# Notebook

August 8, 2020

## 1 Visualization for monitoring status of DOI Collection.

Please note that this notebook is also used for generating automatic report during each run of the mpcite builder

### 1.0.1 Import

[1]:
```
import logging
import os
import json
from pathlib import Path
import plotly
from mpcite.models import ConnectionModel
from mpcite.doi_builder import DoiBuilder
import pandas as pd
from typing import Union, List, Dict
import plotly.express as px
import plotly.graph_objects as go

import numpy as np
import maggma
from datetime import timedelta, date, datetime
from monty.json import MontyDecoder
from datetime import timedelta

import plotly.graph_objs as go
import plotly.offline as py
```

```
/opt/anaconda3/envs/MPCite/lib/python3.7/site-
packages/maggma-0.20.1.dev36+g6fc637b-py3.7.egg/maggma/utils.py:20:
TqdmExperimentalWarning: Using `tqdm.autonotebook.tqdm` in notebook mode. Use
`tqdm.tqdm` instead to force console mode (e.g. in jupyter console)
  from tqdm.autonotebook import tqdm
```

### 1.0.2 Configure

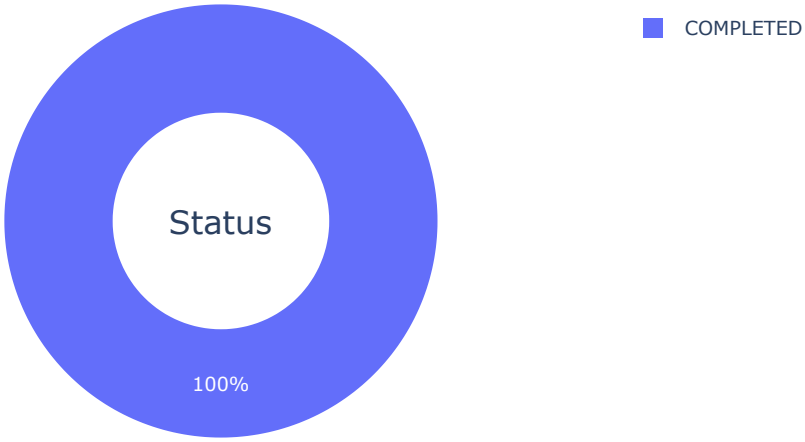
[2]:
```
# configuration stuff
config_file = Path("../files/config_prod.json")
assert config_file.exists(), "input config file does not exist"
bld: DoiBuilder = json.load(config_file.open("r"), cls=MontyDecoder)
```

[3]:
```
doi_store = bld.doi_store
materials_store = bld.materials_store
doi_store.connect()
materials_store.connect()

renderer = "pdf" # change to pdf for saving, notebook for viewing live
```

### 1.0.3 Create helper functions for drawing pie graphs

[4]:
```
def draw_pie_graph(labels:list, values:list, name:str):
    """
    Takes in labels and values and plot it as plotly pie graph
    """
    fig=go.FigureWidget(data=[go.Pie(labels=labels, values=values, hole=.5,␣
 ↪name=name)])
    fig.update_layout(annotations=[dict(text=name, font_size=20,␣
 ↪showarrow=False)])
    fig.show(renderer=renderer) # change renderer = "notebook" to view it live
```

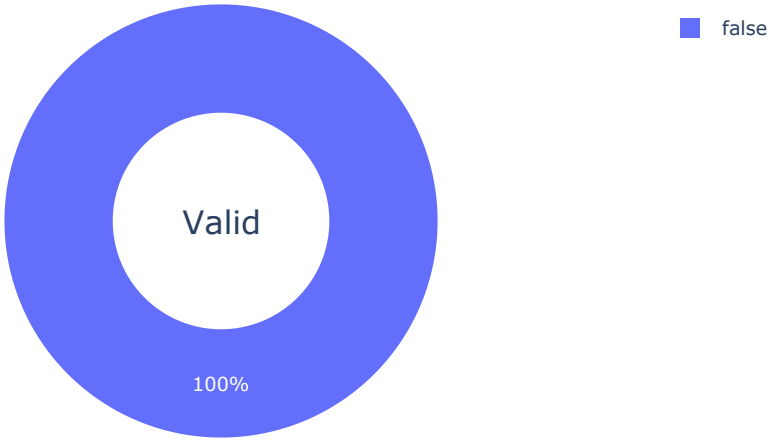
[5]:
```
def make_doi_status_pie_chart_data(doi_store):
    """
    Make DOI status Pie graph by finding out count of each distinct status
    """
    values = doi_store.distinct("status")
    result = dict()
    for v in values:
        result[v] = doi_store.count(criteria={"status":v})
    return result
data = make_doi_status_pie_chart_data(doi_store=doi_store)
draw_pie_graph(list(data.keys()), list(data.values()), "Status")
```




[ ]:

[ ]:

[6]:
```
def make_doi_valid_pie_chart_data(doi_store):
    """
    make doi Pie Graph data by finding out count of each true and false valid␣
 ↪values
    """
    values = doi_store.distinct("valid")
    result = dict()
    for v in values:
        result[v] = doi_store.count(criteria={"valid":v})
    return result
data = make_doi_valid_pie_chart_data(doi_store)
draw_pie_graph(labels = list(data.keys()), values=list(data.values()),␣
 ↪name="Valid")
```




[ ]:

[ ]:

### 1.1 Draw time dependent graphs

[7]:
```
def find_dates_btw(start_dt, end_dt):
    """
    find the number of dates between start date and end date
    """
    def daterange(date1, date2):
        for n in range(int((date2 - date1).days)+1):
            yield date1 + timedelta(n)
    dates = []
    for dt in daterange(start_dt, end_dt+timedelta(days=1)):
        date_format = dt.date()
        dates.append(datetime(date_format.year, date_format.month, date_format.
↪day))
    return dates

def find_earliest_date(store, field):
    """
    find the earliest record date
```

```
    """
    return list(store.query(criteria={}, sort={field:maggma.core.store.Sort.
 ↪Ascending}, limit=1))[0][field]

def find_latest_date(store, field):
    """
    find the latest_record date
    """
    return list(store.query(criteria={}, sort={field:maggma.core.store.Sort.
 ↪Descending}, limit=1))[0][field]
```

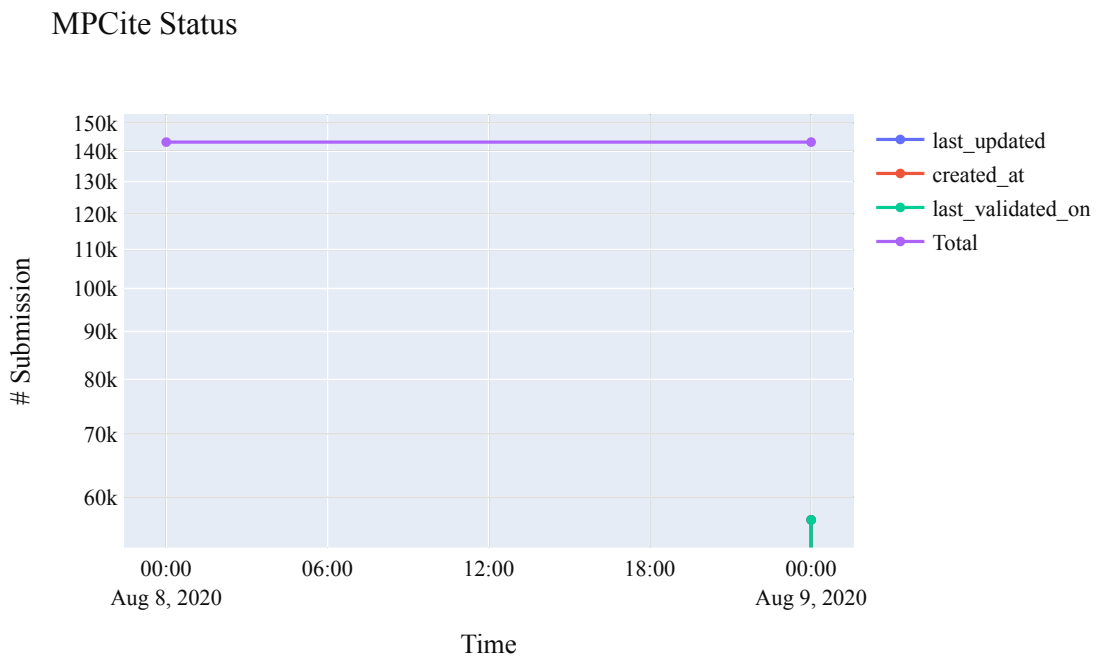
[8]:
```
def make_time_series_data(field_name):
    """
    Find all time series data for that field, put them in buckets of dates.
    """
    dates = find_dates_btw(find_earliest_date(doi_store, field_name),␣
 ↪find_latest_date(doi_store, field_name))
    # last_updated
    result = dict()
    for i in range(len(dates)):
        if i == 0:
            result[dates[i]] = 0
        else:
            c = doi_store.count(criteria={field_name: {"$lte": dates[i]}})
            result[dates[i]] = c
    return result
```

[ ]:

[ ]:

[9]:
```
# find data
last_updated_data = make_time_series_data("last_updated")
created_at_data = make_time_series_data("created_at")
last_validated_on_data = make_time_series_data("last_validated_on")
total = materials_store.count()

fig = go.Figure()

# find all the dates
Xs = set(last_updated_data.keys()).union(created_at_data.keys()).
 ↪union(last_validated_on_data.keys())

# plot last_updated
fig.add_trace(go.Scatter(x=list(last_updated_data.keys()),␣
 ↪y=list(last_updated_data.values()),
                    mode='lines+markers',
```

```
                    name='last_updated'))
# plot created at
fig.add_trace(go.Scatter(x=list(created_at_data.keys()), y=list(created_at_data.
 ↪values()),
                    mode='lines+markers',
                    name='created_at'))
# plot validated on
fig.add_trace(go.Scatter(x=list(last_validated_on_data.keys()),␣
 ↪y=list(last_validated_on_data.values()),
                    mode='lines+markers',
                    name='last_validated_on'))
# plot all materials count
fig.add_trace(go.Scatter(x=list(Xs), y=[total] * len(Xs),
                    mode='lines+markers',
                    name='Total'))
# add features
fig.update_layout(
    title="MPCite Status",
    xaxis_title="Time",
    yaxis_title="# Submission",
    font=dict(
        family="Franklin Gothic",
        size=14,
        color="#0d0d0d"
    ),
    yaxis_type="log",
)

fig.show(renderer=renderer) # change renderer = "notebook" to view it live
```


MPCite Status
150k
140k
130k
120k
110k
100k
90k
80k
70k
60k
# Submission
00:00
Aug 8, 2020
06:00
12:00
18:00
00:00
Aug 9, 2020
Time
last_updated
created_at
last_validated_on
Total

[ ]:

[ ]: