In [1]:

```
# Removing Duplicates
# Your datasets will often contain duplicate values, and
# frequently, you will need to remove these duplicate values.

import pandas as pd
scores = [['Mathematics', 85, 'Science'],
['English', 91, 'Arts'],
['History', 95, 'Chemistry'],
['History', 95, 'Chemistry'],
['English', 95, 'Chemistry'],
]
my_df = pd.DataFrame(scores, columns = ['Subject', 'Score', 'Subject'])
my_df.head()
```

Out[1]:

| | Subject | Score | Subject |
|---|---|---|---|
| **0** | Mathematics | 85 | Science |
| **1** | English | 91 | Arts |
| **2** | History | 95 | Chemistry |
| **3** | History | 95 | Chemistry |
| **4** | English | 95 | Chemistry |

In [2]:

```
# From the above output, you can see that there are some
# duplicate rows (index 2,3), as well as duplicate columns
# (Subject) in our dataset.
```

In [3]:

```
# Removing Duplicate Rows
# To remove duplicate rows, you can call the drop_duplicates()
# method, which keeps the first instance and removes all the
# duplicate rows.
# Here is an example that removes the row at index 3, which is a
# duplicate of the row at index 2, from our dummy dataframe.

result = my_df.drop_duplicates()
result.head()
```

Out[3]:

| | Subject | Score | Subject |
|---|---|---|---|
| **0** | Mathematics | 85 | Science |
| **1** | English | 91 | Arts |
| **2** | History | 95 | Chemistry |
| **4** | English | 95 | Chemistry |

In [4]:

```
# If you want to keep the last instance and remove the
# remaining duplicates, you need to pass the string “last” as the
# value for the keep attribute of the drop_duplicates() method.
# Here is an example:

result = my_df.drop_duplicates(keep='last')
result.head()
```

Out[4]:

| | Subject | Score | Subject |
|---|---|---|---|
| **0** | Mathematics | 85 | Science |
| **1** | English | 91 | Arts |
| **3** | History | 95 | Chemistry |
| **4** | English | 95 | Chemistry |

In [5]:

```
# Finally, if you want to remove all the duplicate rows from your
# Pandas dataframe without keeping any instance, you can pass
# the Boolean value False as the value for the keep attribute, as
# shown in the example below.

result = my_df.drop_duplicates(keep=False)
result.head()
```

Out[5]:

| | Subject | Score | Subject |
|---|---|---|---|
| **0** | Mathematics | 85 | Science |
| **1** | English | 91 | Arts |
| **4** | English | 95 | Chemistry |

In [6]:

```
# By default, the drop_duplicates() method only removes
# duplicate rows, where all the columns contain duplicate
# values. If you want to remove rows based on duplicate values
# in a subset of columns, you need to pass the column list to the
# subset attribute.
# For instance, the script below removes all rows where the
# Score column contains duplicate values.

result = my_df.drop_duplicates(subset=['Score'])
result.head()
```

Out[6]:

| | Subject | Score | Subject |
|---|---|---|---|
| **0** | Mathematics | 85 | Science |
| **1** | English | 91 | Arts |
| **2** | History | 95 | Chemistry |

In [7]:

```
# Removing Duplicate Columns
# There are two main ways to remove duplicate columns in
# Pandas. You can remove two columns with the duplicate
# name, or you can remove two columns containing duplicate
# values for all the rows.
# Let’s create a dummy dataset that contains duplicate column
# names and duplicate values for all rows for different columns.

import pandas as pd
scores = [['Mathematics', 85, 'Science', 85],
['English', 91, 'Arts', 91],
['History', 95, 'Chemistry', 95],
['History', 95, 'Chemistry', 95],
['English', 95, 'Chemistry', 95],
]
my_df = pd.DataFrame(scores, columns = ['Subject', 'Score', 'Subject',
'Percentage'])
my_df.head()
```

Out[7]:

| | Subject | Score | Subject | Percentage |
|---|---|---|---|---|
| **0** | Mathematics | 85 | Science | 85 |
| **1** | English | 91 | Arts | 91 |
| **2** | History | 95 | Chemistry | 95 |
| **3** | History | 95 | Chemistry | 95 |
| **4** | English | 95 | Chemistry | 95 |

In [8]:

```
# You can see that the above dataframe contains two columns
# with the name Subject. Also, the Score and Percentage
# columns have duplicate values for all the rows.
```

In [9]:

```
# Let’s first remove the columns with duplicate names. Here is
# how you can do that using the duplicated() method.

result = my_df.loc[:,~my_df.columns.duplicated()]
result.head()
```

Out[9]:

| | Subject | Score | Percentage |
|---|---|---|---|
| **0** | Mathematics | 85 | 85 |
| **1** | English | 91 | 91 |
| **2** | History | 95 | 95 |
| **3** | History | 95 | 95 |
| **4** | English | 95 | 95 |

In [10]:

```
# To remove the columns with the same values, you can convert
# columns to rows using the “T” attribute and then call drop_
# duplicates() on the transposed dataframe. Finally, you can
# again transpose the resultant dataframe, which will have
# duplicate columns removed. Here is a sample script on how
# you can do that.

result = my_df.T.drop_duplicates().T
result.head()
```

Out[10]:

| | Subject | Score | Subject |
|---|---|---|---|
| **0** | Mathematics | 85 | Science |
| **1** | English | 91 | Arts |
| **2** | History | 95 | Chemistry |
| **3** | History | 95 | Chemistry |
| **4** | English | 95 | Chemistry |

In [11]:

```
# Lets consider another dataframe.

scores = pd.DataFrame({'name':['Adam', 'Bob', 'Dave', 'Fred'],
                       'age': [15, 16, 16, 15],
                       'test1': [95, 81, 89, None],
                       'test2': [80, 82, 84, 88],
                       'teacher': ['Ashby', 'Ashby', 'Jones', 'Jones']})
scores
```

Out[11]:

| | name | age | test1 | test2 | teacher |
|---|---|---|---|---|---|
| **0** | Adam | 15 | 95.0 | 80 | Ashby |
| **1** | Bob | 16 | 81.0 | 82 | Ashby |
| **2** | Dave | 16 | 89.0 | 84 | Jones |
| **3** | Fred | 15 | NaN | 88 | Jones |

In [12]:

```
# Using a pivot table, we can generalize certain groupby behaviors. To get
# the median teacher scores we can run the following:

scores.pivot_table(index='teacher',
                   values=['test1', 'test2'],
                   aggfunc='median')
```

Out[12]:

| | test1 | test2 |
|---|---|---|
| **teacher** | | |
| **Ashby** | 88.0 | 81 |
| **Jones** | 89.0 | 86 |

In [13]:

```
# If we want to aggregate by teacher and age, we simply use a list with
# both of them for the index parameter:

scores.pivot_table(index=['teacher', 'age'],
                   values=['test1', 'test2'],
                   aggfunc='median')
```

Out[13]:

| | | test1 | test2 |
|---|---|---|---|
| **teacher** | **age** | | |
| **Ashby** | **15** | 95.0 | 80 |
| | **16** | 81.0 | 82 |
| **Jones** | **15** | NaN | 88 |
| | **16** | 89.0 | 84 |

In [14]:

```
#If we want to apply multiple functions, just use a list of them. Here, we
# look at the minimum and maximum test scores by teacher:

scores.pivot_table(index='teacher',
                   values=['test1', 'test2'],
                   aggfunc=[min, max])
```

Out[14]:

| | min | | max | |
|---|---|---|---|---|
| | test1 | test2 | test1 | test2 |
| **teacher** | | | | |
| **Ashby** | 81.0 | 80 | 95.0 | 82 |
| **Jones** | 89.0 | 84 | 89.0 | 88 |

In [15]:

```
# One additional feature of pivot tables is the ability to add summary
# rows. Simply by setting margins=True we get this functionality:

scores.pivot_table(index='teacher',
                   values=['test1', 'test2'],
                   aggfunc='median', margins=True)
```

Out[15]:

| | test1 | test2 |
|---|---|---|
| **teacher** | | |
| **Ashby** | 88.0 | 81 |
| **Jones** | 89.0 | 86 |
| **All** | 89.0 | 82 |