Refiner 5

Overview

The Instabase Refiner App helps you “refine” (or “extract”) specific data from similar documents.

Instabase helps you create formulas to extract fields in a set of similar documents.

By the end of this guide, you should be able to:

- Use `scan` functions, with optional parameters
- Implement positional functions
- Identify different types of functional inputs
- Examine and update the output of a Refiner Flow

Prerequisites

This guide assumes that you've completed the guide for Instabase Flow. It assumes understanding of previous keywords, file extensions, and core Instabase concepts.

This guide will use the same paystub data as in the Flow guide, which can be found below:

- ADP paystubs

1. Setting up your workspace

First, set up a new Refiner workspace. If you made mistakes in the previous guide, this allows you to start from a clean slate.

Activity

1. Create a new workspace called `refiner-function-practice`
2. Install the "Refiner V5 Project" (template).
3. Upload the ADP paystubs into an `input` folder.
4. From `post_install_script.ibflow` , **Tools** > **Run**, using `input` as your data set.
5. From `viewme.ibrecipebook` , click **Edit** in "Edit Refiner Program". This is where we'll spend most of our time.

Instructions too quick? Check out the Flow guide for the longer version of the above.

2. A brief UI tour

The Refiner view has the list of documents on the left-hand panel, an image display in the center, and the list of extracted fields on the right.

Clicking different document names on the left-hand panel will bring up their image in the center of the app. The text results from OCR can be seen by toggling the capital A icon above the image display. In the text view, we can highlight and copy text, just like we would with a static text document. We cannot alter the text.

When we add a field in the right panel, we can give it a descriptive name like `employee_name` or `period_beginning` and a type of extraction we want to do. Then, we can use the bottom pane of the program to edit and test each field. In this bottom pane, we can also add a description of the field, if the variable name is not enough. For example, for `period_beginning` , we could add `"The start date of this paystub's pay period"` .

A paystub sheet overview

For this activity, the Refiner app is populated with the ADP paystubs that we added previously.

In this guide, we will add some fields to extract in the right-hand panel.

Activity

1. Open the first document, **fake_adp_1.png**. Change to text view and look at the resulting text. Notice how the structure of the document is preserved with whitespace.

2. Add a field. Give it the label `greeting`, and the expression type `echo`. `echo` is a standard function that simply means "write", or "post". Select the newly created field, and when it opens in the bottom panel, type `echo('hello')`. `'hello'` (in single quotes - double will error), is just a value (argument) that we're asking `echo` to post.
3. Add another field. Give it the label `greeting2` and the expression type `echo`. Give this one the formula `echo(INPUT_COL)`. This time, `INPUT_COL` is our argument. `INPUT_COL` is a variable, which represents the data found in each document.
4. Add another field. Give it the label `greeting3` and the expression type `echo`. Type `echo(Input_Col)`. This will error out, and remind you that case-sensitivity is a must!
5. Add another field. Give it the label `greeting4`. Type `echo(greeting)`. In Refiner, you may reference and use fields that are above the field you are defining. What do you think it'll post? This reinforces how valuable it is to actually update your column names to reflect their content.
6. At the bottom of the app, click the **Run** button. Switch to the Output Table view. Do the outputs match your expectations?

3. Scan functions

Refiner has a collection of functions called Scan Functions.

This family of functions are all ways of performing the same basic actions:

- Finding a coordinate on the page, and...
- Returning some other region of the page relative to that coordinate

For example:

- `scan_right` finds text and returns everything after it
- `scan_below` finds text and returns everything below it

Scan function gotchas

Activity

1. Add a new field labeled `period_beginning`, using the `scan_right` expression. The formula you can use is `scan_right(INPUT_COL, 'Period Beginning:')`. As you can see, this function takes two arguments. First, the `INPUT_COL`, which is our

paystub data. This input is a given for most scan functions. Second, to narrow the result, we're looking for everything to the right of “Period Beginning”.
2. Run and look at the output. There are so many spaces! We can clean that output by selecting the field, **period-beginning**, then **Output** below, then **Clean excess spaces** .
3. Click **Run** again.

Check out the Output Table to see the results on all of the documents.

Some of your fields might not have returned anything at all! We'll resolve that in the next activity.

Activity

1. Edit your `period_beginning` field to say `scan_right(INPUT_COL, 'Period Beginning:', e=1)` .
2. Re-run the program and observe that the missing value has appeared.

Many scan functions support an argument, `e`, which specifies an error tolerance. Adding the argument `e=1` specifies that we can tolerate up to one error. In this example, we're looking for “Period Beginning:", but if the field returns “Period Beginning” without the colon, then that one-off error is permitted.

You might not have realized it, but our text search is invariant of whitespace. The search query `Pay Date` will match the string `Pay Date` . This is a useful feature to point out given the difficulty of normalizing spacing calculations across scanned documents of different resolutions and font sizes.

Next to the formula editor, we can see the documentation of the function we're writing. That should remind us what parameters, like `e`, we can add to our scan functions! More in-depth documentation can be found by going to **Help** –> **Documentation** in the header bar of the app.

Scan below

`scan_below` is similar to `scan_right` , but with a few variations.

Activity

1. Create a new field called `pay_date` and make it a `scan_below` expression. The formula you can use is `scan_below(INPUT_COL, 'Pay Date', num_lines=1)` .

2. Run the program. The output looks incredibly clipped in your results. For this example, only a single character is returned for each output.
3. Change that `scan_below` call to the following: `scan_below(INPUT_COL, 'Pay Date', num_lines=2, right_pad=30)`. This will increase our search output to include multiple lines, and widen the search.
4. Add the `Clean excess spaces` option and run again to filter out the white space.

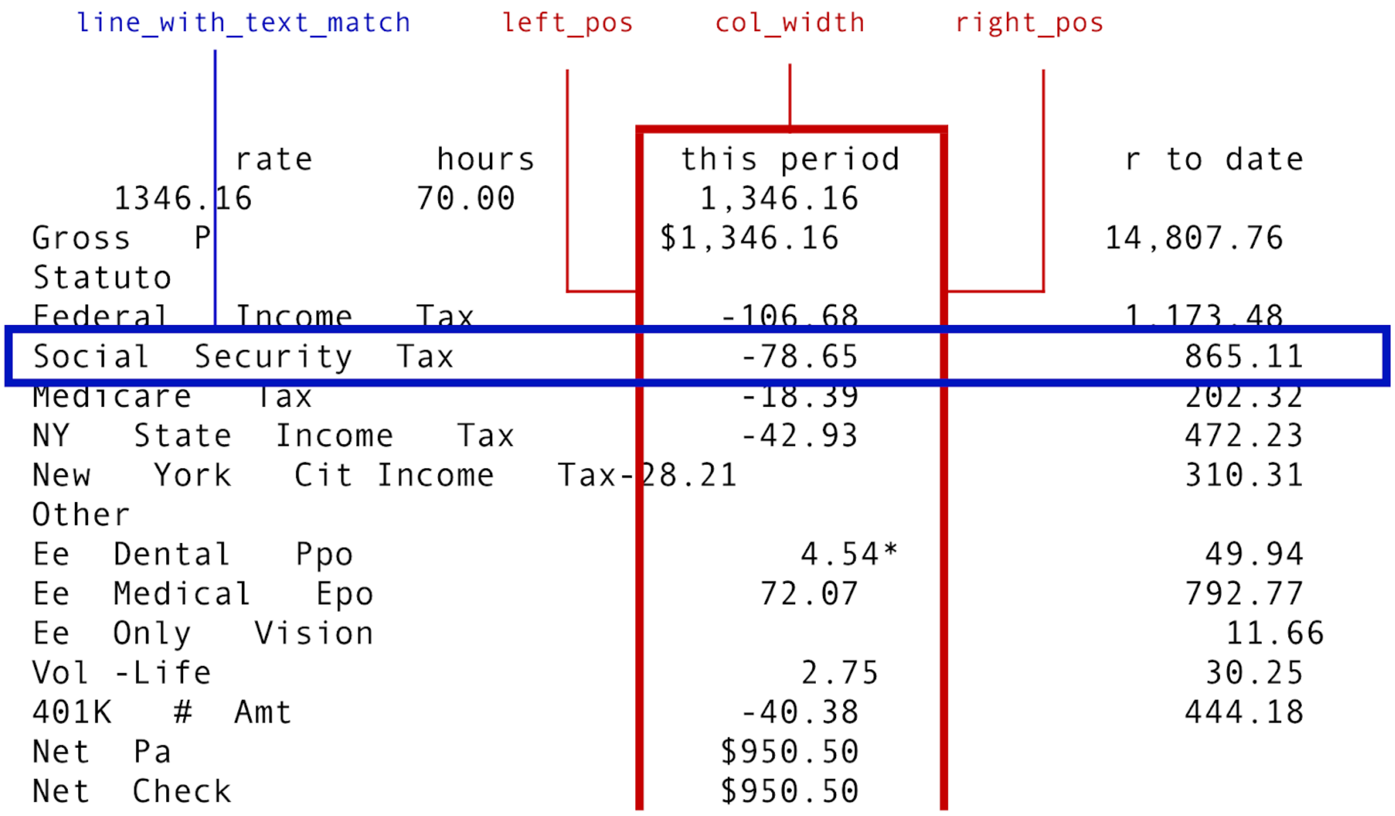
To expand our call, we added an additional `num_line` (2), and increased `padding` to 30. This returns more text, but feels clumsy. To be even more precise, you must first expand your mental model in the next section.

4. Refiner - a mental model

Pretend you are in primary school. Your document is a piece of paper, and Refiner is a pair of craft scissors. Each field you want to extract involves making cuts to winnow the paper down to just the part you want. Sometimes, multiple cuts are easiest.

That basic mental model of paper and scissors will serve you well when visualizing your work in Refiner. Let's construct an example to further cement this mindset.

Say you wanted to extract a person's Social Security Tax amount field from their paystub. At first, your instinct might be to apply some complicated regular expression (which would also work!). However, there is a more robust approach. We can use an approach that leverages the spatial results from the OCR by thinking of the document like this:



```
         rate      hours       this period           r to date
   1346.16        70.00         1,346.16
Gross    P                    $1,346.16            14,807.76
Statuto
Federal   Income    Tax          -106.68             1,173.48
Social  Security  Tax             -78.65               865.11
Medicare    Tax                   -18.39               202.32
NY    State   Income    Tax       -42.93               472.23
New    York    Cit Income    Tax-28.21                 310.31
Other
Ee   Dental    Ppo                  4.54*               49.94
Ee   Medical    Epo               72.07               792.77
Ee   Only    Vision                                    11.66
Vol -Life                           2.75                30.25
401K    #   Amt                   -40.38               444.18
Net   Pa                         $950.50
Net   Check                      $950.50
```

On the one hand, we want to retrieve a row of text based on some search string (Social Security Tax). On the other hand, we want to intersect this row of text against some column (this period) boundaries computed using other document information: the column header.

In other words, we want to turn the lines in the image above into real figures.

Cutting up our paystubs

To reproduce something similar to the above, let's cut our documents up into a few pieces.

Activity

1. Create a new field called `col_pad` that evaluates `echo(2)` . This gives us the value `2` which we can re-use for padding in future functions.
2. Create a new field called `col_left` that evaluates `left_pos(INPUT_COL, 'this period') - col_pad` . This allows us to establish the column width on the left side, with padding accounted for.
3. Create a new field called `col_right` that evaluates `right_pos(INPUT_COL, 'this period') + col_pad` . This establishes column width

on the right side, again accounting for padding.

4. Create a new field called `social_tax` that evaluates `scan(INPUT_COL, 'Social Security', left_pos=col_left, right_pos=col_right, nu` . Finally, this function puts into practice everything we've learned so far. Using `scan` as our main function, we provide all of the different scissor cuts needed to break our paystub into the piece we're looking for, social security tax.
5. Once complete, click the **social_tax field**, then the **dropdown arrow** next to the Run button, then **Run clean**. This will run all functions in order and return your functions.
6. Once complete, click the **social_tax field**, then the **dropdown arrow** next to the Run button, then **Run clean**. This will run all functions in order and return your functions.

The first three columns above draw the vertical lines in the document. The final line cuts out the row, intersecting it with the document.

5. Refiner outputs

Once you're content with the formulas you've created and their outputs, the last step is to share your distilled data.

From formulas to CSV

1. From your Refiner file, click the **Save** button.
2. Click the **Instabase** modal, and then your `refiner_function_practice` directory. Return to your `post_install_script.ibflow` file. Click **Tools** > **Run**.
3. Set your “Input folder” as `input`, then **Run**.
4. Your completed extraction can be found in `out/s4_merge_files/out.ibocr` . Here, you can view your extracted data within Instabase.
5. If you'd like to share your extraction as a csv, click **Export All**. A new file will be produced in `out/s4_merge_files/out.ibocr.csv` .

Conclusion

That's it! You should now have a better sense of how to refine the data you've extracted.

You *should* feel able to complete the following tasks:

- Use a `scan` function, with optional parameters
- Implement positional functions
- Identify different types of functional inputs
- Examine and update the output of a Refiner Flow

If not, reach out to us at training@instabase.com. We'd love to chat about any questions, comments, or concerns that you might've had in completing this guide.