# Introduction

The first part of this series (http://pbpython.com/excel-pandas-comp.html) was very well received so I thought I would continue the theme of showing how to do common Excel tasks in pandas.

In the first article, I focused on common, math tasks in Excel and how to do them in pandas. In this article, I'll focus on some other Excel tasks related to data selection and how to map them to pandas.

Please refer to this post (http://pbpython.com/excel-pandas-comp-2.html) for the full post.

# Getting Set Up

Import the pandas and numpy modules.

In [1]:

```
import pandas as pd
import numpy as np
```

Load in the Excel data that represents a year's worth of sales.

In [2]:

```
df = pd.read_excel("../in/sample-salesv3.xlsx")
```

Take a quick look at the data types to make sure everything came through as expected.

In [3]:

```
df.dtypes
```

Out[3]:

```
account number      int64
name               object
sku                object
quantity            int64
unit price        float64
ext price         float64
date               object
dtype: object
```

You'll notice that our date column is showing up as a generic `object`. We are going to convert it to datetime object to make some selections a little easier.

In [4]:

```
df['date'] = pd.to_datetime(df['date'])
```

In [5]:

```
df.head()
```

Out[5]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **0** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 | 2014-01-01 07:21:51 |
| **1** | 714466 | Trantow-Barrows | S2-77896 | -1 | 63.16 | -63.16 | 2014-01-01 10:00:47 |
| **2** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 | 2014-01-01 13:24:58 |
| **3** | 307599 | Kassulke, Ondricka and Metz | S1-65481 | 41 | 21.05 | 863.05 | 2014-01-01 15:05:22 |
| **4** | 412290 | Jerde-Hilpert | S2-34077 | 6 | 83.21 | 499.26 | 2014-01-01 23:26:55 |

In [6]:

```
df.dtypes
```

Out[6]:

```
account number             int64
name                      object
sku                       object
quantity                   int64
unit price               float64
ext price                float64
date              datetime64[ns]
dtype: object
```

The date is now a datetime object which will be useful in future steps.

# Filtering the data

Similar to the autofilter function in Excel, you can use pandas to filter and select certain subsets of data.

For instance, if we want to just see a specific account number, we can easily do that with pandas.

Note, I am going to use the `head` function to show the top results. This is purely for the purposes of keeping the article shorter.

In [8]:

```
df[df["account number"]==307599].head()
```

Out[8]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **3** | 307599 | Kassulke, Ondricka and Metz | S1-65481 | 41 | 21.05 | 863.05 | 2014-01-01 15:05:22 |
| **13** | 307599 | Kassulke, Ondricka and Metz | S2-10342 | 17 | 12.44 | 211.48 | 2014-01-04 07:53:01 |
| **34** | 307599 | Kassulke, Ondricka and Metz | S2-78676 | 35 | 33.04 | 1156.40 | 2014-01-10 05:26:31 |
| **58** | 307599 | Kassulke, Ondricka and Metz | B1-20000 | 22 | 37.87 | 833.14 | 2014-01-15 16:22:22 |
| **70** | 307599 | Kassulke, Ondricka and Metz | S2-10342 | 44 | 96.79 | 4258.76 | 2014-01-18 06:32:31 |

You could also do the filtering based on numeric values.

In [9]:

```
df[df["quantity"] > 22].head()
```

Out[9]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **0** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 | 2014-01-01 07:21:51 |
| **2** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 | 2014-01-01 13:24:58 |
| **3** | 307599 | Kassulke, Ondricka and Metz | S1-65481 | 41 | 21.05 | 863.05 | 2014-01-01 15:05:22 |
| **14** | 737550 | Fritsch, Russel and Anderson | B1-53102 | 23 | 71.56 | 1645.88 | 2014-01-04 08:57:48 |
| **15** | 239344 | Stokes LLC | S1-06532 | 34 | 71.51 | 2431.34 | 2014-01-04 11:34:58 |

If we want to do more complex filtering, we can use `map` to filter. In this example, let's look for items with sku's that start with B1.

In [10]:

```
df[df["sku"].map(lambda x: x.startswith('B1'))].head()
```

Out[10]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **0** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 | 2014-01-01 07:21:51 |
| **2** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 | 2014-01-01 13:24:58 |
| **6** | 218895 | Kulas Inc | B1-65551 | 2 | 31.10 | 62.20 | 2014-01-02 10:57:23 |
| **14** | 737550 | Fritsch, Russel and Anderson | B1-53102 | 23 | 71.56 | 1645.88 | 2014-01-04 08:57:48 |
| **17** | 239344 | Stokes LLC | B1-50809 | 14 | 16.23 | 227.22 | 2014-01-04 22:14:32 |

It's easy to chain two statements together using the &.

In [11]:

```
df[df["sku"].map(lambda x: x.startswith('B1')) & (df["quantity"] > 22)].head()
```

Out[11]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **0** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 | 2014-01-01 07:21:51 |
| **2** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 | 2014-01-01 13:24:58 |
| **14** | 737550 | Fritsch, Russel and Anderson | B1-53102 | 23 | 71.56 | 1645.88 | 2014-01-04 08:57:48 |
| **26** | 737550 | Fritsch, Russel and Anderson | B1-53636 | 42 | 42.06 | 1766.52 | 2014-01-08 00:02:11 |
| **31** | 714466 | Trantow-Barrows | B1-33087 | 32 | 19.56 | 625.92 | 2014-01-09 10:16:32 |

Another useful function that pandas supports is called `isin`. It allows us to define a list of values we want to look for.

In this case, we look for all records that include two specific account numbers.

In [12]:

```
df[df["account number"].isin([714466,218895])].head()
```

Out[12]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **1** | 714466 | Trantow-Barrows | S2-77896 | -1 | 63.16 | -63.16 | 2014-01-01 10:00:47 |
| **2** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 | 2014-01-01 13:24:58 |
| **5** | 714466 | Trantow-Barrows | S2-77896 | 17 | 87.63 | 1489.71 | 2014-01-02 10:07:15 |
| **6** | 218895 | Kulas Inc | B1-65551 | 2 | 31.10 | 62.20 | 2014-01-02 10:57:23 |
| **8** | 714466 | Trantow-Barrows | S1-50961 | 22 | 84.09 | 1849.98 | 2014-01-03 11:29:02 |

Pandas supports another function called `query` which allows you to efficiently select subsets of data. It does require the installation of numexpr (https://github.com/pydata/numexpr) so make sure you have it installed before trying this step.

If you would like to get a list of customers by name, you can do that with a query, similar to the python syntax shown above.

In [13]:

```
df.query('name == ["Kulas Inc","Barton LLC"]').head()
```

Out[13]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **0** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 | 2014-01-01 07:21:51 |
| **2** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 | 2014-01-01 13:24:58 |
| **6** | 218895 | Kulas Inc | B1-65551 | 2 | 31.10 | 62.20 | 2014-01-02 10:57:23 |
| **33** | 218895 | Kulas Inc | S1-06532 | 3 | 22.36 | 67.08 | 2014-01-09 23:58:27 |
| **36** | 218895 | Kulas Inc | S2-34077 | 16 | 73.04 | 1168.64 | 2014-01-10 12:07:30 |

The query function allows you do more than just this simple example but for the purposes of this discussion, I'm showing it so you are aware that it is out there for you.

# Working with Dates

Using pandas, you can do complex filtering on dates. Before doing anything with dates, I encourage you to sort by the date column to make sure the results return what you are expecting.

In [14]:

```
df = df.sort('date')
df.head()
```

Out[14]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **0** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 | 2014-01-01 07:21:51 |
| **1** | 714466 | Trantow-Barrows | S2-77896 | -1 | 63.16 | -63.16 | 2014-01-01 10:00:47 |
| **2** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 | 2014-01-01 13:24:58 |
| **3** | 307599 | Kassulke, Ondricka and Metz | S1-65481 | 41 | 21.05 | 863.05 | 2014-01-01 15:05:22 |
| **4** | 412290 | Jerde-Hilpert | S2-34077 | 6 | 83.21 | 499.26 | 2014-01-01 23:26:55 |

The python filtering syntax shown before works with dates.

In [15]:

```
df[df['date'] >='20140905'].head()
```

Out[15]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **1042** | 163416 | Purdy-Kunde | B1-38851 | 41 | 98.69 | 4046.29 | 2014-09-05 01:52:32 |
| **1043** | 714466 | Trantow-Barrows | S1-30248 | 1 | 37.16 | 37.16 | 2014-09-05 06:17:19 |
| **1044** | 729833 | Koepp Ltd | S1-65481 | 48 | 16.04 | 769.92 | 2014-09-05 08:54:41 |
| **1045** | 729833 | Koepp Ltd | S2-11481 | 6 | 26.50 | 159.00 | 2014-09-05 16:33:15 |
| **1046** | 737550 | Fritsch, Russel and Anderson | B1-33364 | 4 | 76.44 | 305.76 | 2014-09-06 08:59:08 |

One of the really nice features of pandas is that it understands dates so will allow us to do partial filtering. If we want to only look for data more recent than a specific month, we can do so.

In [16]:

```
df[df['date'] >='2014-03'].head()
```

Out[16]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **242** | 163416 | Purdy-Kunde | S1-30248 | 19 | 65.03 | 1235.57 | 2014-03-01 16:07:40 |
| **243** | 527099 | Sanford and Sons | S2-82423 | 3 | 76.21 | 228.63 | 2014-03-01 17:18:01 |
| **244** | 527099 | Sanford and Sons | B1-50809 | 8 | 70.78 | 566.24 | 2014-03-01 18:53:09 |
| **245** | 737550 | Fritsch, Russel and Anderson | B1-50809 | 20 | 50.11 | 1002.20 | 2014-03-01 23:47:17 |
| **246** | 688981 | Keeling LLC | B1-86481 | -1 | 97.16 | -97.16 | 2014-03-02 01:46:44 |

Of course, you can chain the criteria.

In [17]:

```
df[(df['date'] >='20140701') & (df['date'] <= '20140715')].head()
```

Out[17]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **778** | 737550 | Fritsch, Russel and Anderson | S1-65481 | 35 | 70.51 | 2467.85 | 2014-07-01 00:21:58 |
| **779** | 218895 | Kulas Inc | S1-30248 | 9 | 16.56 | 149.04 | 2014-07-01 00:52:38 |
| **780** | 163416 | Purdy-Kunde | S2-82423 | 44 | 68.27 | 3003.88 | 2014-07-01 08:15:52 |
| **781** | 672390 | Kuhn-Gusikowski | B1-04202 | 48 | 99.39 | 4770.72 | 2014-07-01 11:12:13 |
| **782** | 642753 | Pollich LLC | S2-23246 | 1 | 51.29 | 51.29 | 2014-07-02 04:02:39 |

Because pandas understands date columns, you can express the date value in multiple formats and it will give you the results you expect.

In [26]:

```
df[df['date'] >= 'Oct-2014'].head()
```

Out[26]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **1168** | 307599 | Kassulke, Ondricka and Metz | S2-23246 | 6 | 88.90 | 533.40 | 2014-10-08 06:19:50 |
| **1169** | 424914 | White-Trantow | S2-10342 | 25 | 58.54 | 1463.50 | 2014-10-08 07:31:40 |
| **1170** | 163416 | Purdy-Kunde | S1-27722 | 22 | 34.41 | 757.02 | 2014-10-08 09:01:18 |
| **1171** | 163416 | Purdy-Kunde | B1-33087 | 7 | 79.29 | 555.03 | 2014-10-08 15:39:13 |
| **1172** | 672390 | Kuhn-Gusikowski | B1-38851 | 30 | 94.64 | 2839.20 | 2014-10-09 00:22:33 |

In [27]:

```
df[df['date'] >= '10-10-2014'].head()
```

Out[27]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **1174** | 257198 | Cronin, Oberbrunner and Spencer | S2-34077 | 13 | 12.24 | 159.12 | 2014-10-10 02:59:06 |
| **1175** | 740150 | Barton LLC | S1-65481 | 28 | 53.00 | 1484.00 | 2014-10-10 15:08:53 |
| **1176** | 146832 | Kiehn-Spinka | S1-27722 | 15 | 64.39 | 965.85 | 2014-10-10 18:24:01 |
| **1177** | 257198 | Cronin, Oberbrunner and Spencer | S2-16558 | 3 | 35.34 | 106.02 | 2014-10-11 01:48:13 |
| **1178** | 737550 | Fritsch, Russel and Anderson | B1-53636 | 10 | 56.95 | 569.50 | 2014-10-11 10:25:53 |

When working with time series data, if we convert the data to use the date as at the index, we can do some more filtering.

Set the new index using `set_index`.

In [28]:

```
df2 = df.set_index(['date'])
df2.head()
```

Out[28]:

| date | account number | name | sku | quantity | unit price | ext price |
|---|---|---|---|---|---|---|
| **2014-01-01 07:21:51** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 |
| **2014-01-01 10:00:47** | 714466 | Trantow-Barrows | S2-77896 | -1 | 63.16 | -63.16 |
| **2014-01-01 13:24:58** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 |
| **2014-01-01 15:05:22** | 307599 | Kassulke, Ondricka and Metz | S1-65481 | 41 | 21.05 | 863.05 |
| **2014-01-01 23:26:55** | 412290 | Jerde-Hilpert | S2-34077 | 6 | 83.21 | 499.26 |

We can slice the data to get a range.

In [29]:

```
df2["20140101":"20140201"].head()
```

Out[29]:

| date | account number | name | sku | quantity | unit price | ext price |
|---|---|---|---|---|---|---|
| **2014-01-01 07:21:51** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 |
| **2014-01-01 10:00:47** | 714466 | Trantow-Barrows | S2-77896 | -1 | 63.16 | -63.16 |
| **2014-01-01 13:24:58** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 |
| **2014-01-01 15:05:22** | 307599 | Kassulke, Ondricka and Metz | S1-65481 | 41 | 21.05 | 863.05 |
| **2014-01-01 23:26:55** | 412290 | Jerde-Hilpert | S2-34077 | 6 | 83.21 | 499.26 |

Once again, we can use various date representations to remove any ambiguity around date naming conventions.

In [38]:

```
df2["2014-Jan-1":"2014-Feb-1"].head()
```

Out[38]:

| date | account number | name | sku | quantity | unit price | ext price |
|---|---|---|---|---|---|---|
| **2014-01-01 07:21:51** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 |
| **2014-01-01 10:00:47** | 714466 | Trantow-Barrows | S2-77896 | -1 | 63.16 | -63.16 |
| **2014-01-01 13:24:58** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 |
| **2014-01-01 15:05:22** | 307599 | Kassulke, Ondricka and Metz | S1-65481 | 41 | 21.05 | 863.05 |
| **2014-01-01 23:26:55** | 412290 | Jerde-Hilpert | S2-34077 | 6 | 83.21 | 499.26 |

In [39]:

```
df2["2014-Jan-1":"2014-Feb-1"].tail()
```

Out[39]:

| date | account number | name | sku | quantity | unit price | ext price |
|---|---|---|---|---|---|---|
| **2014-01-31 22:51:18** | 383080 | Will LLC | B1-05914 | 43 | 80.17 | 3447.31 |
| **2014-02-01 09:04:59** | 383080 | Will LLC | B1-20000 | 7 | 33.69 | 235.83 |
| **2014-02-01 11:51:46** | 412290 | Jerde-Hilpert | S1-27722 | 11 | 21.12 | 232.32 |
| **2014-02-01 17:24:32** | 412290 | Jerde-Hilpert | B1-86481 | 3 | 35.99 | 107.97 |
| **2014-02-01 19:56:48** | 412290 | Jerde-Hilpert | B1-20000 | 23 | 78.90 | 1814.70 |

In [40]:

```
df2["2014"].head()
```

Out[40]:

| date | account number | name | sku | quantity | unit price | ext price |
|---|---|---|---|---|---|---|
| **2014-01-01 07:21:51** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 |
| **2014-01-01 10:00:47** | 714466 | Trantow-Barrows | S2-77896 | -1 | 63.16 | -63.16 |
| **2014-01-01 13:24:58** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 |
| **2014-01-01 15:05:22** | 307599 | Kassulke, Ondricka and Metz | S1-65481 | 41 | 21.05 | 863.05 |
| **2014-01-01 23:26:55** | 412290 | Jerde-Hilpert | S2-34077 | 6 | 83.21 | 499.26 |

In [42]:

```
df2["2014-Dec"].head()
```

Out[42]:

| date | account number | name | sku | quantity | unit price | ext price |
|---|---|---|---|---|---|---|
| **2014-12-01 20:15:34** | 714466 | Trantow-Barrows | S1-82801 | 3 | 77.97 | 233.91 |
| **2014-12-02 20:00:04** | 146832 | Kiehn-Spinka | S2-23246 | 37 | 57.81 | 2138.97 |
| **2014-12-03 04:43:53** | 218895 | Kulas Inc | S2-77896 | 30 | 77.44 | 2323.20 |
| **2014-12-03 06:05:43** | 141962 | Herman LLC | B1-53102 | 20 | 26.12 | 522.40 |
| **2014-12-03 14:17:34** | 642753 | Pollich LLC | B1-53636 | 19 | 71.21 | 1352.99 |

# Additional String Functions

Pandas has support for vectorized string functions as well. If we want to identify all the skus that contain

a certain value, we can use `str.contains`. In this case, we know that the sku is always represented in the same way, so B1 only shows up in the front of the sku.

In [43]:

```
df[df['sku'].str.contains('B1')].head()
```

Out[43]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **0** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 | 2014-01-01 07:21:51 |
| **2** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 | 2014-01-01 13:24:58 |
| **6** | 218895 | Kulas Inc | B1-65551 | 2 | 31.10 | 62.20 | 2014-01-02 10:57:23 |
| **14** | 737550 | Fritsch, Russel and Anderson | B1-53102 | 23 | 71.56 | 1645.88 | 2014-01-04 08:57:48 |
| **17** | 239344 | Stokes LLC | B1-50809 | 14 | 16.23 | 227.22 | 2014-01-04 22:14:32 |

We can string queries together and use sort to control how the data is ordered.

A common need I have in Excel is to understand all the unique items in a column. For instance, maybe I only want to know when customers purchased in this time period. The unique function makes this trivial.

In [44]:

```
df[(df['sku'].str.contains('B1-531')) & (df['quantity']>40)].sort(columns=['quantity','name'
```

Out[44]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **684** | 642753 | Pollich LLC | B1-53102 | 46 | 26.07 | 1199.22 | 2014-06-08 19:33:33 |
| **792** | 688981 | Keeling LLC | B1-53102 | 45 | 41.19 | 1853.55 | 2014-07-04 21:42:22 |
| **176** | 383080 | Will LLC | B1-53102 | 45 | 89.22 | 4014.90 | 2014-02-11 04:14:09 |
| **1213** | 604255 | Halvorson, Crona and Champlin | B1-53102 | 41 | 55.05 | 2257.05 | 2014-10-18 19:27:01 |
| **1215** | 307599 | Kassulke, Ondricka and Metz | B1-53102 | 41 | 93.70 | 3841.70 | 2014-10-18 23:25:10 |
| **1128** | 714466 | Trantow-Barrows | B1-53102 | 41 | 55.68 | 2282.88 | 2014-09-27 10:42:48 |
| **1001** | 424914 | White-Trantow | B1-53102 | 41 | 81.25 | 3331.25 | 2014-08-26 11:44:30 |

# Bonus Task

I frequently find myself trying to get a list of unique items in a long list within Excel. It is a multi-step process to do this in Excel but is fairly simple in pandas. We just use the `unique` function on a column to get the list.

In [45]:

```
df["name"].unique()
```

Out[45]:

```
array([u'Barton LLC', u'Trantow-Barrows', u'Kulas Inc',
       u'Kassulke, Ondricka and Metz', u'Jerde-Hilpert', u'Koepp Ltd',
       u'Fritsch, Russel and Anderson', u'Kiehn-Spinka', u'Keeling LLC',
       u'Frami, Hills and Schmidt', u'Stokes LLC', u'Kuhn-Gusikowski',
       u'Herman LLC', u'White-Trantow', u'Sanford and Sons',
       u'Pollich LLC', u'Will LLC', u'Cronin, Oberbrunner and Spencer',
       u'Halvorson, Crona and Champlin', u'Purdy-Kunde'], dtype=object)
```

If we wanted to include the account number, we could use `drop_duplicates`.

In [48]:

```
df.drop_duplicates(subset=["account number","name"]).head()
```

Out[48]:

| | account number | name | sku | quantity | unit price | ext price | date |
|---|---|---|---|---|---|---|---|
| **0** | 740150 | Barton LLC | B1-20000 | 39 | 86.69 | 3380.91 | 2014-01-01 07:21:51 |
| **1** | 714466 | Trantow-Barrows | S2-77896 | -1 | 63.16 | -63.16 | 2014-01-01 10:00:47 |
| **2** | 218895 | Kulas Inc | B1-69924 | 23 | 90.70 | 2086.10 | 2014-01-01 13:24:58 |
| **3** | 307599 | Kassulke, Ondricka and Metz | S1-65481 | 41 | 21.05 | 863.05 | 2014-01-01 15:05:22 |
| **4** | 412290 | Jerde-Hilpert | S2-34077 | 6 | 83.21 | 499.26 | 2014-01-01 23:26:55 |

We are obviously pulling in more data than we need and getting some non-useful information, so select only the first and second columns using `ix`.

In [49]:

```
df.drop_duplicates(subset=["account number","name"]).ix[:,[0,1]]
```

Out[49]:

| | account number | name |
|---|---|---|
| **0** | 740150 | Barton LLC |
| **1** | 714466 | Trantow-Barrows |
| **2** | 218895 | Kulas Inc |
| **3** | 307599 | Kassulke, Ondricka and Metz |
| **4** | 412290 | Jerde-Hilpert |
| **7** | 729833 | Koepp Ltd |
| **9** | 737550 | Fritsch, Russel and Anderson |
| **10** | 146832 | Kiehn-Spinka |
| **11** | 688981 | Keeling LLC |
| **12** | 786968 | Frami, Hills and Schmidt |
| **15** | 239344 | Stokes LLC |
| **16** | 672390 | Kuhn-Gusikowski |
| **18** | 141962 | Herman LLC |

| | account number | name |
|---|---|---|
| **20** | 424914 | White-Trantow |
| **21** | 527099 | Sanford and Sons |
| **30** | 642753 | Pollich LLC |
| **37** | 383080 | Will LLC |
| **51** | 257198 | Cronin, Oberbrunner and Spencer |
| **67** | 604255 | Halvorson, Crona and Champlin |
| **106** | 163416 | Purdy-Kunde |

I hope you found this useful. I encourage you to try and apply these ideas to some of your own repetitive Excel tasks and streamline your work flow.