CELFOCUS

Brownbag

Python and Pandas - More than cuddly bears and deadly snakes

Python and Pandas for data wrangling

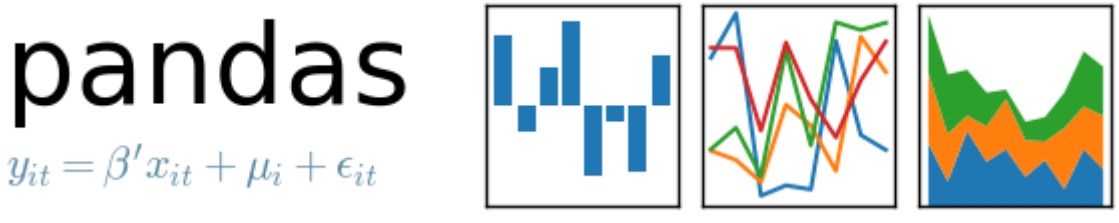
It all starts with Python...

Maybe not this kind of Python...

Maybe something more cuddly like pandas!

Sorry but also not this kind of Pandas...

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.



pandas (Python Data Analysis Library) is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Data Wrangling with Pandas, NumPy, and IPython

Python for Data Analysis

O'REILLY®

Wes McKinney

Book on Amazon (https://www.amazon.com/dp/1491957662/ref=cm_sw_em_r_mt_dp_U_iyjPCb6QF2KWN)

You can also go to:

https://pandas.pydata.org/ (https://pandas.pydata.org/)

And get the full reference on the pandas library online.

Summary

In this brownbag we will:

Create Data - We begin by creating/extracting our own data set for analysis. We will store this data set to a csv file so that you can get some experience pulling data from a file.
Get Data - We will learn how to read from a file. Our data will be around Test Data coming from TestRail.
Prepare Data - Here we will simply take a look at the data and make sure it is clean. By clean I mean we will take a look inside the contents of the data file and look for any anomalies. These can include missing data, inconsistencies in the data, or any other data that seems out of place. If any are found we will then have to make decisions on what to do with these records.
Analyze Data - We will simply calculate a simple Tests passed metric for a specific Test Run.
Present Data - Through tabular data and a graph, clearly show the end user what is the status of a specific milestone or run.

Create/Get Data

The first data set will consist of the Test Statuses that we have defined in TestRail and their id.

Setup access to our Testrail instance

- We need credentials but it's safer to keep them stored in a file and tokenized

In [23]:
```
import yaml

with open("config.yml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile, Loader=yaml.FullLoader)

host = cfg['testrail']['host']
user = cfg['testrail']['user']
password = cfg['testrail']['token']
```

- Let's instantiate the access to TestRail and create a small function to retrieve what Test Statuses are supported currently.

In [24]:

```
from testrail_original import *
import pprint
import pandas as pd
import time
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_colwidth', -1)

# Setup Access to instance
client = APIClient(str(host))
client.user = user
client.password = cfg['testrail']['token']


def get_statuses():
    '''
    GET index.php?/api/v2/get_statuses
    '''
    statuses = client.send_get('get_statuses/')
    df_statuses = pd.DataFrame.from_records(statuses)
    return df_statuses

df_statuses = get_statuses()
df_statuses
```

Out[24]:

	color_bright	color_dark	color_medium	id	is_final	is_system	is_untested	label	name
0	4901251	45136	45136	1	True	True	False	Passed	passed
1	7368320	1118482	1118482	2	True	True	False	Blocked	blocked
2	15790320	11579568	15395562	3	False	True	True	Untested	untested
3	15781221	16760832	16760832	4	False	True	False	Retest	retest
4	13177876	14813465	14813465	5	True	True	False	Failed	failed
5	14385169	14385169	14385169	6	True	False	False	Not Applicable	not_applicable
6	9519093	9519093	9519093	7	False	False	False	Not Delivered	not_delivered

Cleaning data and saving to file

There are too many columns and some are not useful for us... Let's **clean some data**. And store it in a **csv** file.

In [25]:
```
df_statuses = df_statuses.loc[:, ['id', 'label']]
df_statuses.rename(columns={'id': 'status_id'}, inplace=True)
df_statuses
```

Out[25]:

	status_id	label
0	1	Passed
1	2	Blocked
2	3	Untested
3	4	Retest
4	5	Failed
5	6	Not Applicable
6	7	Not Delivered

- Let's Export the dataframe to a ***csv*** file. We can name the file ***TestRail_test_statuses.csv***. The function ***to_csv*** will be used to export the file. The file will be saved in the same location of the notebook unless specified otherwise.
- If I have doubts about the function to use I can just ask for it's description

In [ ]:
```
df_statuses.to_csv?
```

The only parameters we will use are ***index*** and ***header***. Setting these parameters to False will prevent the index and header names from being exported. Change the values of these parameters to get a better understanding of their use.

In [26]:
```
df_statuses.to_csv('TestRail_test_statuses.csv',index=False,header=True)
```

Get Data via REST API

Let's get some more data that allows for a more meaningful analysis. Check the results from a specific run. More data on the types of data we can extract from **TestRail** is available in this link (http://docs.gurock.com/testrail-api2/start).

In [27]:
```
results = client.send_get('get_run/9909')

print (results)
```

```
{'id': 9909, 'suite_id': 4571, 'name': 'NOC Portal - Automation - Smoke Tests - Disco
very - 12032019-1405', 'description': None, 'milestone_id': 1112, 'assignedto_id': 7
5, 'include_all': True, 'is_completed': False, 'completed_on': None, 'config': None,
'config_ids': [], 'passed_count': 36, 'blocked_count': 0, 'untested_count': 2, 'retes
t_count': 0, 'failed_count': 6, 'custom_status1_count': 0, 'custom_status2_count': 0,
'custom_status3_count': 0, 'custom_status4_count': 0, 'custom_status5_count': 0, 'cus
tom_status6_count': 0, 'custom_status7_count': 0, 'project_id': 84, 'plan_id': None,
'created_on': 1552411401, 'created_by': 75, 'url': 'https://celfocus.testrail.net/ind
ex.php?/runs/view/9909'}
```

mmmm... This looks a little hard to read

If only there was a way to clean this data into tabular format simply...

In [28]:
```
df_results = pd.DataFrame.from_dict(
    results, orient='index', columns=['Values'])

df_results = df_results.T
df_results
```

Out[28]:

	id	suite_id	name	description	milestone_id	assignedto_id	include_all	is_completed	completed_on	config	...	custo
Values	9909	4571	NOC Portal - Automation - Smoke Tests - Discovery - 12032019-1405	None	1112	75	True	False	None	None	...	0

1 rows × 28 columns

Clean up data

Again we have to much data and can prune some columns

In [29]:
```
df_results = df_results.loc[:, ['id',
                                'name',
                                'passed_count',
                                'blocked_count',
                                'untested_count',
                                'retest_count',
                                'failed_count',]]

df_results
```

Out[29]:

	id	name	passed_count	blocked_count	untested_count	retest_count	failed_count
Values	9909	NOC Portal - Automation - Smoke Tests - Discovery - 12032019-1405	36	0	2	0	6

Let's now perform some calculations to know how many test cases we have and calculate progress, success and failure rates.

In [30]:
```
df_results['Total_Tests'] = df_results['passed_count'] + \
    df_results['blocked_count'] + \
    df_results['retest_count'] + df_results['failed_count']

df_results
```

Out[30]:

	id	name	passed_count	blocked_count	untested_count	retest_count	failed_count	Total_Tests
Values	9909	NOC Portal - Automation - Smoke Tests - Discovery - 12032019-1405	36	0	2	0	6	42

Calculate metrics

Calculated Ratios:

$$PassRate = \frac{PassedTests}{TotalTests}$$

$$FailRate = \frac{FailedTests}{TotalTests}$$

In [31]:
```
df_results['Pass_Rate'] = df_results['passed_count'] / df_results['Total_Tests']
df_results['Fail_Rate'] = df_results['failed_count'] / df_results['Total_Tests']

df_results
```

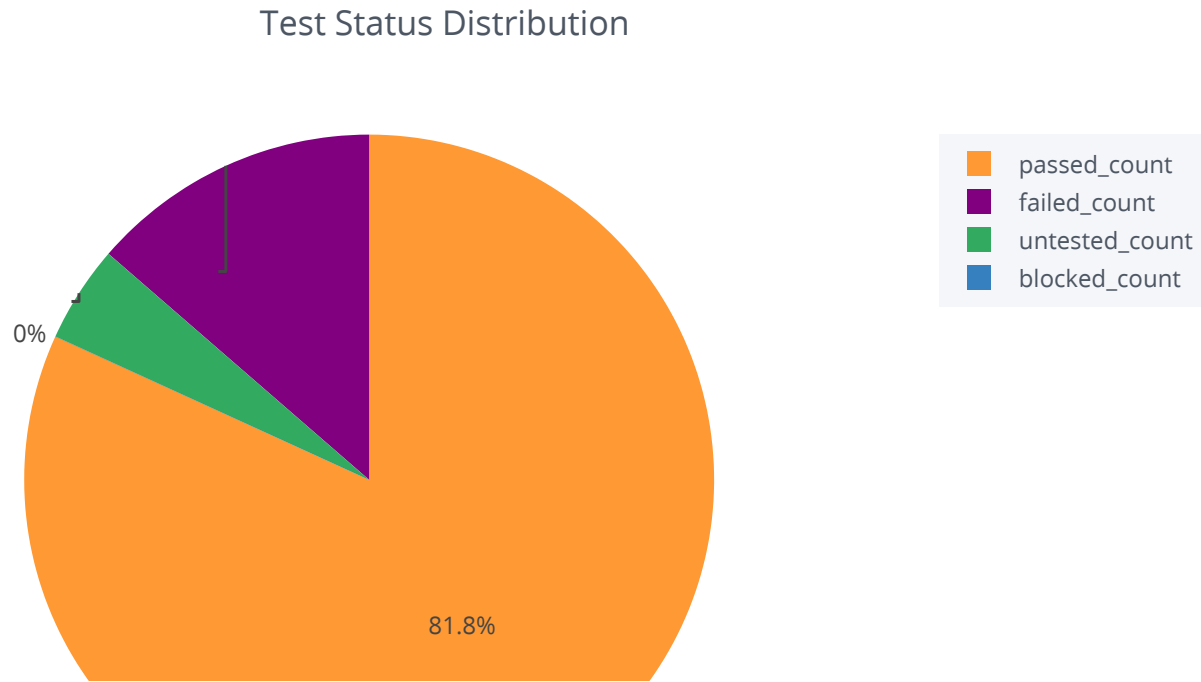
Out[31]:

	id	name	passed_count	blocked_count	untested_count	retest_count	failed_count	Total_Tests	Pass_Rate	Fail_Rate
Values	9909	NOC Portal - Automation - Smoke Tests - Discovery - 12032019-1405	36	0	2	0	6	42	0.857143	0.142857

Plot results

In [53]:

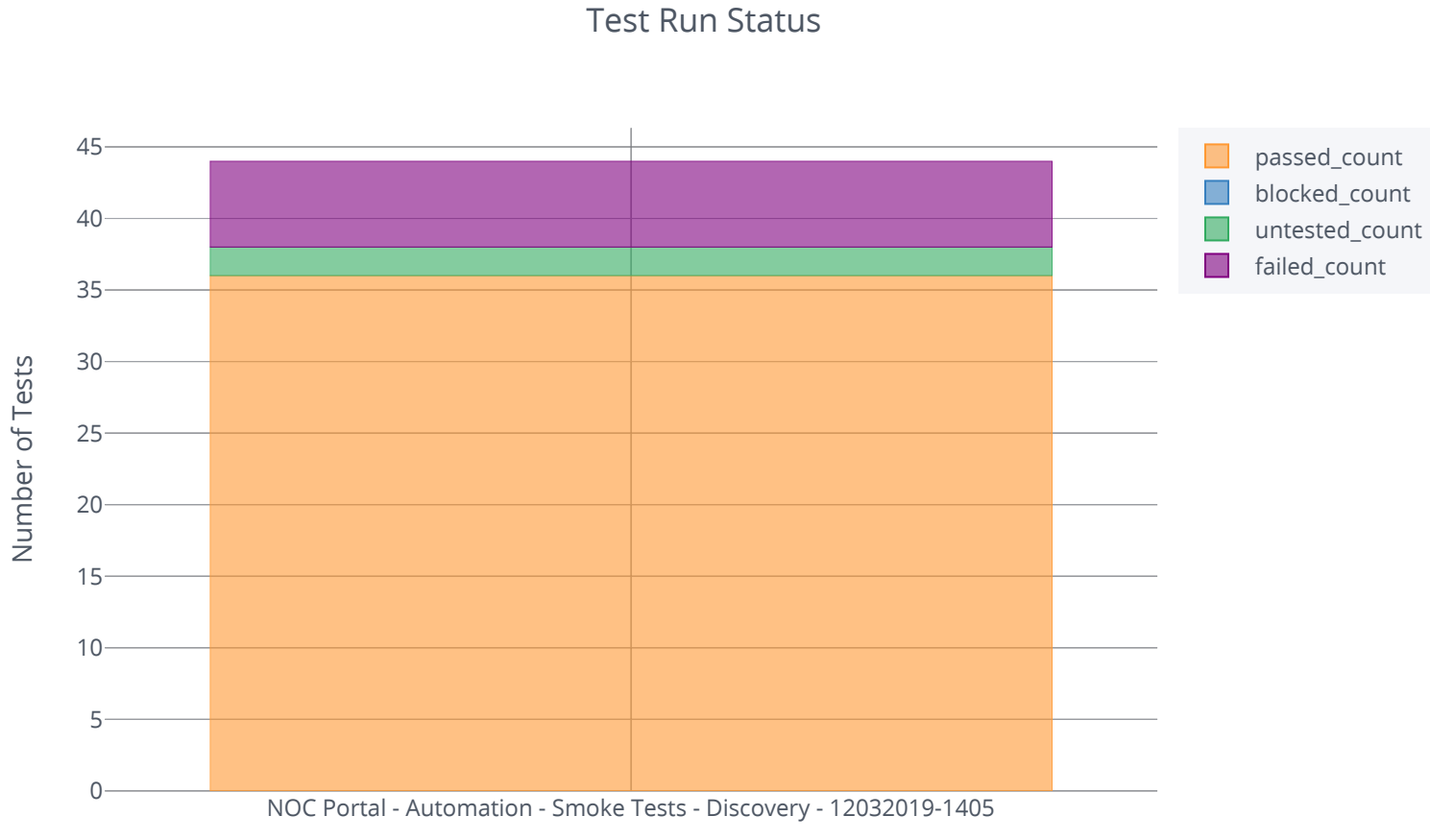
```
import plotly.graph_objs as go
import plotly as py
import ipywidgets as widgets
import cufflinks as cf

cf.go_offline()
py.offline.init_notebook_mode(connected=True)

df_pie = df_results[['passed_count', 'blocked_count',
                     'untested_count', 'failed_count']].T
df_pie = df_pie.reset_index()

df_pie
df_pie.iplot(kind='pie', labels='index', values='Values',
             title='Test Status Distribution')
```



- Let's plot another type of graph

In [51]:

```
df_bar_chart = df_results[['name','passed_count', 'blocked_count', 'untested_count', 'failed_count']]
df_bar_chart = df_bar_chart.set_index('name')
df_bar_chart


df_bar_chart.iplot(kind='bar', barmode='stack', yTitle='Number of Tests', title='Test Run Status')
```



Export to plot.ly »

More Complex scenario

What if I want to show something more complex like getting all results from a Test Run over time.

- Let's define a function that accepts the Run id as a parameter and retrieves the results over time via REST API returning a dataframe with those results.

In [35]:
```
def get_results_for_run(run_id):
    '''
    Create a dataframe with the status of the list of tests from a run or plan
    #GET index.php?/api/v2/get_results_for_run/:run_id
    '''
    results = client.send_get(
        'get_results_for_run/'+str(run_id)+'&status_id=1,5')
    df_results = pd.DataFrame.from_records(results)
    df_results['run_id'] = str(run_id)

    if 'created_on' in df_results.columns:
        df_results['created_on'] = pd.to_datetime(
            df_results['created_on'], unit='s')
        #df_results['created_on'] = df_results['created_on'].dt.date
    else:
        df_results = pd.DataFrame(
            columns=['defects', 'created_on', 'section_id', 'test_id'])
        df_results['created_on'] = "NA"

    df_results_filtered = df_results.loc[:, [
        'run_id', 'test_id', 'status_id', 'created_on', 'defects']]
    return df_results
```

- Let's run the function for an example Test Run from TestRail.

In [40]:
```
df_results_run = get_results_for_run('6974')
df_results_run = df_results_run.loc[:, ['created_on', 'status_id']]
df_results_run = df_results_run.dropna()
df_results_run = df_results_run.set_index('created_on')
df_results_run.head()
```

Out[40]:

created_on	status_id
2018-11-29 14:25:09	1
2018-11-20 11:04:14	1
2018-11-20 11:04:14	1
2018-11-16 10:42:59	1
2018-11-16 10:42:59	1

- Now let's plot using an interactive type of graph produced by Plotly.
    1. We will resample to daily status changes
    2. Sum the results per status per day
    3. Get the cumulative sum so we see the progress over time

In [46]:
```
import cufflinks as cf
import pandas as pd
import plotly as py

cf.go_offline()
py.offline.init_notebook_mode(connected=True)


df_sum_of_results_per_day = df_results_run.groupby(
    'status_id').resample('D')['status_id'].sum()

df_sum_of_results_per_day.head()
```

Out[46]:
```
status_id  created_on
1          2018-09-06    7
           2018-09-07    5
           2018-09-08    0
           2018-09-09    0
           2018-09-10    2
Name: status_id, dtype: int64
```

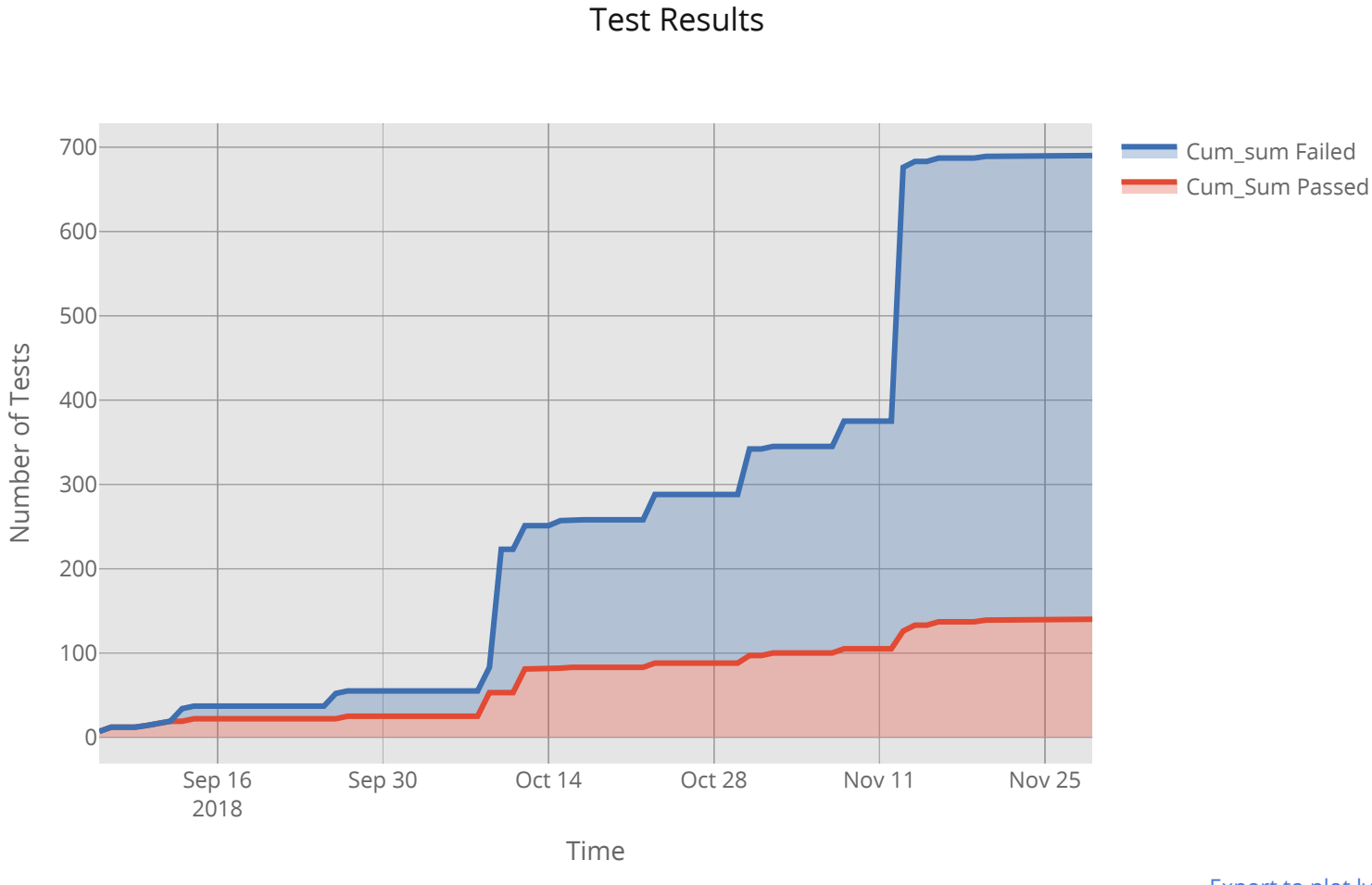
- Calculate Cumulative Sums for statuses

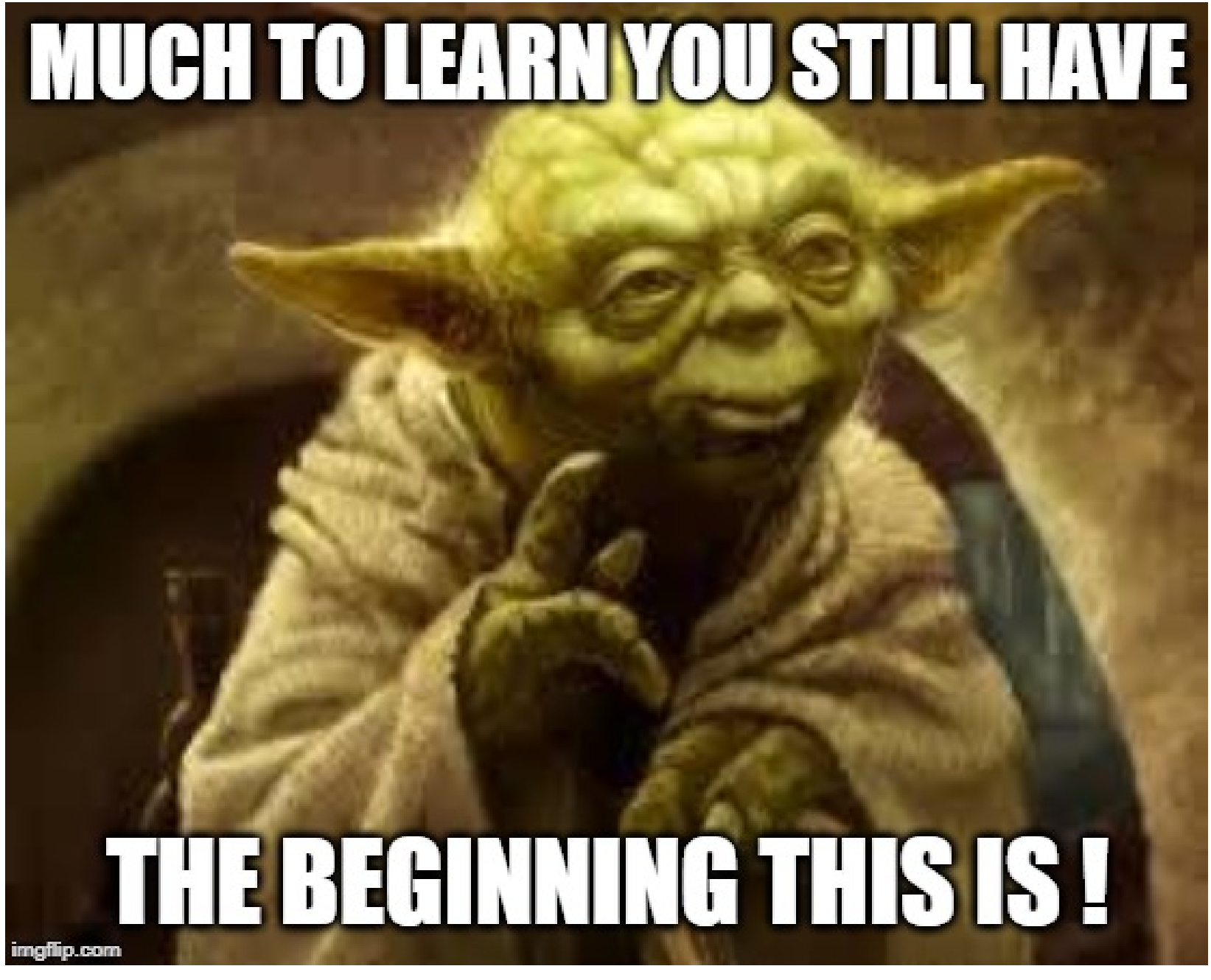
In [47]:
```
df_to_graph = df_sum_of_results_per_day

df_to_graph = df_to_graph.unstack('status_id')
# remove unwanted extra line due to unstack
df_to_graph = df_to_graph.reset_index()
df_to_graph = df_to_graph.fillna(0)

df_to_graph['cum_sum passed'] = df_to_graph[1].cumsum()
df_to_graph['cum_sum failed'] = df_to_graph[5].cumsum()

df_to_graph.columns = ['Date', 'Passed',
                       'Failed', 'Cum_Sum Passed', 'Cum_sum Failed']
df_to_graph = df_to_graph.set_index('Date')

df_to_graph = df_to_graph.drop(['Passed', 'Failed'], axis=1)

df_to_graph.tail()
```

Out[47]:

	Cum_Sum Passed	Cum_sum Failed
Date		
2018-11-25	139.0	550.0
2018-11-26	139.0	550.0
2018-11-27	139.0	550.0
2018-11-28	139.0	550.0
2018-11-29	140.0	550.0

In [52]:
```
df_to_graph.iplot(kind='area', fill=True,
                  title='Test Results', theme='ggplot', width=3, xTitle='Time',yTitle='N
umber of Tests')
```




MUCH TO LEARN YOU STILL HAVE
THE BEGINNING THIS IS !
imgflip.com