('WFH_Survey-Responses-NSW.csv')
df.head(3)
```

	Response ID	What year were you born?	What is your gender?	Which of the following best describes your industry?	Which of the following best describes your industry? (Detailed)	Which of the following best describes your current occupation?	Which of the following best describes your current occupation? (Detailed)	How many people are currently employed by your organisation?	Do you manage people as part of your current occupation?	Which of the following best describes your household?	...	My organisation encouraged people to work remotely	My organisation was well prepared for me to work remotely
0	1	1972	Female	Manufacturing	Food Product Manufacturing	Clerical and administrative	Other Clerical and Administrative	Between 20 and 199	No	Couple with no dependent children	...	NaN	NaN
1	2	1972	Male	Wholesale Trade	Other Goods Wholesaling	Managers	Chief Executives, General Managers and Legisla...	Between 1 and 4	Yes	Couple with dependent children	...	Somewhat agree	Somewhat agree
2	3	1982	Male	Electricity, Gas, Water and Waste Services	Gas Supply	Managers	Chief Executives, General Managers and Legisla...	More than 200	Yes	One parent family with dependent children	...	Somewhat agree	Somewhat agree

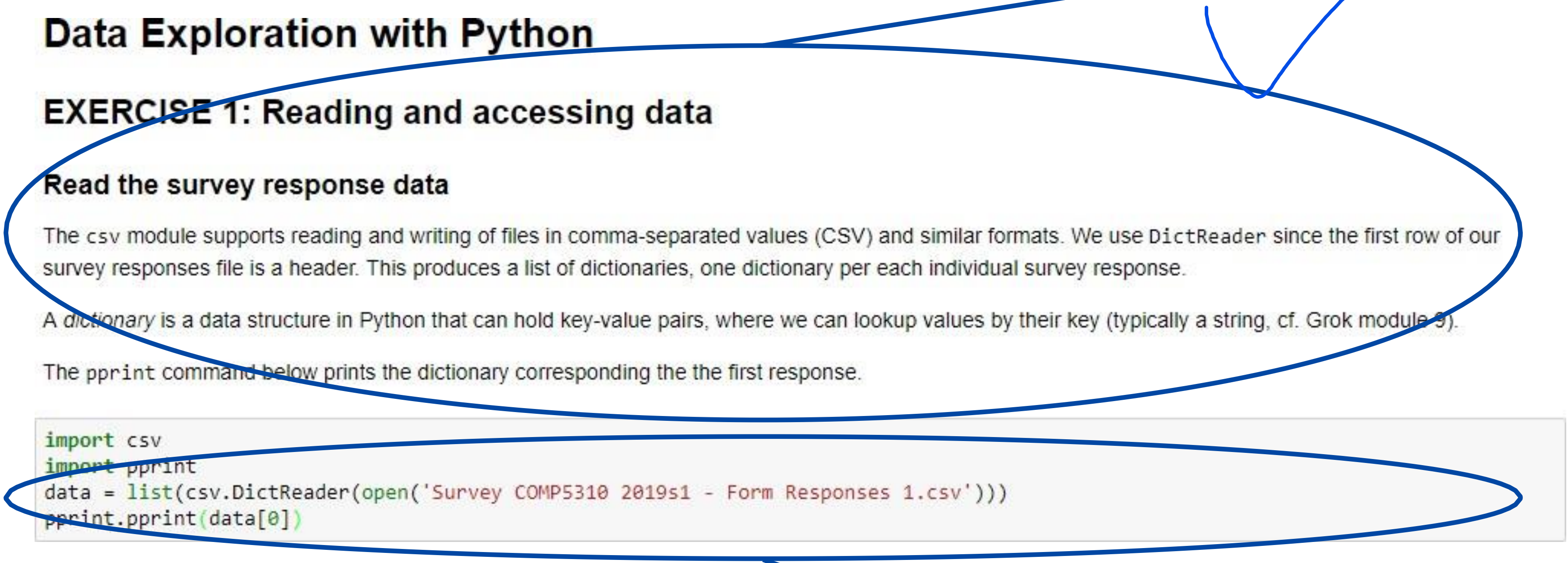
3 rows × 23 columns

Python has excellent open-source data libraries

- **scipy:** libraries for scientific and technical computing.
- **numpy:** support for large multidimensional arrays and matrices.
- **matplotlib:** port of MATLAB plotting functionality.
- **seaborn:** abstraction on top of matplotlib (less flexible but easier to use).
- **scikit-learn:** machine learning library.
- **nltk:** natural language toolkit.
- **pandas:** R-like data frame and associated manipulations.

Jupyter notebook cells

Markdown cell for formatted text



Data Exploration with Python

EXERCISE 1: Reading and accessing data

Read the survey response data

The `csv` module supports reading and writing of files in comma-separated values (CSV) and similar formats. We use `DictReader` since the first row of our survey responses file is a header. This produces a list of dictionaries, one dictionary per each individual survey response.

A *dictionary* is a data structure in Python that can hold key-value pairs, where we can lookup values by their key (typically a string, cf. Grok module 9).

The `pprint` command below prints the dictionary corresponding the the first response.

```
import csv
import pprint
data = list(csv.DictReader(open('Survey COMP5310 2019s1 - Form Responses 1.csv')))
pprint.pprint(data[0])
```

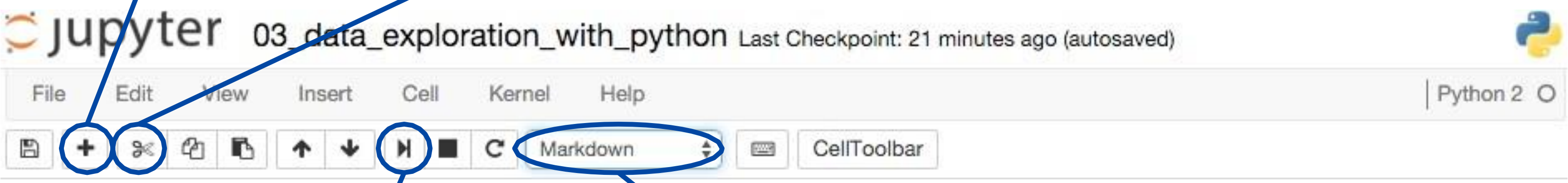
Code cell for writing Python commands

Jupyter menu bar

✓

Add new cell

Cut current cell


jupyter
03_data_exploration_with_python Last Checkpoint: 21 minutes ago (autosaved)
File
Edit
View
Insert
Cell
Kernel
Help
Python 2
Markdown
CellToolbar

“Run” cell
- Execute code
- Render text

Change cell type

Reading data using csv

✓

- Python **csv** module.
  - Reads/writes comma-separated values (csv) files with escaping.
  - csv.reader() reads rows into arrays.
  - csv.DictReader() reads rows into ***dictionaries.***

```
import csv
with open('WFH-Survey-Responses-NSW.csv') as csv_file:
    reader = csv.reader(csv_file, delimiter=",")
```

```
import csv
with open('WFH-Survey-Responses-NSW.csv', mode="r") as csv_file:
    reader = csv.DictReader(csv_file)
```

Reading data using Pandas

- Reading csv files using **Pandas** is simple.
  - read_csv() reads comma-separated values (csv) files into DataFrame.
- You can also use **Pandas** to read excel files.
  - read_excel() reads files with xls, xlsx, xlsm, xlsb, odf, ods and odt file extensions.

```
import pandas as pd
df = pd.read_csv('WFH-Survey-Responses-NSW.csv')
```

```
import pandas as pd
df = pd.read_excel('WFH-Survey-Responses-NSW.xlsx')
```

Pandas: Series and DataFrames

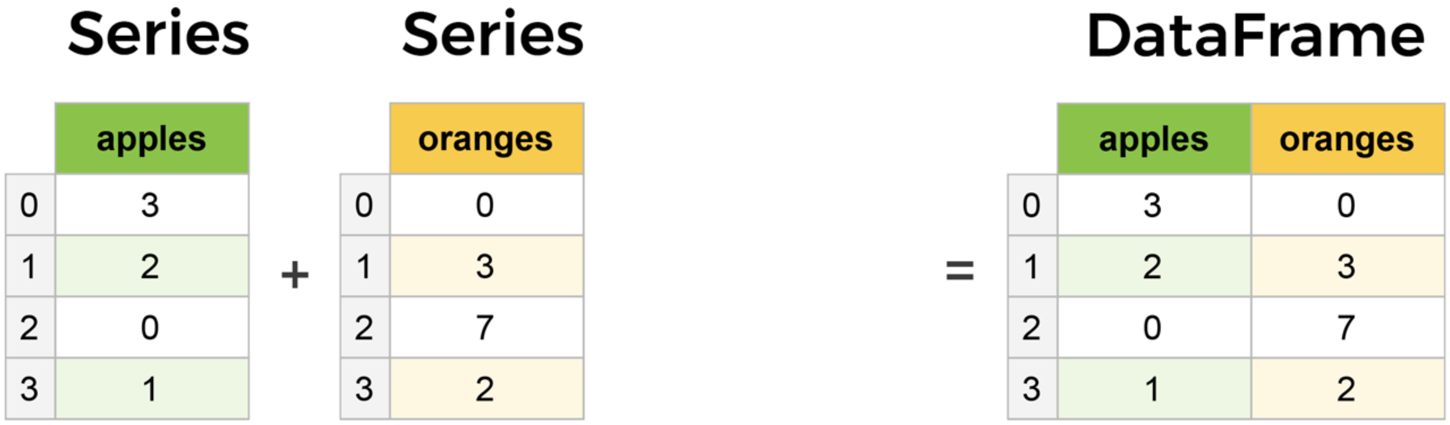
✓

Series

*A Pandas **Series** is like **a column in a table**.*
*It is a **one-dimensional array** holding data of any type.*

DataFrames

*A Pandas **DataFrame** is a **2 dimensional data structure**,*
*like a 2 dimensional array, or a **table with rows and columns**.*



Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Pandas Series

✓

- Create a Series

```
apples = pd.Series([3,2,0,1])
```

- Can access a Series in the same way as a list

```
apples[2]
```

- Can explicitly specify the index for accessing a Series

```
apples = pd.Series([3,2,0,1], index=[‘d1’, ‘d2’, ’d3’, ‘d4’)
apples[‘d3’]
```

- Can construct a Series from a dictionary

```
apples_dic = {'d1': 3, 'd2': 2, 'd3': 0, 'd4': 1}
apples = pd.Series(apples_dic)
```

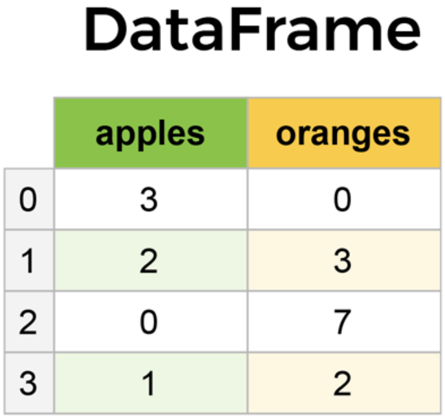
Pandas DataFrame

- DataFrame can be considered as a sequence of aligned Series objects, i.e., they share the same index.

```
oranges = pd.Series([0,3,7,2], index=[‘d1’, ‘d2’, ’d3’, ‘d4’)
dietary = pd.DataFrame({'apples': apples, 'oranges': oranges})
```

- DataFrame can also be considered as a specialization of a dictionary, which maps a column name to a Series of column data
  - dietary[‘apples’] refers to a column of the DataFrame
- Rows of a DataFrame can be accessed with loc, iloc

```
dietary.loc[‘d3’]
dietary.iloc[2]
```



DataFrame

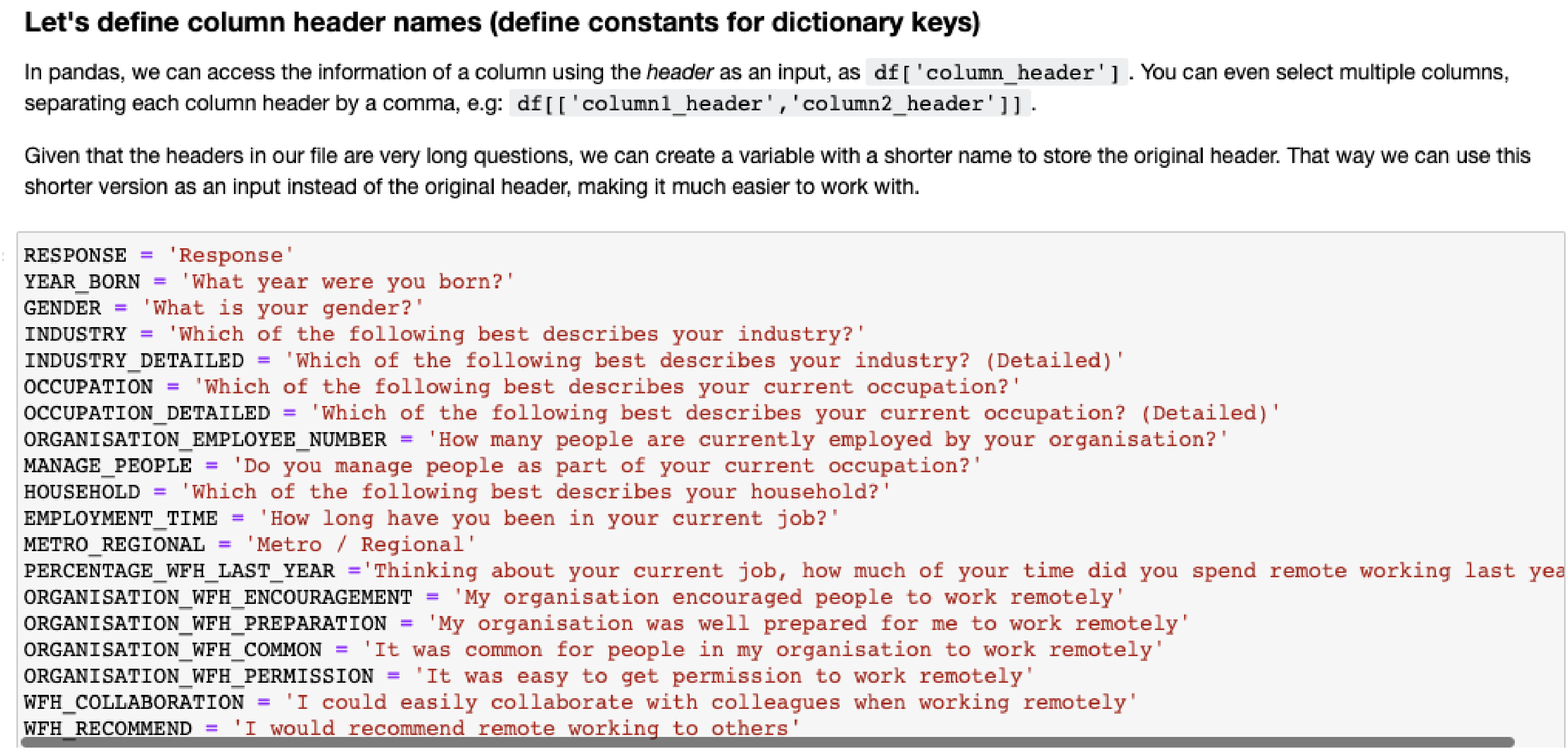
	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Pandas DataFrame

- To access a column using Pandas, you just need to know the name of the column you want to read. ✓
  - df[“column_1”] will give you access to the column named “column_1”.
- You can rename columns to make it easier to access. ✓
  - df.rename(columns={“A”: “a”, “B”: “ b”}) will rename columns with names “A” and “B” to “a” and “b”, respectively.
- You can “drop” columns that you don’t need.
  - df.drop(columns=[“A”, “B”, “C”]) will drop the columns with names “A”, “B” and “C”.
- You can “drop” ALL rows/columns containing NaN values.
  - df.dropna() to drop ALL rows with NaN values.
  - df.dropna(“columns”) to drop ALL columns with NaN values.

DESCRIPTIVE STATISTICS

Renaming Columns for easier access



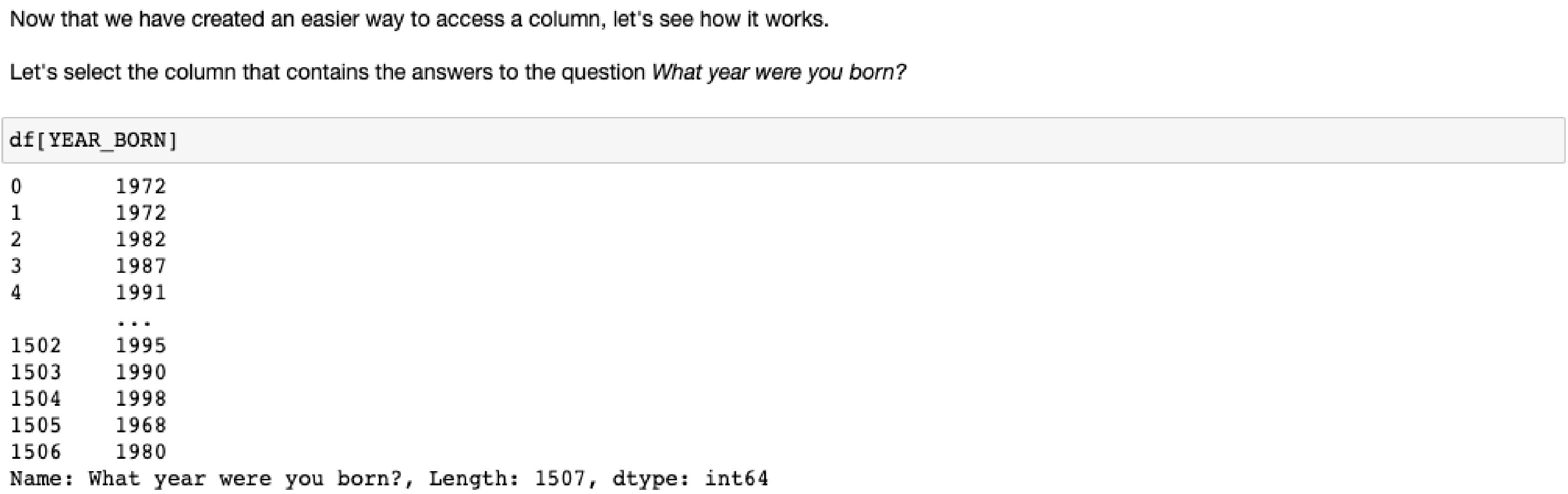
Let's define column header names (define constants for dictionary keys)

In pandas, we can access the information of a column using the *header* as an input, as `df['column_header']`. You can even select multiple columns, separating each column header by a comma, e.g: `df[['column1_header','column2_header']]`.

Given that the headers in our file are very long questions, we can create a variable with a shorter name to store the original header. That way we can use this shorter version as an input instead of the original header, making it much easier to work with.

```
RESPONSE = 'Response'
YEAR_BORN = 'What year were you born?'
GENDER = 'What is your gender?'
INDUSTRY = 'Which of the following best describes your industry?'
INDUSTRY_DETAILED = 'Which of the following best describes your industry? (Detailed)'
OCCUPATION = 'Which of the following best describes your current occupation?'
OCCUPATION_DETAILED = 'Which of the following best describes your current occupation? (Detailed)'
ORGANISATION_EMPLOYEE_NUMBER = 'How many people are currently employed by your organisation?'
MANAGE_PEOPLE = 'Do you manage people as part of your current occupation?'
HOUSEHOLD = 'Which of the following best describes your household?'
EMPLOYMENT_TIME = 'How long have you been in your current job?'
METRO_REGIONAL = 'Metro / Regional'
PERCENTAGE_WFH_LAST_YEAR ='Thinking about your current job, how much of your time did you spend remote working last yea
ORGANISATION_WFH_ENCOURAGEMENT = 'My organisation encouraged people to work remotely'
ORGANISATION_WFH_PREPARATION = 'My organisation was well prepared for me to work remotely'
ORGANISATION_WFH_COMMON = 'It was common for people in my organisation to work remotely'
ORGANISATION_WFH_PERMISSION = 'It was easy to get permission to work remotely'
WFH_COLLABORATION = 'I could easily collaborate with colleagues when working remotely'
WFH_RECOMMEND = 'I would recommend remote working to others'
```

Accessing Columns



Now that we have created an easier way to access a column, let's see how it works.

Let's select the column that contains the answers to the question *What year were you born?*

```
df[YEAR_BORN]
```

```
0          1972
1          1972
2          1982
3          1987
4          1991
           ...
1502       1995
1503       1990
1504       1998
1505       1968
1506       1980
Name: What year were you born?, Length: 1507, dtype: int64
```

Cleaning data: convert to correct types

The Python **csv module** reads everything as ***string types***.

- 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.

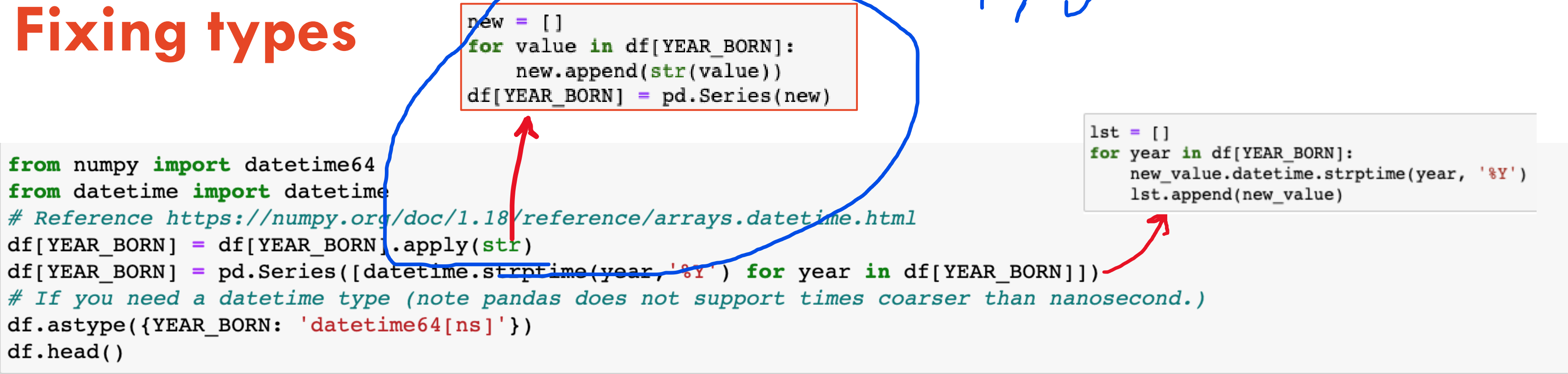
Pandas will ***guess types***, unless specified.

```
import pandas as pd
df = pd.read_csv('WFH-Survey-Responses-NSW.csv’ , dtype={"A":
int, "B": float})
```

- If not specified, you can convert as appropriate afterwards, if needed.
  - pandas.DataFrame.dtype
  - DataFrame.astype

Fixing types

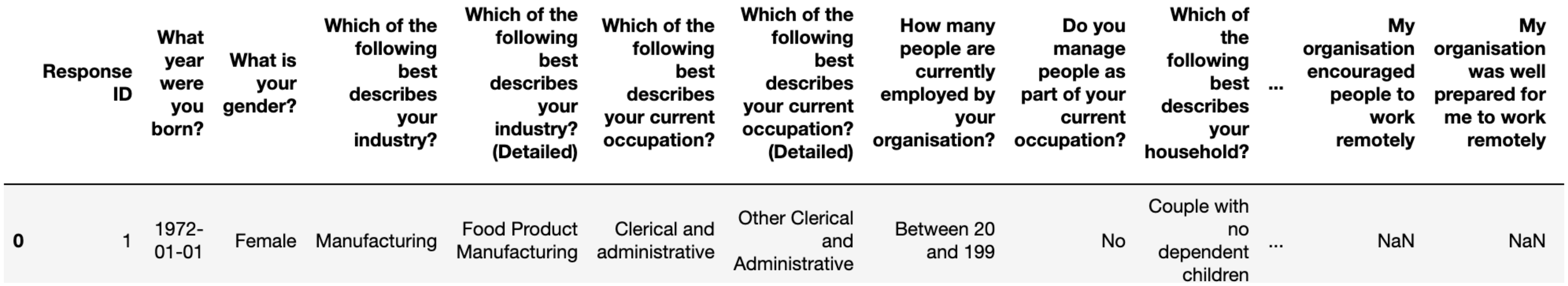
apply



```
new = []
for value in df[YEAR_BORN]:
    new.append(str(value))
df[YEAR_BORN] = pd.Series(new)
```

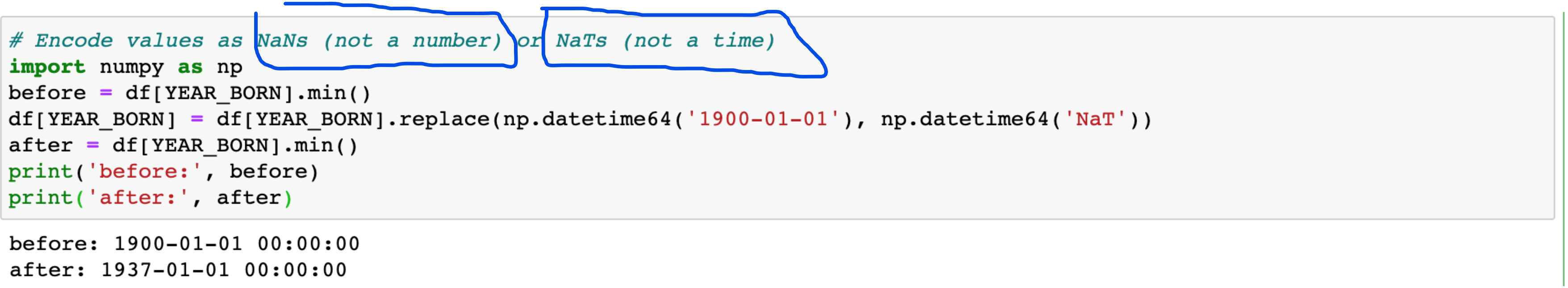
```
from numpy import datetime64
from datetime import datetime
# Reference https://numpy.org/doc/1.18/reference/arrays.datetime.html
df[YEAR_BORN] = df[YEAR_BORN].apply(str)
df[YEAR_BORN] = pd.Series([datetime.strptime(year, '%Y') for year in df[YEAR_BORN]])
# If you need a datetime type (note pandas does not support times coarser than nanosecond.)
df.astype({YEAR_BORN: 'datetime64[ns]'})
df.head()
```

```
lst = []
for year in df[YEAR_BORN]:
    new_value.datetime.strptime(year, '%Y')
    lst.append(new_value)
```



	Response ID	What year were you born?	What is your gender?	Which of the following best describes your industry?	Which of the following best describes your industry? (Detailed)	Which of the following best describes your current occupation?	Which of the following best describes your current occupation? (Detailed)	How many people are currently employed by your organisation?	Do you manage people as part of your current occupation?	Which of the following best describes your household?	...	My organisation encouraged people to work remotely	My organisation was well prepared for me to work remotely
0	1	1972-01-01	Female	Manufacturing	Food Product Manufacturing	Clerical and administrative	Other Clerical and Administrative	Between 20 and 199	No	Couple with no dependent children	...	NaN	NaN

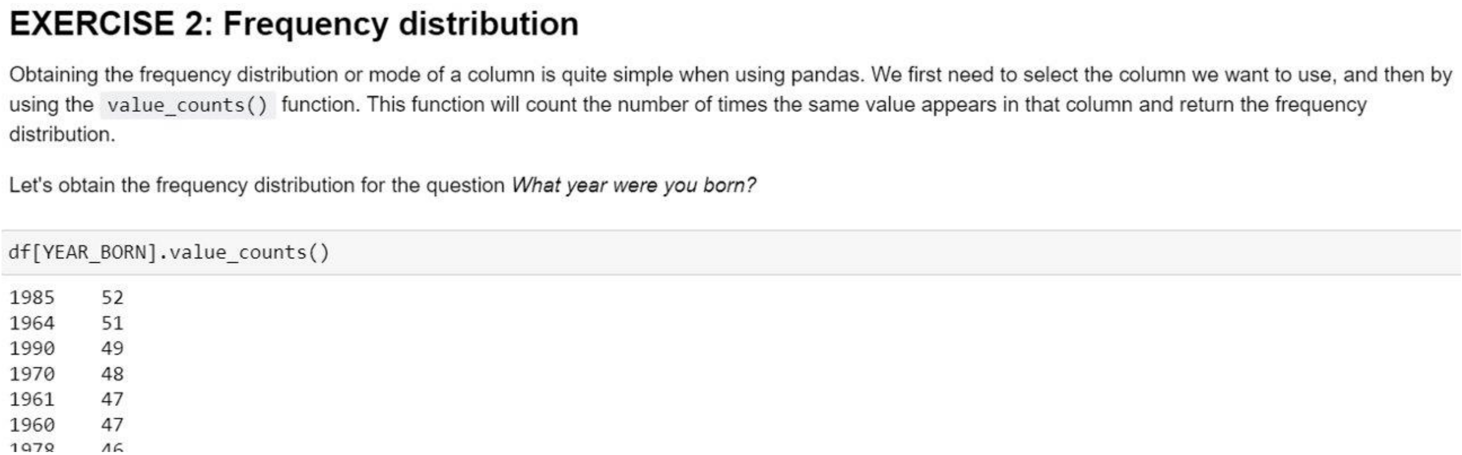
Encoding NaNs or NaTs



```
# Encode values as NaNs (not a number) or NaTs (not a time)
import numpy as np
before = df[YEAR_BORN].min()
df[YEAR_BORN] = df[YEAR_BORN].replace(np.datetime64('1900-01-01'), np.datetime64('NaT'))
after = df[YEAR_BORN].min()
print('before:', before)
print('after:', after)
```

```
before: 1900-01-01 00:00:00
after: 1937-01-01 00:00:00
```

Frequency distributions and mode



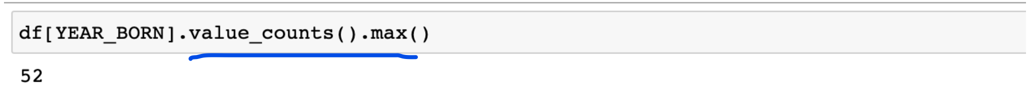
EXERCISE 2: Frequency distribution

Obtaining the frequency distribution or mode of a column is quite simple when using pandas. We first need to select the column we want to use, and then by using the `value_counts()` function. This function will count the number of times the same value appears in that column and return the frequency distribution.

Let's obtain the frequency distribution for the question *What year were you born?*

```
df[YEAR_BORN].value_counts()
```

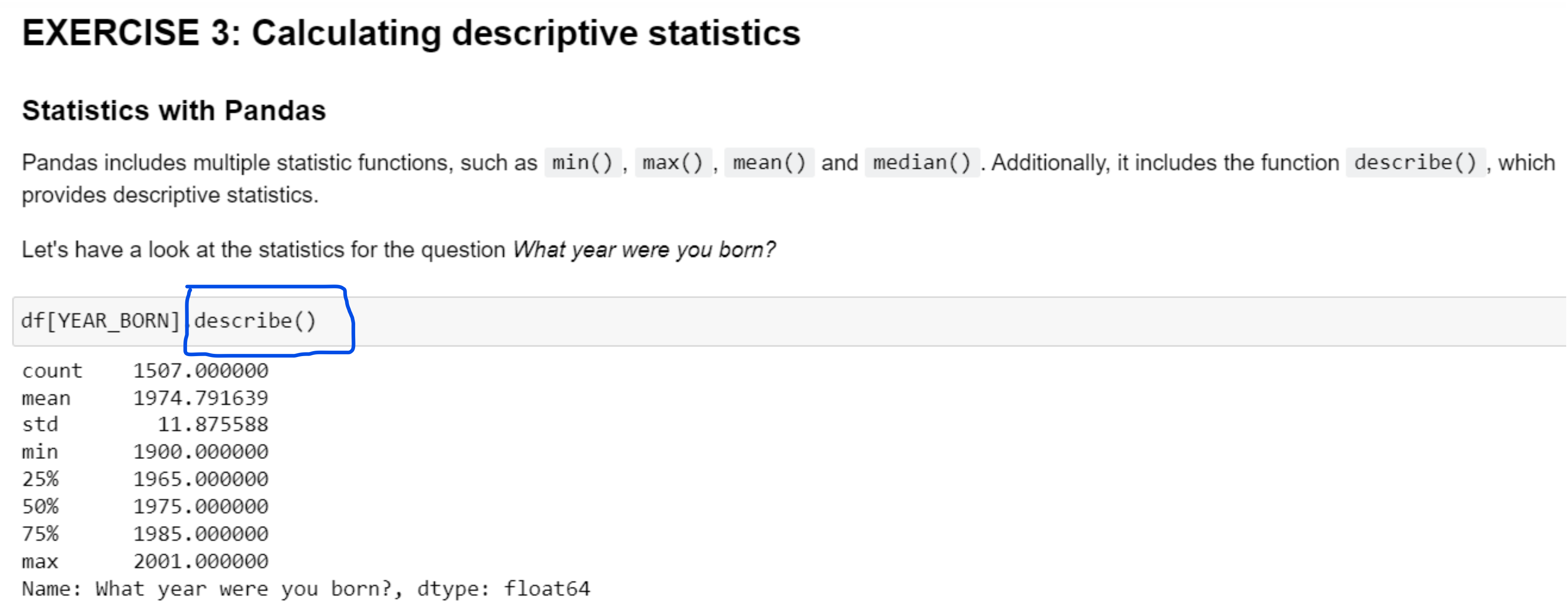
```
1985    52
1964    51
1990    49
1970    48
1961    47
1960    47
1978    46
```



```
df[YEAR_BORN].value_counts().max()
```

```
52
```

Central tendency and dispersion with pandas



EXERCISE 3: Calculating descriptive statistics

Statistics with Pandas

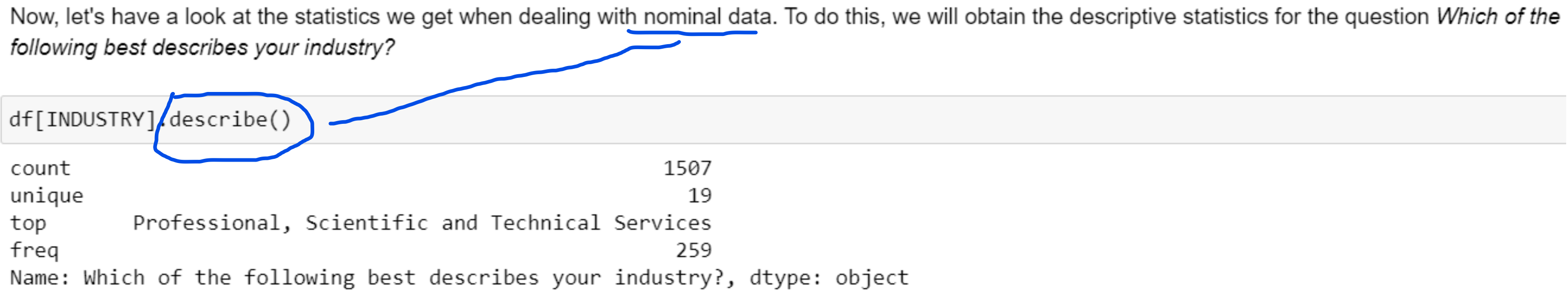
Pandas includes multiple statistic functions, such as `min()`, `max()`, `mean()` and `median()`. Additionally, it includes the function `describe()`, which provides descriptive statistics.

Let's have a look at the statistics for the question *What year were you born?*

```
df[YEAR_BORN].describe()
```

```
count    1507.000000
mean     1974.791639
std        11.875588
min      1900.000000
25%      1965.000000
50%      1975.000000
75%      1985.000000
max      2001.000000
Name: What year were you born?, dtype: float64
```

Central tendency and dispersion with pandas



Now, let's have a look at the statistics we get when dealing with nominal data. To do this, we will obtain the descriptive statistics for the question *Which of the following best describes your industry?*

```
df[INDUSTRY].describe()
```

```
count                                              1507
unique                                               19
top       Professional, Scientific and Technical Services
freq                                                259
Name: Which of the following best describes your industry?, dtype: object
```

VISUALISATION

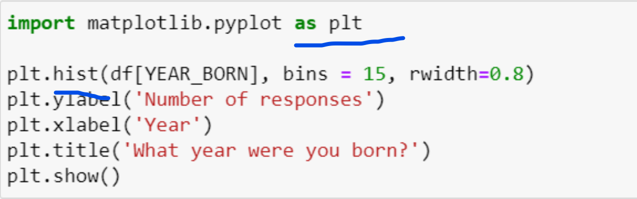
THE UNIVERSITY OF SYDNEY

Visualising data with matplotlib

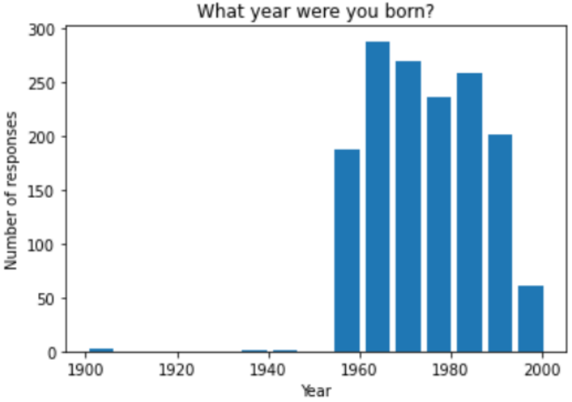
- Matplotlib provides functionality for creating various plots.
- Bar charts, line charts, scatter plots, etc.
- Reference page for pyplot:
  - http://matplotlib.org/api/pyplot_api.html
- Documentation:
  - http://matplotlib.org/contents.html

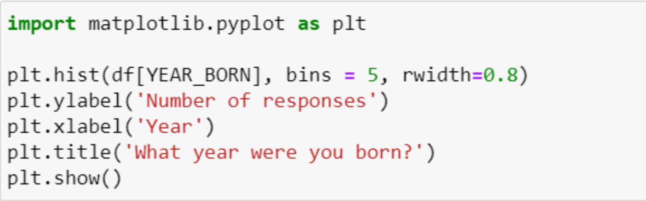
Making a histogram with matplotlib

– You can select the number of bins, columns width, etc.

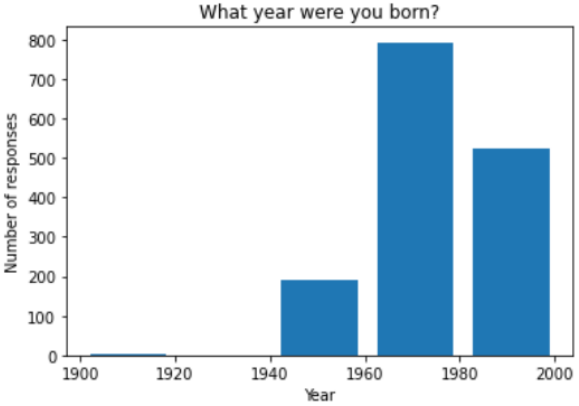


```
import matplotlib.pyplot as plt

plt.hist(df[YEAR_BORN], bins = 15, rwidth=0.8)
plt.ylabel('Number of responses')
plt.xlabel('Year')
plt.title('What year were you born?')
plt.show()
```

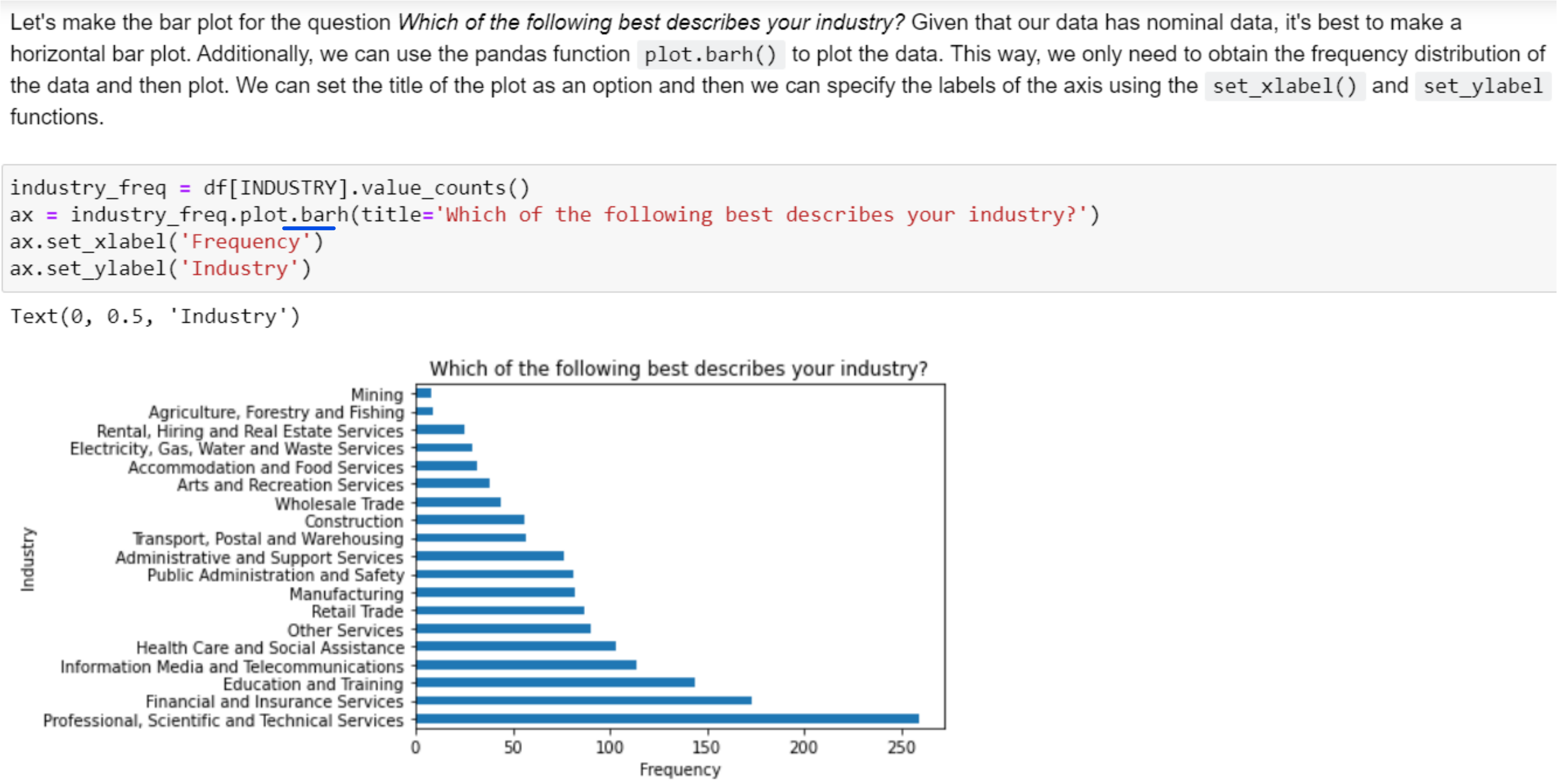




```
import matplotlib.pyplot as plt

plt.hist(df[YEAR_BORN], bins = 5, rwidth=0.8)
plt.ylabel('Number of responses')
plt.xlabel('Year')
plt.title('What year were you born?')
plt.show()
```



Creating a bar chart



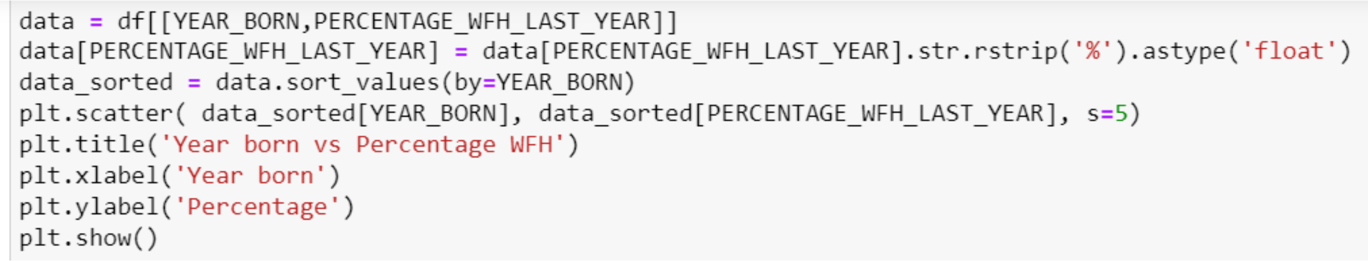
Let's make the bar plot for the question *Which of the following best describes your industry?* Given that our data has nominal data, it's best to make a horizontal bar plot. Additionally, we can use the pandas function `plot.barh()` to plot the data. This way, we only need to obtain the frequency distribution of the data and then plot. We can set the title of the plot as an option and then we can specify the labels of the axis using the `set_xlabel()` and `set_ylabel` functions.

```
industry_freq = df[INDUSTRY].value_counts()
ax = industry_freq.plot.barh(title='Which of the following best describes your industry?')
ax.set_xlabel('Frequency')
ax.set_ylabel('Industry')
```

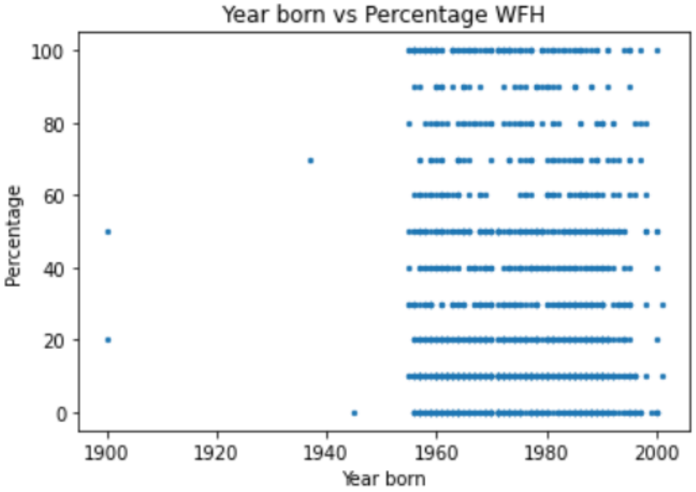
```
Text(0, 0.5, 'Industry')
```

Which of the following best describes your industry?
Mining
Agriculture, Forestry and Fishing
Rental, Hiring and Real Estate Services
Electricity, Gas, Water and Waste Services
Accommodation and Food Services
Arts and Recreation Services
Wholesale Trade
Construction
Transport, Postal and Warehousing
Administrative and Support Services
Public Administration and Safety
Manufacturing
Retail Trade
Other Services
Health Care and Social Assistance
Information Media and Telecommunications
Education and Training
Financial and Insurance Services
Professional, Scientific and Technical Services
Industry
0
50
100
150
200
250
Frequency

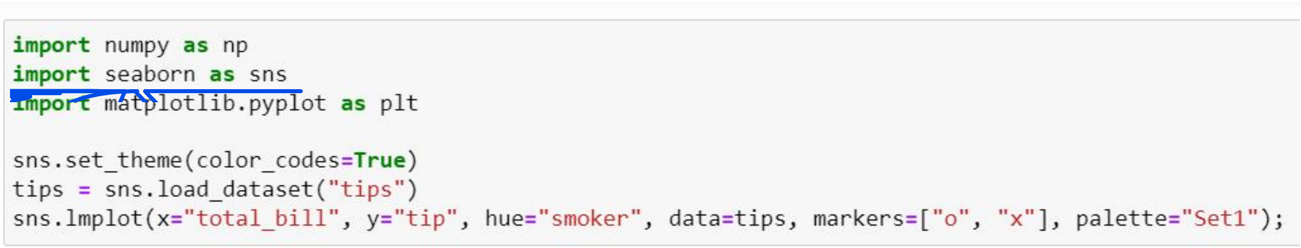
Creating a scatter plot



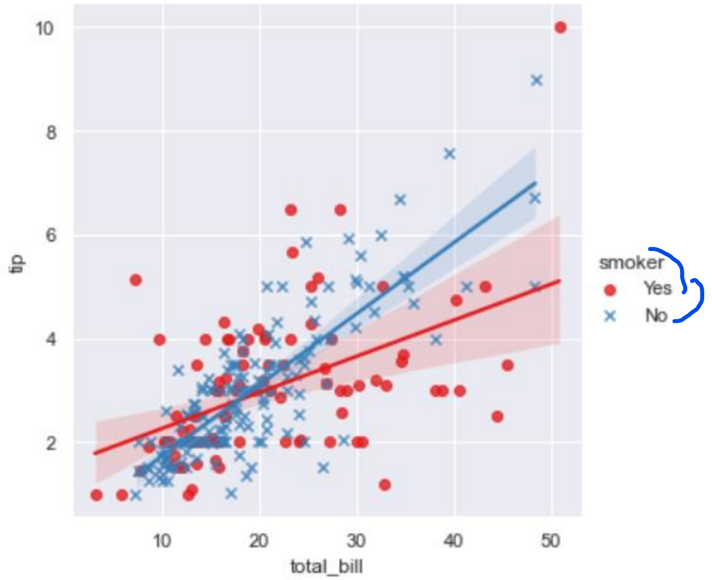
```
data = df[[YEAR_BORN,PERCENTAGE_WFH_LAST_YEAR]]
data[PERCENTAGE_WFH_LAST_YEAR] = data[PERCENTAGE_WFH_LAST_YEAR].str.rstrip('%').astype('float')
data_sorted = data.sort_values(by=YEAR_BORN)
plt.scatter( data_sorted[YEAR_BORN], data_sorted[PERCENTAGE_WFH_LAST_YEAR], s=5)
plt.title('Year born vs Percentage WFH')
plt.xlabel('Year born')
plt.ylabel('Percentage')
plt.show()
```



Creating a scatter plot



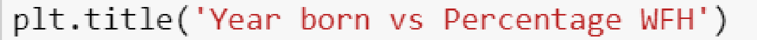
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme(color_codes=True)
tips = sns.load_dataset("tips")
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips, markers=["o", "x"], palette="Set1");
```



2 Categories

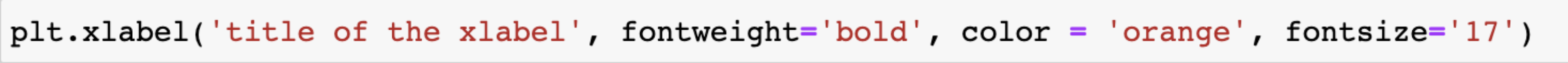
Customise plots

- Customise plot title:
  - title()



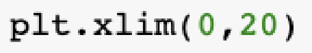
```
plt.title('Year born vs Percentage WFH')
```

- Customise axis titles:
  - xlabel() and ylabel()



```
plt.xlabel('title of the xlabel', fontweight='bold', color = 'orange', fontsize='17')
```

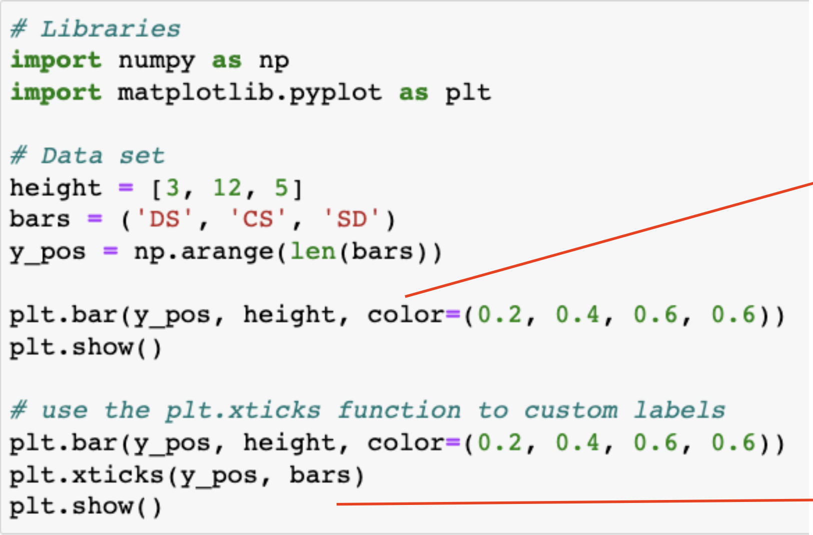
- Change axis limits:
  - xlim() and ylim().



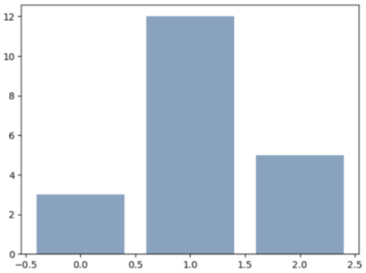
```
plt.xlim(0,20)
```

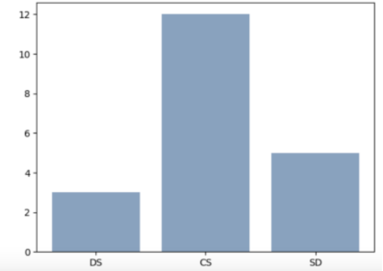
Customise plots

- Customize Axis Tick Labels:
  - x_ticks() and y_ticks().



```
# Libraries
import numpy as np
import matplotlib.pyplot as plt

# Data set
height = [3, 12, 5]
bars = ('DS', 'CS', 'SD')
y_pos = np.arange(len(bars))

plt.bar(y_pos, height, color=(0.2, 0.4, 0.6, 0.6))
plt.show()

# use the plt.xticks function to custom labels
plt.bar(y_pos, height, color=(0.2, 0.4, 0.6, 0.6))
plt.xticks(y_pos, bars)
plt.show()
```



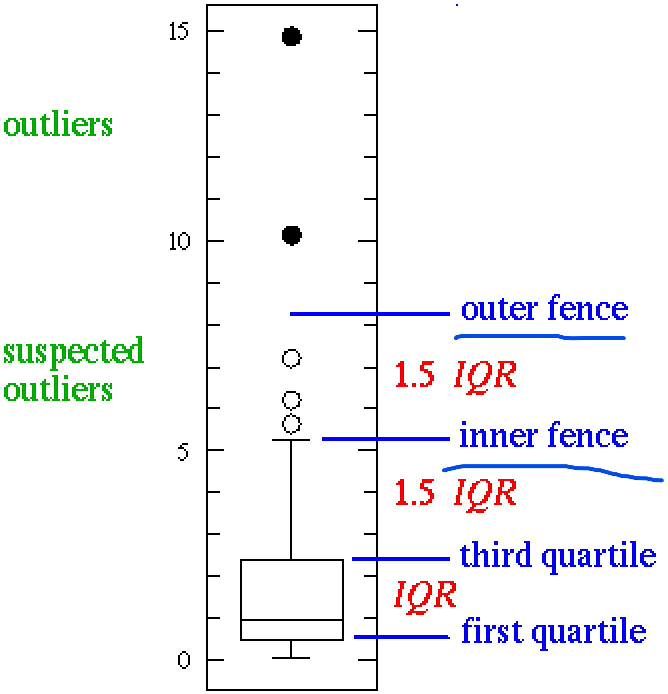


BOX PLOTS AND CORRELATION

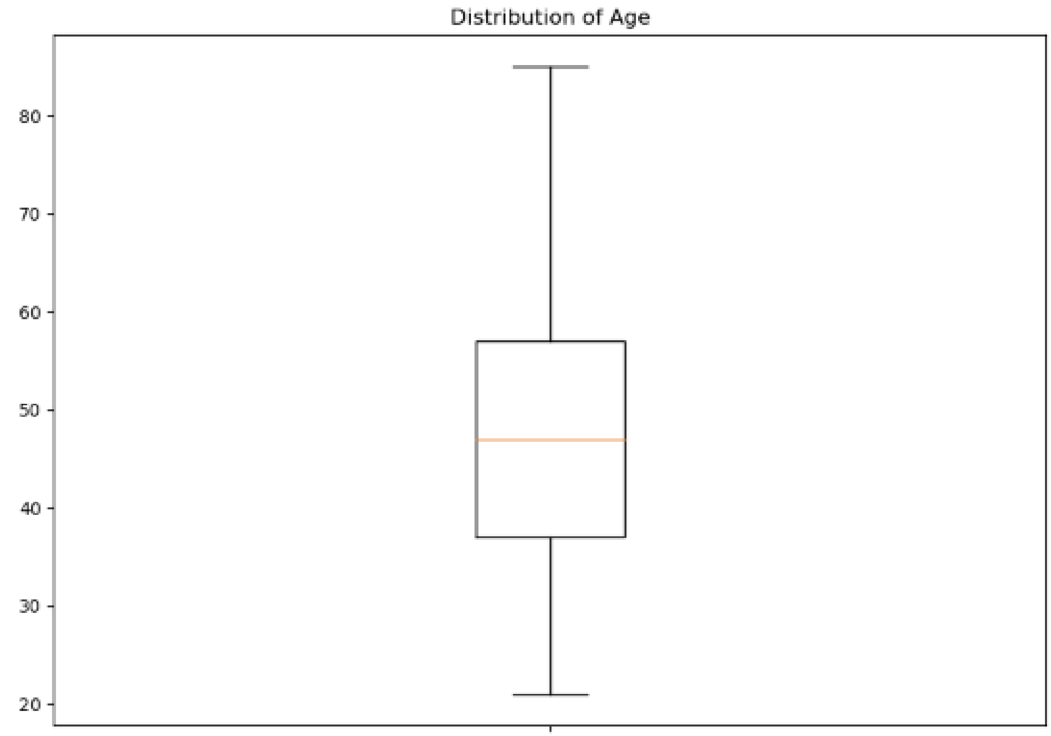
Using boxplots to compare distributions

- 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 25th percentile.
  - Median (Q2) – equivalent to 50th percentile.
  - Third quartile (Q3) – equivalent to 75th percentile.
  - Upper inner fence – Q3+1.5*IQR.
- Values outside fences are outliers.
- Sometimes include outer fences at Q1-3*IQR and Q3+3*IQR.

Box plots illustrated

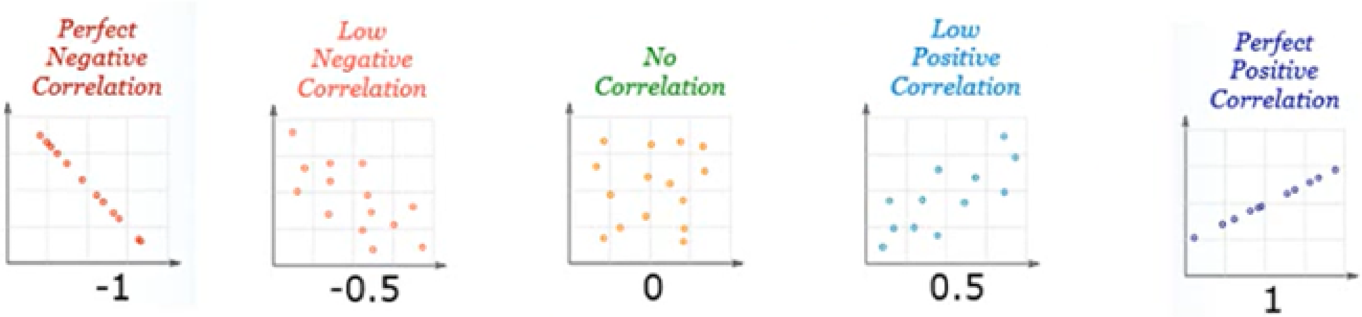


A box plot for the age distribution



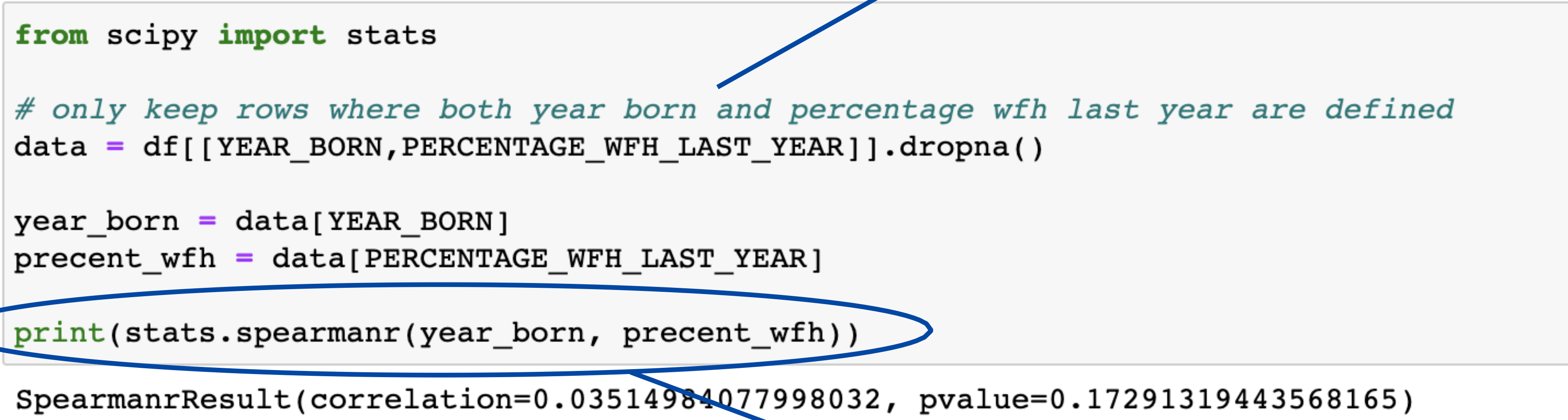
Using correlation statistics to measure dependence

- Using correlation statistics to measure dependence.
- Scipy includes various correlation statistics.
  - Pearson's *r* for two normally distributed variables.
  - 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 year born and percentage wfh last year are defined
data = df[[YEAR_BORN,PERCENTAGE_WFH_LAST_YEAR]].dropna()

year_born = data[YEAR_BORN]
precent_wfh = data[PERCENTAGE_WFH_LAST_YEAR]

print(stats.spearmanr(year_born, precent_wfh))
```

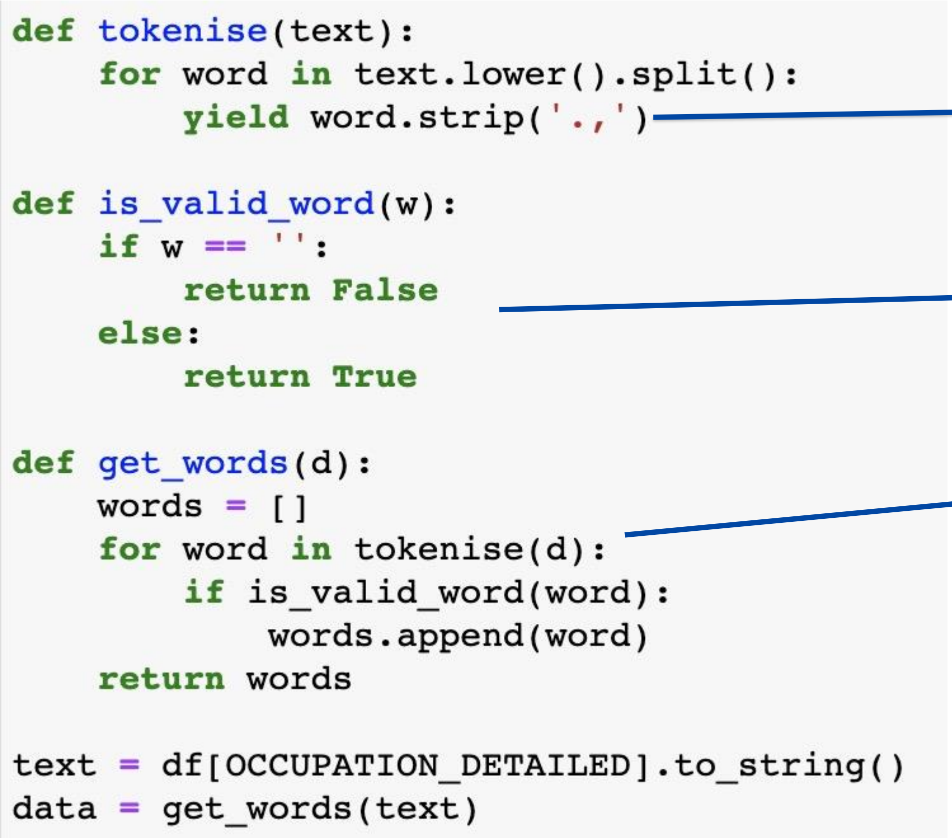
```
SpearmanrResult(correlation=0.03514984077998032, pvalue=0.1729131944356816 5)
```

Calculate Spearman’s rho

TEXT DATA

THE UNIVERSITY OF SYDNEY

A simple whitespace tokeniser



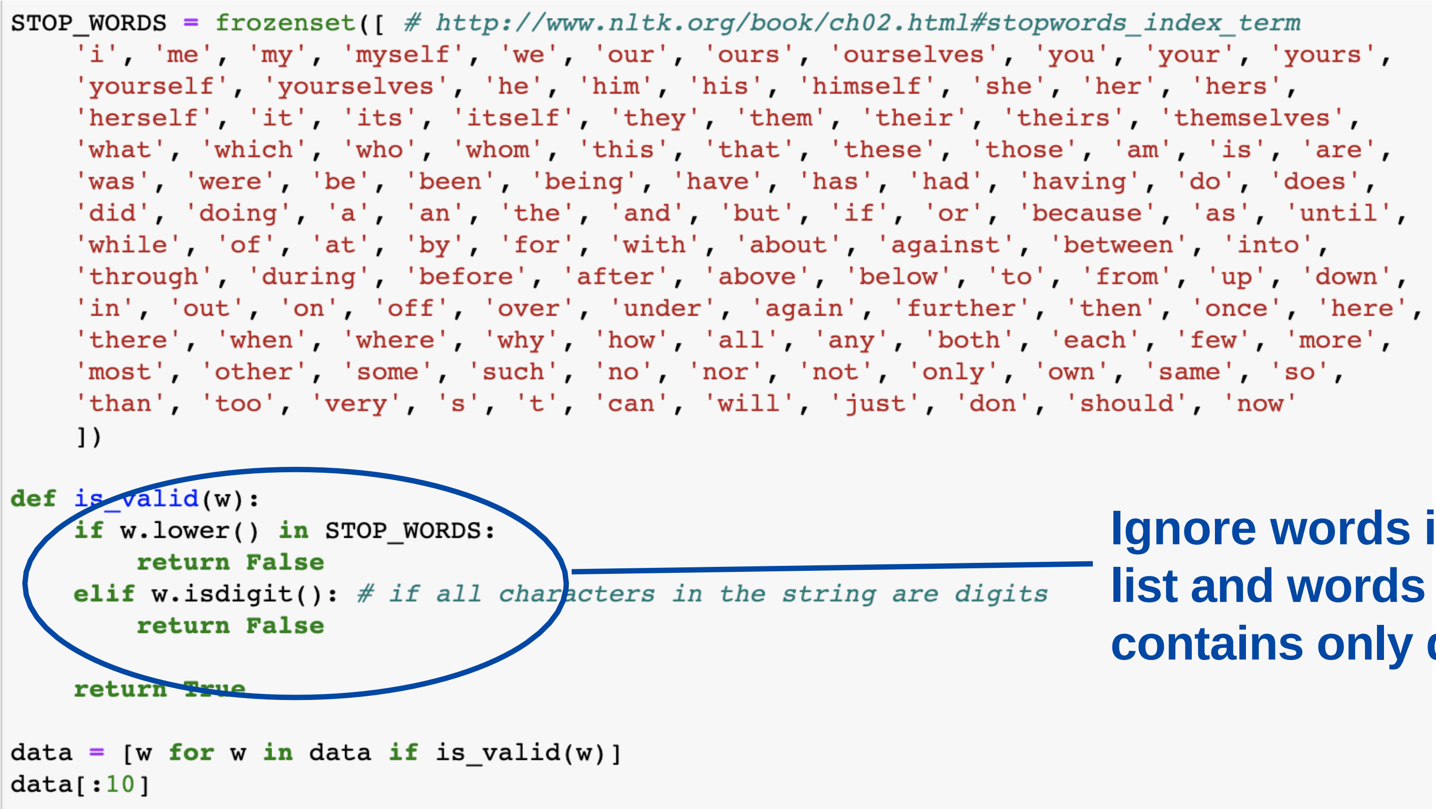
```python
def tokenise(text):
    for word in text.lower().split():
        yield word.strip('.,')

def is_valid_word(w):
    if w == '':
        return False
    else:
        return True

def get_words(d):
    words = []
    for word in tokenise(d):
        if is_valid_word(word):
            words.append(word)
    return words

text = df[OCCUPATION_DETAILED].to_string()
data = get_words(text)
```

Convert text string to lower case and split on whitespace
Remove leading/trailing ‘.’ and ‘,’

Ignore empty strings

Get each word token

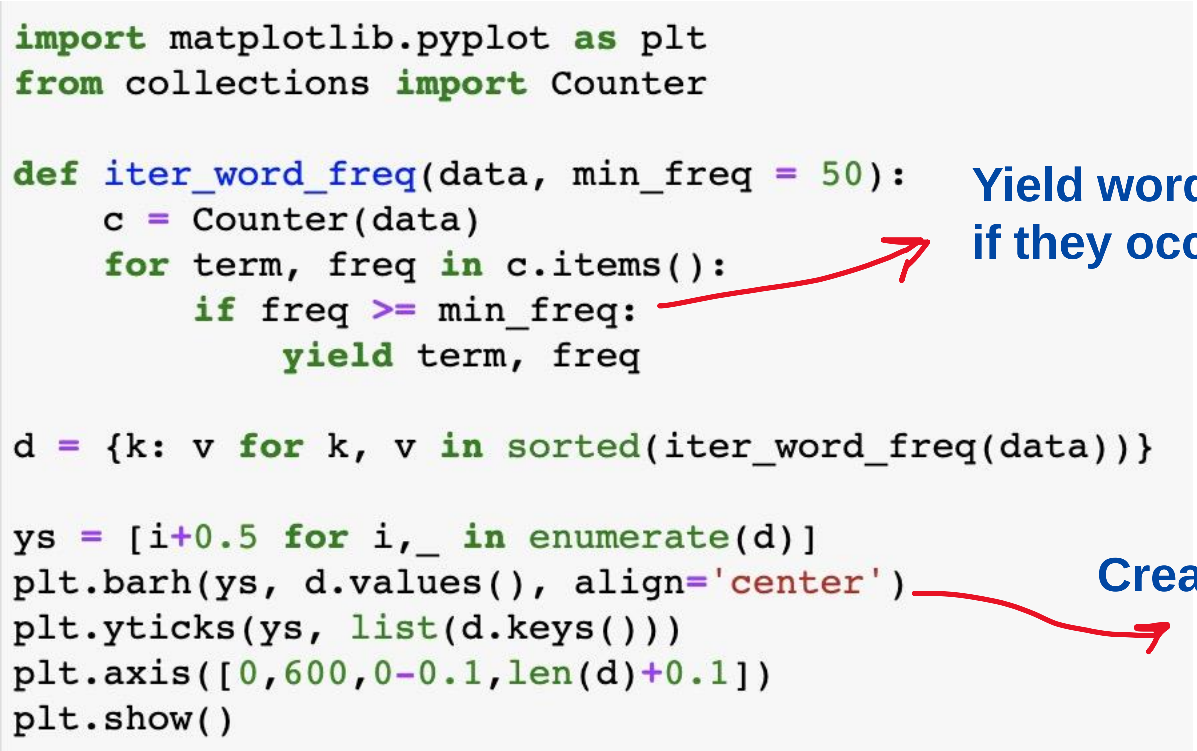
Removing stop words



```
STOP_WORDS = frozenset([ # http://www.nltk.org/book/ch02.html#stopwords_index_term
    'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours',
    'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
    'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
    'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',
    'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',
    'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
    'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
    'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',
    'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
    'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
    'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now'
    ])

def is_valid(w):
    if w.lower() in STOP_WORDS:
        return False
    elif w.isdigit(): # if all characters in the string are digits
        return False

    return True

data = [w for w in data if is_valid(w)]
data[:10]
```

Ignore words in stop list and words contains only digits

Plotting most frequent words

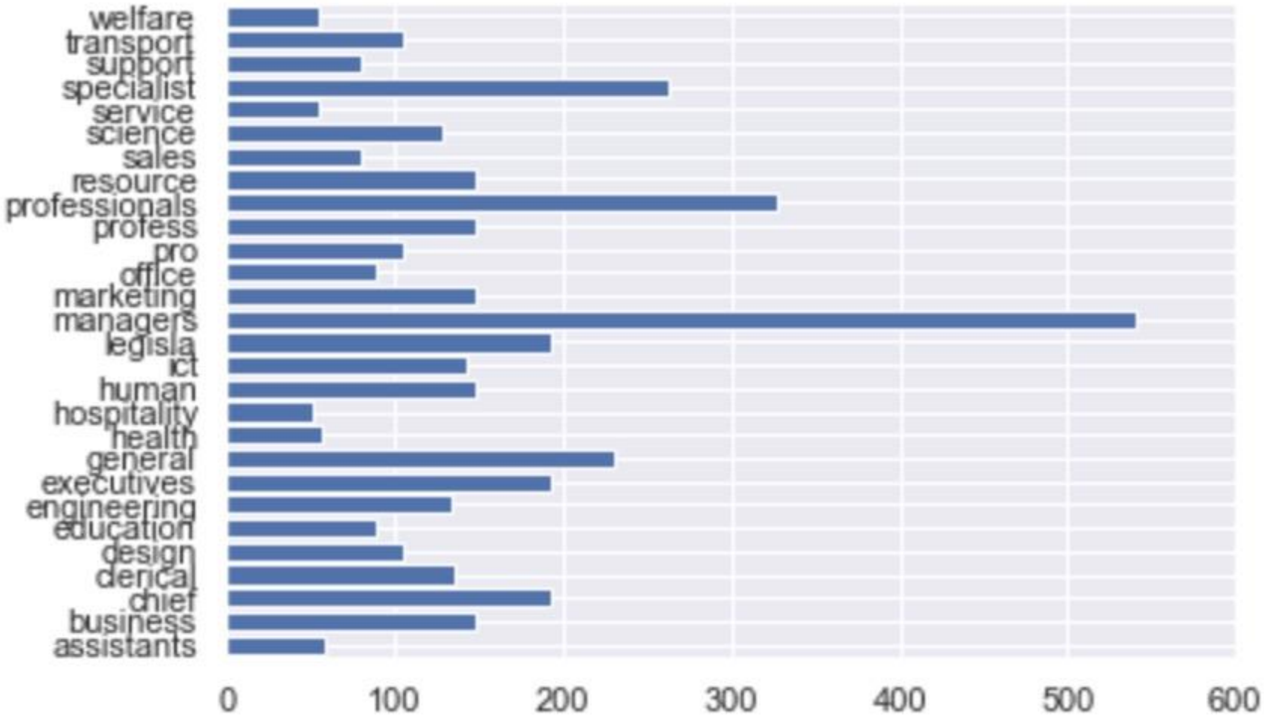


```
import matplotlib.pyplot as plt
from collections import Counter

def iter_word_freq(data, min_freq = 50):
    c = Counter(data)
    for term, freq in c.items():
        if freq >= min_freq:
            yield term, freq

d = {k: v for k, v in sorted(iter_word_freq(data))}

ys = [i+0.5 for i,_ in enumerate(d)]
plt.barh(ys, d.values(), align='center')
plt.yticks(ys, list(d.keys()))
plt.axis([0,600,0-0.1,len(d)+0.1])
plt.show()
```

Yield words and their frequencies if they occur 50 or more times

Create a horizontal bar chart

A term frequency bar chart



REVIEW

Notes

- Python is 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.

W3 Review: Data Exploration with Python

Objective

- Learn Python tools for exploring a new data set programmatically.

Lecture

- Pandas
- Descriptive statistics, e.g., median, quartiles, IQR, outliers.
- Descriptive visualisation, e.g., boxplots.

Readings

- Data Science from Scratch: Ch 5

Exercises

- matplotlib: Visualisation.
- pandas: Descriptive stats.

TO-DO in W3

- Ed Lessons Python modules 7-9.
- Ed Lessons SQL modules 18-19.