

pandas_tutorial

February 6, 2025

1 Introduction to pandas

This notebook demonstrates some key functionality of the pandas package. See: <https://pandas.pydata.org/>

See here for getting started documentation: https://pandas.pydata.org/docs/getting_started/index.html

You may find these code examples useful: <https://github.com/paskhaver/pandas-in-action>

For more information on the book Pandas in Action by Boris Paskhaver, see: <https://www.manning.com/books/pandas-in-action>

Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter 3rd Edition, by Wes McKinney is a highly recommended reference. The open edition is here: <https://wesmckinney.com/book/> - the code examples for this book are here: <https://github.com/wesm/pydata-book/tree/3rd-edition>

```
[1]: import pandas as pd
```

1.1 Load our data set

See https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html for more info on read_csv.

```
[2]: # You may need to modify the filename depending on where you have stored the ↵
      ↵ file.
      students_df = pd.read_csv('./data/students.csv')
```

After loading the dataset, we can easily view a representation of it.

```
[3]: students_df # Notice there's no need to call print
```

```
[3]:
```

	Student ID	Name	Age	Subject	Year of Study	\
0	2703f3f0	Mr Clifford Watson	25.0	English Literature	1.0	
1	a8040287	Elliott Ward	25.0	Computer Science	4.0	
2	d8da5486	Miss Pauline Dunn	22.0	Engineering	4.0	
3	3ac1b74d	Mr Dominic Mason	22.0	Physics	1.0	
4	67850858	Mrs Melanie Brown	18.0	English Literature	3.0	
..	
96	a8be1ec3	Kelly Foster	22.0	Engineering	1.0	
97	3b69ff22	Sara Austin	19.0	Computer Science	34.0	

98	716fb45f	Miss Grace Miller	22.0	English Literature	4.0
99	34b97db2	Miss Lydia Saunders	23.0	Physics	2.0
100	34b97db2	Miss Lydia Saunders	23.0	Physics	2.0

	Country of Origin
0	Saint Barthelemy
1	Guinea
2	Afghanistan
3	Palau
4	Algeria
..	...
96	Netherlands
97	Liechtenstein
98	Comoros
99	Faroe Islands
100	Faroe Islands

[101 rows x 6 columns]

The read_csv function automatically created an index column - you can see the numbers 0 to 100 in the DataFrame above. We can view the index as follows:

```
[4]: students_df.index
```

```
[4]: RangeIndex(start=0, stop=101, step=1)
```

This dataset already has a Student ID column which we could specify as the index column.

```
[5]: students_df = pd.read_csv('./data/students.csv', index_col='Student ID')
```

Now when we view the DataFrame, we can see that Student ID is being used as the index column.

```
[6]: students_df
```

```
[6]:
```

	Name	Age	Subject	Year of Study \
Student ID				
2703f3f0	Mr Clifford Watson	25.0	English Literature	1.0
a8040287	Elliott Ward	25.0	Computer Science	4.0
d8da5486	Miss Pauline Dunn	22.0	Engineering	4.0
3ac1b74d	Mr Dominic Mason	22.0	Physics	1.0
67850858	Mrs Melanie Brown	18.0	English Literature	3.0
...
a8be1ec3	Kelly Foster	22.0	Engineering	1.0
3b69ff22	Sara Austin	19.0	Computer Science	34.0
716fb45f	Miss Grace Miller	22.0	English Literature	4.0
34b97db2	Miss Lydia Saunders	23.0	Physics	2.0
34b97db2	Miss Lydia Saunders	23.0	Physics	2.0

Country of Origin

```

Student ID
2703f3f0    Saint Barthelemy
a8040287          Guinea
d8da5486      Afghanistan
3ac1b74d          Palau
67850858      Algeria
...
a8be1ec3      Netherlands
3b69ff22      Liechtenstein
716fb45f      Comoros
34b97db2      Faroe Islands
34b97db2      Faroe Islands

```

[101 rows x 5 columns]

The index has changed accordingly:

```
[7]: students_df.index
```

```

[7]: Index(['2703f3f0', 'a8040287', 'd8da5486', '3ac1b74d', '67850858', '62dd3a69',
          '6b22a999', '4b744b9a', '45c54817', '5d5e1224',
          ...,
          'f5273fa2', 'e8f05741', '655726b1', '27eca82c', 'bf9937ac', 'a8be1ec3',
          '3b69ff22', '716fb45f', '34b97db2', '34b97db2'],
          dtype='object', name='Student ID', length=101)

```

1.2 Explore our dataset

We can easily find out how many records we loaded:

```
[8]: len(students_df)
```

```
[8]: 101
```

We can examine the first few records in our dataset using head:

```
[9]: students_df.head()
```

```

[9]:
      Student ID      Name  Age  Subject  Year of Study \
0  2703f3f0  Mr Clifford Watson  25.0  English Literature      1.0
1  a8040287    Elliott Ward  25.0    Computer Science      4.0
2  d8da5486  Miss Pauline Dunn  22.0      Engineering      4.0
3  3ac1b74d  Mr Dominic Mason  22.0      Physics      1.0
4  67850858  Mrs Melanie Brown  18.0  English Literature      3.0

      Country of Origin
Student ID
2703f3f0    Saint Barthelemy

```

a8040287	Guinea
d8da5486	Afghanistan
3ac1b74d	Palau
67850858	Algeria

By default, this shows the first five rows in the DataFrame, but we can also specify the number we want:

```
[10]: students_df.head(10)
```

```
[10]:
```

	Name	Age	Subject	Year of Study	\
Student ID					
2703f3f0	Mr Clifford Watson	25.0	English Literature	1.0	
a8040287	Elliot Ward	25.0	Computer Science	4.0	
d8da5486	Miss Pauline Dunn	22.0	Engineering	4.0	
3ac1b74d	Mr Dominic Mason	22.0	Physics	1.0	
67850858	Mrs Melanie Brown	18.0	English Literature	3.0	
62dd3a69	Mr Frederick Price	22.0	Medicine	2.0	
6b22a999	Charles Hayward	23.0	Engineering	2.0	
4b744b9a	Garry Thornton	NaN	English Literature	4.0	
45c54817	Mrs Sian Wilson	20.0	Computer Science	2.0	
5d5e1224	Grace Walton-Kelly	20.0	History	1.0	

	Country of Origin
Student ID	
2703f3f0	Saint Barthelemy
a8040287	Guinea
d8da5486	Afghanistan
3ac1b74d	Palau
67850858	Algeria
62dd3a69	Guinea-Bissau
6b22a999	Comoros
4b744b9a	Cuba
45c54817	Korea
5d5e1224	Haiti

Similarly, we can view the last few rows in our dataset using tail:

```
[11]: students_df.tail()
```

```
[11]:
```

	Name	Age	Subject	Year of Study	\
Student ID					
a8be1ec3	Kelly Foster	22.0	Engineering	1.0	
3b69ff22	Sara Austin	19.0	Computer Science	34.0	
716fb45f	Miss Grace Miller	22.0	English Literature	4.0	
34b97db2	Miss Lydia Saunders	23.0	Physics	2.0	
34b97db2	Miss Lydia Saunders	23.0	Physics	2.0	

Student ID	Country of Origin
a8be1ec3	Netherlands
3b69ff22	Liechtenstein
716fb45f	Comoros
34b97db2	Faroe Islands
34b97db2	Faroe Islands

We can also get a random sample of records from the DataFrame using `sample`.

```
[12]: students_df.sample(10)
```

```
[12]:
```

	Name	Age	Subject	Year of Study	\
Student ID					
3eec010c	Howard Marshall	NaN	Biology	1.0	
3b921658	Pamela Mason	22.0	History	2.0	
8956f118	Sara Smith	25.0	Engineering	2.0	
e506b33a	Bethany Davies	21.0	Biology	2.0	
a8040287	Ellriott Ward	25.0	Computer Science	4.0	
5d5e1224	Grace Walton-Kelly	20.0	History	1.0	
3c1fe24c	Melanie Sheppard	21.0	Medicine	1.0	
27166ecb	Dr Terry Lewis	245.0	Mathematics	3.0	
f5273fa2	Leigh Palmer	24.0	English Literature	3.0	
a7bb5b63	Steven Ahmed	22.0	Physics	1.0	

Student ID	Country of Origin
3eec010c	Swaziland
3b921658	Bahrain
8956f118	Israel
e506b33a	Malaysia
a8040287	Guinea
5d5e1224	Haiti
3c1fe24c	Lebanon
27166ecb	Fiji
f5273fa2	Belize
a7bb5b63	Guernsey

We can get a quick overview of the data using `info`. This shows that we have 101 non-null values of `Name`, but only 99 non-null values of `Age` - it looks like we have a couple of missing age values. We can also see that `Age` and `Year of Study` are floating-point values and that `Name`, `Subject` and `Country of Origin` have an 'object' Dtype - this is how strings are represented by default.

```
[13]: students_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 101 entries, 2703f3f0 to 34b97db2
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
  0   ...              101            object
  1   ...              101            object
  2   ...              101            object
  3   ...              101            object
  4   ...              101            object
```

```

---  -----
0   Name                101 non-null    object
1   Age                 99 non-null     float64
2   Subject             101 non-null    object
3   Year of Study       100 non-null     float64
4   Country of Origin   101 non-null    object
dtypes: float64(2), object(3)
memory usage: 4.7+ KB

```

We can use shape to determine how many rows and columns we have. Notice that the number of columns does not include the index column.

```
[14]: students_df.shape # Output: (number of rows, number of columns)
```

```
[14]: (101, 5)
```

We used index above to examine the DataFrame index. Here we use columns to show our column headings. The index column is not included in the list.

```
[15]: students_df.columns
```

```
[15]: Index(['Name', 'Age', 'Subject', 'Year of Study', 'Country of Origin'],
dtype='object')
```

We can use dtypes to specifically check the data types:

```
[16]: students_df.dtypes
```

```
[16]: Name                object
Age                 float64
Subject             object
Year of Study       float64
Country of Origin   object
dtype: object
```

We can get a statistical summary of the numeric data using describe:

```
[17]: students_df.describe()
```

```
[17]:
```

	Age	Year of Study
count	99.000000	100.00000
mean	23.848485	2.76000
std	22.625777	3.35484
min	5.000000	1.00000
25%	20.000000	1.00000
50%	22.000000	2.00000
75%	24.000000	3.25000
max	245.000000	34.00000

If we have a lot of numeric columns, it can be easier view view transposition of the DataFrame returned by describe. We can easily do this using T:

```
[18]: students_df.describe().T
```

```
[18]:
```

	count	mean	std	min	25%	50%	75%	max
Age	99.0	23.848485	22.625777	5.0	20.0	22.0	24.00	245.0
Year of Study	100.0	2.760000	3.354840	1.0	1.0	2.0	3.25	34.0

```
[19]: students_df[10:15]
```

```
[19]:
```

	Name	Age	Subject	Year of Study	\
Student ID					
2c142f24	Jade Kirby	24.0	Biology	1.0	
3eec010c	Howard Marshall	NaN	Biology	1.0	
f3cbb058	Dr Scott Wilson	18.0	English Literature	4.0	
6a406b49	Melissa Ali	18.0	Computer Science	1.0	
836d85df	Carolyn Walker	18.0	Mathematics	4.0	

	Country of Origin
Student ID	
2c142f24	Fiji
3eec010c	Swaziland
f3cbb058	Namibia
6a406b49	Israel
836d85df	Angola

1.3 Clean our data

1.3.1 Handling missing values

We could run into problems if our DataFrame contains rows with missing data, e.g., if we try to use the data to build machine learning models.

We can find out how many missing values for each column as follows:

```
[20]: students_df.isnull().sum()
```

```
[20]:
```

Name	0
Age	2
Subject	0
Year of Study	1
Country of Origin	0
dtype:	int64

There are two main ways in which we might handle missing values. 1. We could just delete the rows that contain the missing values. This is OK when we have enough data to build an effective machine learning model, even when those rows are deleted. 2. We could fill in (or impute) the missing values by trying to guess the value or by using a sensible default value that won't unduly skew our model. This may enable us to benefit from data that otherwise would be deleted and can be a good approach when data is less plentiful.

1.3.2 Imputation: filling in missing values

In the examples below we use `fillna` to fill in the missing 'Age' and 'Years of Study' values with their medians. The median value can be a better choice than the mean value because it is less affected by outliers.

```
[21]: median_age = students_df['Age'].median()
      students_df['Age'] = students_df['Age'].fillna(median_age)
```

```
[22]: median_year_of_study = students_df['Year of Study'].median()
      students_df['Year of Study'] = students_df['Year of Study'].
      ↪fillna(median_year_of_study)
```

Now that we have filled in the missing values, we can confirm that there are no more missing values.

```
[23]: students_df.isnull().sum()
```

```
[23]: Name          0
      Age           0
      Subject       0
      Year of Study  0
      Country of Origin  0
      dtype: int64
```

1.3.3 Deleting rows with missing values

We can use `dropna` rather than `fillna` if we want to delete the rows with missing values:

```
[24]: # We've replaced the missing values already, so these lines will have no effect:
      students_df = students_df.dropna(subset=['Age'])
      students_df = students_df.dropna(subset=['Year of Study'])
```

You might see `inplace=True` being used as in the examples below. This code works but it's no more efficient than the equivalent code above, and it's proposed that the `inplace` parameters be removed in future. See this [stack overflow link](https://stackoverflow.com/questions/45570984/in-pandas-is-inplace-true-considered-harmful-or-not) for more info: <https://stackoverflow.com/questions/45570984/in-pandas-is-inplace-true-considered-harmful-or-not>

```
[25]: students_df.dropna(subset=['Age'], inplace=True)
      students_df.dropna(subset=['Year of Study'], inplace=True)
```

We had 101 records when we checked above. Here we confirm that we haven't deleted any.

```
[26]: len(students_df)
```

```
[26]: 101
```

1.3.4 Checking for outliers

There are various ways in which we could check for outliers. For example, we could visualise the data using a box plot (we'll soon learn more about data visualisation). We could also calculate

the mean and the standard deviation and identify values that were more than, say, 3 standard deviations away from the mean.

For purposes of this example, we'll assume ages of less than 17 or greater than 30 should be confirmed, and that the year of study should be in the range 1 to 4.

In the examples below the vertical bar | is the OR operator.

You can also use a single ampersand & for AND, and a tilde ~ for NOT.

```
[27]: students_df[(students_df['Age'] < 17) | (students_df['Age'] > 30)]
```

```
[27]:
```

	Name	Age	Subject	Year of Study	\
Student ID					
27166ecb	Dr Terry Lewis	245.0	Mathematics	3.0	
95fadd5e	Sheila Berry	5.0	Biology	1.0	

	Country of Origin
Student ID	
27166ecb	Fiji
95fadd5e	Romania

```
[28]: students_df[(students_df['Year of Study'] < 1) | (students_df['Year of Study'] > 4)]
```

```
[28]:
```

	Name	Age	Subject	Year of Study	\
Student ID					
3b69ff22	Sara Austin	19.0	Computer Science	34.0	

	Country of Origin
Student ID	
3b69ff22	Liechtenstein

Ages of 245 and 5 both seem unreasonable, as does a year of study of 34. We can again replace these values or filter out the rows that contain them.

Here we replace the age outliers with the median. We do this using loc and setting a Boolean condition to identify the values we want to replace.

```
[29]: students_df.loc[(students_df['Age'] < 17) | (students_df['Age'] > 30), 'Age'] = median_age
```

And here we replace the year of study outliers with the median.

```
[30]: students_df.loc[(students_df['Year of Study'] < 1) | (students_df['Year of Study'] > 4), 'Year of Study'] = median_year_of_study
```

In the example below, we show how to filter out the outliers:

```
[31]: # We have already replaced the outlier values, so this code won't actually
      ↪ remove any rows
students_df = students_df[(students_df['Age'] >= 17) & (students_df['Age'] <=
      ↪ 30)]
students_df = students_df[(students_df['Year of Study'] >= 1) &
      ↪ (students_df['Year of Study'] <= 4)]
```

```
[32]: # Confirming again that we still have 101 rows
len(students_df)
```

```
[32]: 101
```

1.3.5 Modify the datatypes

When we checked the datatypes earlier, we saw that ‘Age’ and ‘Year of Study’ were floating point values.

```
[33]: # Checking the datatypes again
students_df.dtypes
```

```
[33]: Name                object
Age                  float64
Subject              object
Year of Study        float64
Country of Origin    object
dtype: object
```

It probably makes more sense to represent these with integers rather than floating point values. Here we show how to convert them using `astype`.

```
[34]: students_df['Age'] = students_df['Age'].astype(int)
students_df['Year of Study'] = students_df['Year of Study'].astype(int)
```

```
[35]: # Confirm that 'Age' and 'Year of Study' are now integers
students_df.dtypes
```

```
[35]: Name                object
Age                  int32
Subject              object
Year of Study        int32
Country of Origin    object
dtype: object
```

```
[36]: # The DataFrame also shows 'Age' and 'Year of Study' as integers
students_df.head()
```

```
[36]:
```

	Name	Age	Subject	Year of Study	\
Student ID					

2703f3f0	Mr Clifford Watson	25	English Literature	1
a8040287	Elliott Ward	25	Computer Science	4
d8da5486	Miss Pauline Dunn	22	Engineering	4
3ac1b74d	Mr Dominic Mason	22	Physics	1
67850858	Mrs Melanie Brown	18	English Literature	3

Country of Origin	
Student ID	
2703f3f0	Saint Barthelemy
a8040287	Guinea
d8da5486	Afghanistan
3ac1b74d	Palau
67850858	Algeria

We can also convert the object type column types to strings.

```
[37]: students_df['Name'] = students_df['Name'].astype("string")
students_df['Subject'] = students_df['Subject'].astype("string")
students_df['Country of Origin'] = students_df['Country of Origin'].
      ↪astype("string")

students_df.dtypes
```

```
[37]: Name                string[python]
Age                    int32
Subject               string[python]
Year of Study         int32
Country of Origin     string[python]
dtype: object
```

1.4 More data exploration

Here we use `nunique` to see how many unique values each column has.

```
[38]: students_df.nunique()
```

```
[38]: Name                100
Age                    8
Subject               10
Year of Study         4
Country of Origin     81
dtype: int64
```

We can use `value_counts` to count each unique value for a particular column. Here we see that the ages 22 and 21 occur 16 times each.

```
[39]: students_df['Age'].value_counts()
```

```
[39]: Age
      22    16
      21    16
      24    15
      23    14
      25    12
      18    12
      20     9
      19     7
      Name: count, dtype: int64
```

The `value_counts` method works with categorical data as well. Here we see that English Literature is the most commonly studied subject in our dataset.

```
[40]: students_df['Subject'].value_counts()
```

```
[40]: Subject
      English Literature    16
      Computer Science    13
      Medicine            11
      Biology             11
      Law                 11
      Engineering         9
      Physics              8
      Mathematics         8
      History              7
      Art                  7
      Name: count, dtype: Int64
```

And here we see that Belize is the most frequently occurring country of origin. The values in this dataset were generated randomly, so there is a wide spread of countries of origin.

```
[41]: students_df['Country of Origin'].value_counts()
```

```
[41]: Country of Origin
      Belize              4
      Comoros             3
      Netherlands Antilles 2
      Cyprus              2
      Chad                2
      ..
      Niger              1
      Tuvalu             1
      Kenya            1
      Saint Barthelemy    1
      Lebanon            1
      Name: count, Length: 81, dtype: Int64
```

We can also explore different groupings of data. Here we use `groupby` and `mean` to show the average

student age by subject.

```
[42]: students_df.groupby('Subject')['Age'].mean()
```

```
[42]: Subject
      Art                19.428571
      Biology            20.727273
      Computer Science   21.153846
      Engineering        22.888889
      English Literature  22.625000
      History            22.714286
      Law                21.818182
      Mathematics        20.875000
      Medicine           22.181818
      Physics            22.750000
      Name: Age, dtype: float64
```

Similarly, here we show the average year of study by country of origin.

```
[43]: students_df.groupby('Country of Origin')['Year of Study'].mean()
```

```
[43]: Country of Origin
      Afghanistan        4.0
      Algeria            3.0
      Angola             3.0
      Anguilla           4.0
      Antigua and Barbuda 3.0
      ...
      Uruguay            4.0
      Uzbekistan          1.0
      Vietnam            3.0
      Wallis and Futuna   4.0
      Zimbabwe           1.0
      Name: Year of Study, Length: 81, dtype: float64
```

1.5 Identifying duplicates

We can check for duplicates using a combination of duplicated and sum. The duplicated method returns a series of Boolean True or False values. True if the row is a duplicate of a row above, and False otherwise.

Here we show the series of Boolean values - the final True shows that the last record is a duplicate.

```
[44]: students_df.duplicated()
```

```
[44]: Student ID
      2703f3f0    False
      a8040287    False
```

```

d8da5486    False
3ac1b74d    False
67850858    False
...
a8be1ec3    False
3b69ff22    False
716fb45f    False
34b97db2    False
34b97db2    True
Length: 101, dtype: bool

```

And here we use sum to count the number of duplicate rows (False is equivalent to 0 and True is equivalent to 1).

```
[45]: students_df.duplicated().sum()
```

```
[45]: 1
```

We can see the duplicated rows using loc (for location) and passing in the True/False values returned by students_df.duplicated().

```
[57]: students_df.loc[students_df.duplicated()]
```

```
[57]:
```

	Title	Forename	Surname	Age	Country of Origin	Subject	\
Student ID							
34b97db2	Miss	Lydia	Saunders	23	Faroe Islands	Physics	

	Year of Study
Student ID	
34b97db2	2

We can easily remove duplicate rows from the DataFrame using drop_duplicates.

```
[64]: students_df = students_df.drop_duplicates()
```

We started with 101 rows and have just dropped a duplicate row. So we can expect to have 100 rows remaining.

```
[65]: len(students_df)
```

```
[65]: 100
```

```
[66]: students_df.tail()
```

```
[66]:
```

	Name	Age	Subject	Year of Study	\
Student ID					
bf9937ac	Mr Victor Smith	20	Law	1	
a8be1ec3	Kelly Foster	22	Engineering	1	
3b69ff22	Sara Austin	19	Computer Science	2	
716fb45f	Miss Grace Miller	22	English Literature	4	

34b97db2	Miss Lydia Saunders	23	Physics	2
----------	---------------------	----	---------	---

	Country of Origin
Student ID	
bf9937ac	Bangladesh
a8be1ec3	Netherlands
3b69ff22	Liechtenstein
716fb45f	Comoros
34b97db2	Faroe Islands

1.6 Split the 'Name' column into 'Title', 'Forename', and 'Surname', handling cases where the title is not present

```
[47]: def split_name(name):
      """
      Split name into Title, Forename, and Surname. Return these as a tuple.
      Title just gets an empty string if not included in the name.
      """
      parts = name.split()
      if len(parts) == 3:
          return parts[0], parts[1], parts[2]
      else:
          return '', parts[0], parts[1]
```

Apply the function to each name in the DataFrame

```
[48]: students_df[['Title', 'Forename', 'Surname']] = \
      students_df['Name'].apply(lambda name: pd.Series(split_name(name)))
```

```
[49]: students_df.head()
```

```
[49]:
```

	Name	Age	Subject	Year of Study	\
Student ID					
2703f3f0	Mr Clifford Watson	25	English Literature		1
a8040287	Elliott Ward	25	Computer Science		4
d8da5486	Miss Pauline Dunn	22	Engineering		4
3ac1b74d	Mr Dominic Mason	22	Physics		1
67850858	Mrs Melanie Brown	18	English Literature		3

	Country of Origin	Title	Forename	Surname
Student ID				
2703f3f0	Saint Barthelemy	Mr	Clifford	Watson
a8040287	Guinea		Elliott	Ward
d8da5486	Afghanistan	Miss	Pauline	Dunn
3ac1b74d	Palau	Mr	Dominic	Mason
67850858	Algeria	Mrs	Melanie	Brown

Now that we have Title, Forename and Surname columns, we can use drop to remove the redundant

Name column.

```
[50]: students_df = students_df.drop(columns=['Name'])
```

```
[51]: students_df.head()
```

```
[51]:
```

	Age	Subject	Year of Study	Country of Origin	Title \
Student ID					
2703f3f0	25	English Literature	1	Saint Barthelemy	Mr
a8040287	25	Computer Science	4	Guinea	
d8da5486	22	Engineering	4	Afghanistan	Miss
3ac1b74d	22	Physics	1	Palau	Mr
67850858	18	English Literature	3	Algeria	Mrs

	Forename	Surname
Student ID		
2703f3f0	Clifford	Watson
a8040287	Elliott	Ward
d8da5486	Pauline	Dunn
3ac1b74d	Dominic	Mason
67850858	Melanie	Brown

Let's see what values we extracted for Title.

```
[52]: students_df['Title'].value_counts()
```

```
[52]: Title
```

	67
Mr	13
Miss	8
Dr	7
Mrs	6

Name: count, dtype: int64

We can reorder the columns if we wish.

```
[53]: column_order = ['Title', 'Forename', 'Surname', 'Age', 'Country of Origin',  
    ↪ 'Subject', 'Year of Study']  
students_df = students_df[column_order]
```

```
[54]: students_df.head()
```

```
[54]:
```

	Title	Forename	Surname	Age	Country of Origin	Subject \
Student ID						
2703f3f0	Mr	Clifford	Watson	25	Saint Barthelemy	English Literature
a8040287		Elliott	Ward	25	Guinea	Computer Science
d8da5486	Miss	Pauline	Dunn	22	Afghanistan	Engineering
3ac1b74d	Mr	Dominic	Mason	22	Palau	Physics
67850858	Mrs	Melanie	Brown	18	Algeria	English Literature

Student ID	Year of Study
2703f3f0	1
a8040287	4
d8da5486	4
3ac1b74d	1
67850858	3

1.7 Save the processed dataset

Finally we will write the processed dataset to its own file.

```
[55]: students_df.to_csv('./data/students_processed.csv')
```