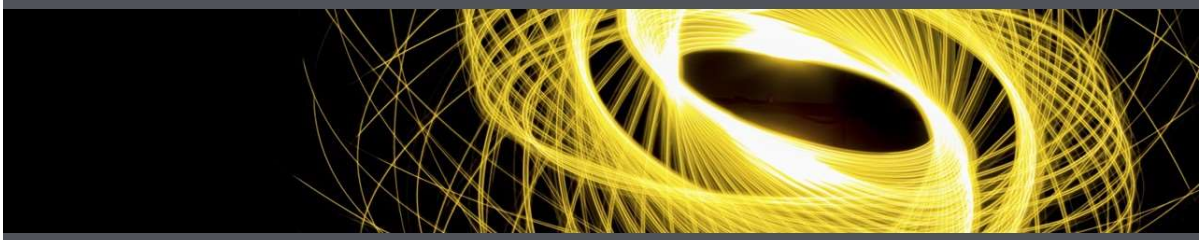
Department of Computer Science

University of Reading

CSMAD21 – Applied Data Science with Python



Pandas

1

University of Reading

Lecture Objectives

- Differentiate, interact, implement and modify the data structures available in Pandas (Series and Dataframes).
- Perform pre-processing activities and data manipulation.
- Implement basic data analysis.

2

University of Reading

Outline

- Data Structures:
  - Series
  - Dataframes
- Indexing Dataframes
  - Rows
  - Columns
  - Subsets
- Data Processing – Missing Data
- Vectorisation
- Data Analysis
  - Describe
  - Unique
- Wide and Long Data
- Group By
- Combining Dataframes
- Time series

3

University of Reading

Pandas – General Concepts

- Pandas has two main data structures:
  - Series in pandas are one dimensional arrays of data.
  - Data frames in pandas store two dimensional arrays
- Both are indexed using labels to refer to specific locations or subsets of the array.

4

Import

Generally the pandas module is imported as:

- **import pandas as pd**

So that it can be used with less typing:

In [2]:
```
import pandas as pd
x = pd.DataFrame()
x
```

Out[2]: —

University of Reading

Pandas Data Structures - Series

- Series in pandas are one dimensional arrays where each index in the array is labelled. They are constructed using:

pd.Series(data,index=index)

- If no index is given, then one is constructed using 0,...,len(data)-1. The data argument must be either an array or list, a dictionary or a scalar value (integer or float).

5

In [ ]:
```
sr1 = pd.Series([3,4,5],index=['one','two', 'three'])
sr1
```

- Series can be constructed as a dictionary by specifying the value to be stored at each location.

In [ ]:
```
dc1 = { 'one': 1, 'two': 2, 'three': 3, 'four':4}
sr2 = pd.Series(dc1)
sr2
```

Indexig

Unlike a dictionary, Series in pandas have an ordering. Values can be accessed both by their index and also by a numeric index:

In [ ]:
```
dc1['one']
#dc1
```

In [ ]:
```
sr2
sr2[0]
```

In [ ]:
```
sr2['one']
```

In [ ]:
```
sr2[0:2]
```

In [ ]:
```
sr2['one':'two']
```

University of Reading

Pandas Data Structures – Dataframes

- Data can be stored in tables with one column for each variable, and one row for each object.

Columns

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

6

University of Reading

Pandas Data Structures – Dataframes

- Data can be stored in tables with one column for each variable, and one row for each object.

Rows

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

7

University of Reading

Pandas Data Structures – Dataframes

- A data frame is a tabular data structure for storing data that have multiple attributes. It is similar to a matrix or array, but each column can have a different data type.
- Just as with a Series, every row in a data frame has an index.
- They are constructed using:

pandas.DataFrame(data=None, index=None, columns=None)

8

In [ ]:
```
## Our first dataframe:
df1 = pd.DataFrame(data = ([1,2,3,4,5]), columns = ['column_1'])
df1
```

DataFrame - From a Dictionarie

- Data frames can be built by using Dictionaries, with each index making up a column of the data frame.

In [ ]:
```
##First wwe need a dictionary
dc2 = {'column_1': [1,2,3,4,5], 'column_2': ['a','b','c','d','e']}
dc2
#type(dc2)
```

In [ ]:
```
##We use the dictionary to define the dataset
df2 = pd.DataFrame(dc2)
df2
```

DataFrame - Reading data with pandas – CSV files

- Tabular data is often stored in text files, with one line per row of the table, and characters separating the columns.
- Reading a data frame from a comma separated value (CSV) file is easy in pandas :

In [7]:
```
diamonds = pd.read_csv("Datasets/diamonds.csv")
diamonds.head() ###you can use head to visualice, by default, the first 5 rows of a
```

Out[7]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31

	carat	cut	color	clarity	depth	table	price	x	y	z
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

- You can specify the separator used to format the data, read_csv will default to a comma (','), and read_table will default to a tab ('\t').

DataFrame - Reading data with pandas – HTML

- Pandas can also scrape tables from HTML given a URL using pd.read_html. It returns a Python list of all tables scraped from the web page.

In [ ]:
```
url = ("http://www.reading.ac.uk/modules/module.aspx?sacyr=1819&school=MPS")
modules = pd.read_html(url)
type(modules)
```

In [ ]:
```
modules[0].head()
```

University of Reading

Dataframes - Indexing

- Indexing by Columns
- Indexing by Rows
- Indexing Subsets

Warning!

Beware that objects returned by indexing a subset of a Series or Data Frame in pandas are (generally) **views** of the original object, not copies, and modifications will change the corresponding elements in the original Series or Data Frame.

9

Indexig in Pandas - Columns

- You can choose a subset columns in your data frame by just indexing into it using the column name:

In [ ]:
```
#diamonds.head()
diamonds['cut'].head()
diamonds.cut.head()
type(diamonds.cut.head())
```

- **These both return Series objects.**

- It is also possible to select a list of columns and return a data frame by providing a list of column names:

In [ ]:
```
diamonds[['cut','depth','carat']].head()
#type(diamonds[['cut','depth','carat']].head())
```

Indexig in Pandas - Rows

- Data frames have a special **loc** attribute that allows for indexing. Whole rows can be selected by specifying : as the columns.

In [ ]:
```
diamonds.loc[2]
```

In [ ]:
```
diamonds.loc[2:4]
```

Indexig in Pandas - Subsets

- The **loc** attribute can take a subset of rows and columns

In [ ]:
```
diamonds.loc[2:4,'cut':'depth']
```

In [ ]:
```
diamonds.loc[2:4,['cut','price']]
```

- To use integers to index into a pandas data frame as you would with a more traditional array, you can use the **iloc** attribute:

In [ ]:
```
diamonds.head()
diamonds.iloc[2:5,1:6]
```

In [ ]:
```
diamonds.iloc[2:5,[1,2,3,5]]
diamonds.iloc[2:5,[3,2,1,5]]
```

- **Warning!**: Beware that objects returned by indexing a subset of a Series or Data Frame in pandas are (generally) views of the original object, not copies, and modifications will change the corresponding elements in the original Series or Data Frame.

In [ ]:
```
##Lets make a DataFrame
dc2 = {'column_1': [1,2,3,4,5], 'column_2': ['a','b','c','d','e']}
df2 = pd.DataFrame(dc2)
df2
```

In [ ]:
```
##We assign our DataFrame to a new one
df2_1 = df2
df2_1
```

In [ ]:
```
##If we remove one column from df2_1.....
```

```
del df2_1['column_2']
df2_1
```

In [ ]:
```
##The original DataFrame df2 is also affected.
df2
```

In [ ]:
```
##You can prevent this by defining .copy
```

In [ ]:
```
##Lets make a DataFrame
dc2 = {'column_1': [1,2,3,4,5], 'column_2': ['a','b','c','d','e']}
df2 = pd.DataFrame(dc2)
df2
```

In [ ]:
```
##We assign our DataFrame to a new one but now it is a copy
df2_1 = df2.copy()
df2_1
```

In [ ]:
```
##If we remove one column from df2_1.....
del df2_1['column_2']
df2_1
```

In [ ]:
```
##The original DataFrame df2 is NOT affected.
df2
```

In [ ]:
```

```

University of Reading

Data Processing – Missing Data

- Pandas represents missing numeric values using NaN (not a number). There are several methods to detect and drop missing values from a data frame.

- **dropna()** – Drop some rows or columns if they contain missing data.
- **isnull()** –Generates a data structure of Boolean sindicating whether each value is missing.
- **fillna()** – Fills in missing values.

10

As an example, a create a dataframe with some missing values:

In [ ]:
```
import numpy as np
```

```
dogs_dc = {'Name': ['Tom','Lupita','Olivia',np.nan,'Marcel'], 'Age': [5,4,1,10,np.na
dogs_df = pd.DataFrame(dogs_dc)
dogs_df
```

- Checking which values are missing:

In [ ]:
```
dogs_df.isnull()
```

Dropna

Dropping any row with a missing value:

In [ ]:
```
dogs_df.dropna()
```

In [ ]:
```
##it does not affect the original DataFrame
dogs_df
```

Fill Na

Filling in all missing values:

In [ ]:
```
dogs_df.fillna(0)
```

In [ ]:
```
dogs_df
```

Filling in missing values with column specific values:

In [ ]:
```
dogs_df.fillna({'Name': 'Doggy', 'Age': 7, 'Breed':'Mutt'})
```

University of Reading

Vectorisation

- Vectorisation is when we “broadcast” an operation over a vector or array of data.
- It is important to learn the distinction between vectorised element-wise operations and matrix style operations when manipulating data. Vectorised element-wise operations work on individual values of data frames or series one by one.
- All the mathematical operations you expect can be applied element-wise in pandas.
- This also works for columns in a data frame.

11

Series:

In [ ]:
```
a = pd.Series([1,2,3,4])
b = pd.Series([2,3,4,1])
```

In [ ]:
```
a + b
```

In [ ]:
```
a * b
```

Dataframes:

In [ ]:
```
diamonds['price/carat'] = diamonds.price/diamonds.carat
diamonds.head()
```

- Here we are creating a new column in the data frame called price/carat and assigning the newly calculated column to it.

University of Reading

Data Analysis – Describe

- Data frames in pandas have a describe method that will calculate various statistics for each column in the data frame:
- Notice that these statistics only make sense for numerical variables, **so the columns with other forms of data have been omitted.**

12

In [ ]:
```
diamonds.describe()
```

In [ ]:
```
diamonds.head()
```

University of Reading

Data Analysis – Describe

- Notice that these statistics only make sense for numerical variables, **so the columns with other forms of data have been omitted.**
- This gives us various statistics of the data:
  - **count** total number of values in column
  - **mean** average of values in column
  - **std** standard deviation of values in column
  - **min** minimum value in the column
  - **25%** 1st quartile, 25% of the values are less than this value
  - **50%** median, 50% of the values are less than this value
  - **75%** 3rd quartile, 75% of the values are less than this value
  - **max** maximum value in the column
- Not every numeric value can be meaningfully analysed in this way (e.g. mean of id).

13

Data Analysis - Summarasing

Unique

The unique method will list unique values in a series:

In [ ]:
```
diamonds['cut'].unique()
```

- We can use the value_counts method of a series to count how many times each unique vaule occurs:

In [ ]:
```
diamonds['cut'].value_counts()
```

University of Reading

Wide and Long Data

- So far we have been working with wide data organised with columns corresponding to variables or attributes, and rows corresponding to the objects we are collecting data about.

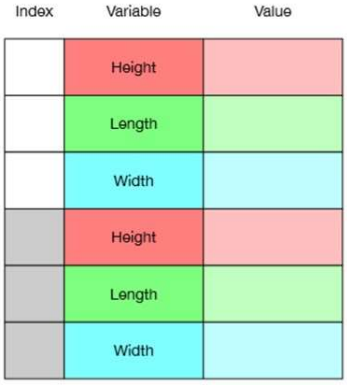
14

In [ ]:
```
##An example of the wide data is the diamonds dataset
diamonds.head()
```

University of Reading

Wide and Long Data

- Another way of organising the same data is to store it as "long" data, where each row stores the variable name and value, and optionally a corresponding key.



15

In [ ]:
```
##Lest transform the diamonds dataset in Long Data
diamonds.melt(id_vars=['cut'], value_vars =['carat','depth', 'color'])
#diamonds.melt(id_vars=['cut'], value_vars =['carat','depth', 'color']).tail()
```

University of Reading

Group By

- We can take data in pandas and group rows together based on the values in a specific column.

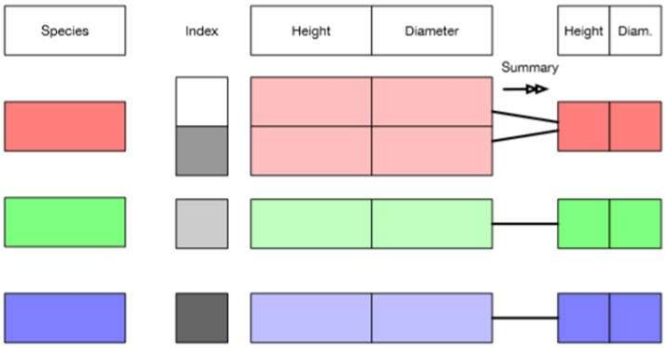
- This is an example data frame where we want to group together rows based on the value in the species column.

16

University of Reading

Group By

- We can group rows together based on the values in a specific column. We can then take these groups and apply functions to summarise the values in each group.



17

Example: from our diamonds data set we can group the diamonds (rows) together into groups all having the same name:

This will group the diamonds data set into groups all having the same common cut. The resulting object is a pandas GroupBy object.

In [8]:
```
diamonds.groupby('cut')
```

Out[8]:
```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x00000184FE566CC0>
```

In [9]:
```
diamonds.groupby('cut')[['price']].mean()
```

Out[9]:

cut	price
Fair	4358.757764
Good	3928.864452
Ideal	3457.541970
Premium	4584.257704
Very Good	3981.759891

- We could also group the data on multiple columns.
- This returns a GroupBy object with groups for each unique combination of genus name and species name. We can calculate summary statistics of selected columns of the grouped data:

In [10]:
```
diamonds.groupby(['cut','clarity'])
```

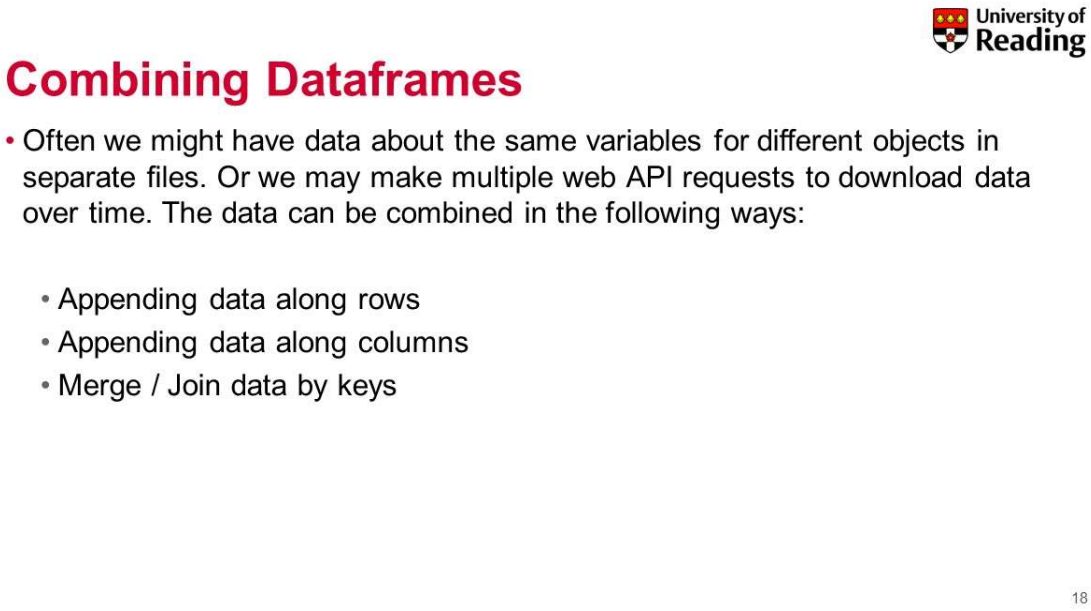
Out[10]:
```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x00000184FE5BEB38>
```

In [11]:
```
diamonds.groupby(['cut','clarity'])[['price', 'carat']].mean()
```

Out[11]:

cut	clarity	price	carat
Fair	**I1**	3703.533333	1.361000
	IF	1912.333333	0.474444
	SI1	4208.279412	0.964632
	SI2	5173.916309	1.203841
	VS1	4165.141176	0.879824
	VS2	4174.724138	0.885249
	VVS1	3871.352941	0.664706
	VVS2	3349.768116	0.691594
Good	**I1**	3596.635417	1.203021
	IF	4098.323944	0.616338
	SI1	3689.533333	0.830397
	SI2	4580.260870	1.035227
	VS1	3801.445988	0.757685
	VS2	4262.236196	0.850787
	VVS1	2254.774194	0.502312
	VVS2	3079.108392	0.614930
Ideal	**I1**	4335.726027	1.222671
	IF	2272.913366	0.455041
	SI1	3752.118169	0.801808
	SI2	4755.952656	1.007925
	VS1	3489.744497	0.674714
	VS2	3284.550385	0.670566
	VVS1	2468.129458	0.495960
	VVS2	3250.290100	0.586213
Premium	**I1**	3947.331707	1.287024
	IF	3856.143478	0.603478
	SI1	4455.269371	0.908601
	SI2	5545.936928	1.144161
	VS1	4485.462041	0.793308
	VS2	4550.331248	0.833774
	VVS1	2831.206169	0.534821
	VVS2	3795.122989	0.654724
Very Good	**I1**	4078.226190	1.281905
	IF	4396.216418	0.618769

cut	clarity	price	carat
	SI1	3932.391049	0.845978
	SI2	4988.688095	1.064338
	VS1	3805.353239	0.733307
	VS2	4215.759552	0.811181
	VVS1	2459.441065	0.494588
	VVS2	3037.765182	0.566389



We are going to see an example of each case, but first, lets create some dataframes.

In [ ]:
```
##Sample dataframes
dogs_df1 = pd.DataFrame({'Name': ['Tom','Lupita','Olivia','Loki','Marcel'], 'Age': [
dogs_df1
```

In [ ]:
```
dogs_df2 = pd.DataFrame({'Name': ['Doddy','Tequila','Gin','Cookie','Spotty'], 'Age':
dogs_df2
```

In [ ]:
```
dogs_df3 = pd.DataFrame({'Name': ['Lupita','Tequila','Gin','Loki'], 'Colour': ['Red'
dogs_df3
```

Appending data along rows

In [ ]:
```
pd.concat([dogs_df1, dogs_df2])
```

Appending data along columns

In [ ]:
```
pd.concat([dogs_df1, dogs_df2], axis = 1)
```

Mergin/join data by keys

Inner Join

In [ ]:
```
pd.merge(dogs_df1, dogs_df3, on='Name')
```

In [ ]:
```
pd.merge(dogs_df2, dogs_df3, on='Name')
```

Left/Right Join

In [ ]:
```
pd.merge(dogs_df1, dogs_df3, on='Name', how = 'left')
```

In [ ]:
```
pd.merge(dogs_df1, dogs_df3, on='Name', how = 'right')
```

Outer Join

In [ ]:
```
pd.merge(dogs_df1, dogs_df3, on='Name', how='outer')
```

University of Reading

Data Manipulation - Selection/Filtering Data

- Pandas allows to filter or select data by te command **loc/iloc**. This operation is similar to a WHERE clause in the SQL language. It follows this structure:
  - **data.loc/iloc[row_selection, column_slection]**
- The row selection area is where the comparison opperators(==, !=, >, etc) and/or logical are placed.

19

We are going to run some examples on the diamonds dataset.

In [ ]:
```
diamonds.head()
```

In [ ]:
```
diamonds.loc[diamonds['cut'] == 'Ideal'].head()
#len(diamonds), len(diamonds.loc[diamonds['cut'] == 'Ideal'])
```

In [ ]:
```
diamonds.loc[(diamonds['cut'] == 'Ideal')&(diamonds['color'] == 'E')].head()
```

University of Reading

Time Series

- When working with time series data, it is useful to have a data type for storing dates and times. Although Python has it's own module for this, we will use the pandas Timestamp data type.

20

University of Reading

Time Series – Resampling

- We might be interested in looking at how values vary at a different frequency than we have recorded them. pandas allows us to easily resample time series to change the frequency at which values are recorded.
- The most straightforward application of this is to *downsample* a time series. We use the resample method and specify how we would like to reduce the time points for a given time period down to a single value.
- In pandas you can easily specify various time periods:
  - **Y** Yearly M Monthly
  - **W** Weekly
  - **D** Daily H Hourly
  - **5min** Every 5 minutes

21

In [ ]:

```
###Lets create a dataframe that describes the amount of coffe sold in a 24hrs cafete

##Making time instances of 30 min difference
time_frame = pd.date_range('2020-03-01', '2020-03-31', freq='30min')
##Adding the time instances as index in a dataframe
df = pd.DataFrame(index = time_frame)
##Creating some sales sample data (np methods that we reviewed last week)
df['coffe_sales'] = np.random.randint(low=0, high=10, size=len(df.index))

##Looking at the data:
df.head()
len(df)
```

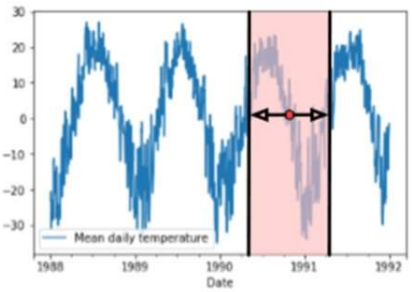
In [ ]:

In [ ]:
```
###Lest resamplethe data by Week:
df.coffe_sales.resample('W').sum()
```

University of Reading

Time Series – Rolling Windows

- As well as resampling the time series data, we can also use something called a rolling window. This calculates a new value at each time point by considering a window of time points to either side.
- We need to specify the window to use, and the function to use to reduce the values in the window down to a single value.



22

In [112…
```
##Following the same dataset as befor "Caffeteria coffe sales in a month" lets defin
##Loking at the original data:
df.head(10)
```

Out[112…

	coffe_sales
2020-03-01 00:00:00	1
2020-03-01 00:30:00	0
2020-03-01 01:00:00	2
2020-03-01 01:30:00	6
2020-03-01 02:00:00	0
2020-03-01 02:30:00	6
2020-03-01 03:00:00	1
2020-03-01 03:30:00	2
2020-03-01 04:00:00	8
2020-03-01 04:30:00	3

In [113…
```
##Defining rolling windows sum of 5
df.coffe_sales.rolling(5).sum().head(10)
```

Out[113…
```
2020-03-01 00:00:00     NaN
2020-03-01 00:30:00     NaN
2020-03-01 01:00:00     NaN
2020-03-01 01:30:00     NaN
2020-03-01 02:00:00     9.0
```

```
2020-03-01 02:30:00    14.0
2020-03-01 03:00:00    15.0
2020-03-01 03:30:00    15.0
2020-03-01 04:00:00    17.0
2020-03-01 04:30:00    20.0
Freq: 30T, Name: coffe_sales, dtype: float64
```

University of Reading

Questions

24

University of Reading

Summary

- Pandas is a library that provides data structures(series and dataframes) that allows to analyse the data in a natural way.
- Pandas can represent vectors (series) and tabular data (dataframes).
- It provides methods to combine data.
- It allows to represent time series
- Data analysis methods such as Unique, Describe and Group By can be implemented to do data discovery.
- Pandas provides methods to do data processing.

23

University of Reading

References

- Thomas McKinney, Wes 'Python for data analysis : data wrangling with Pandas, NumPy, and IPython'.

25

---

https://pandas.pydata.org/pandas-docs/stable/index.html