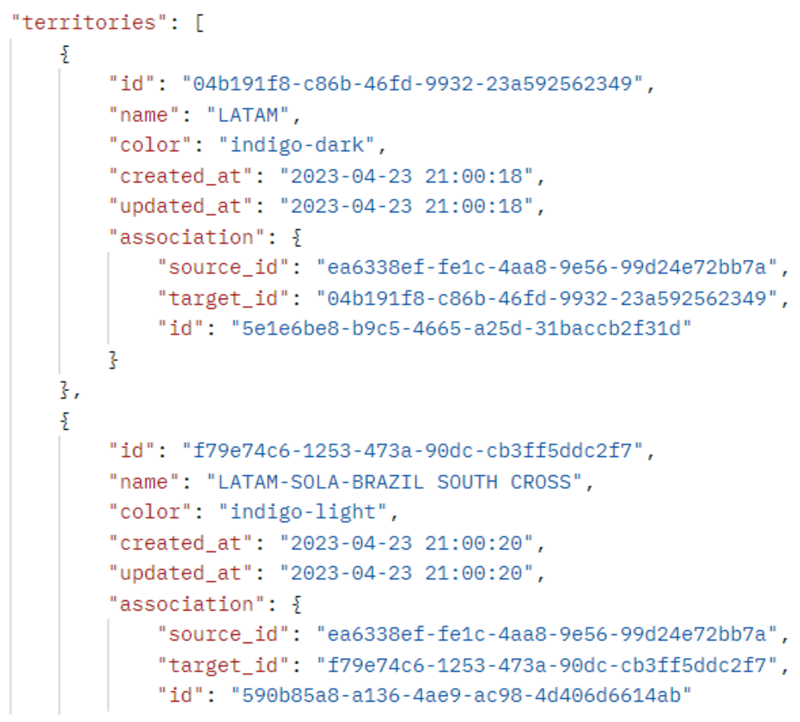
SuccessApp Memory Leak Incident

SuccessApp is one of many of our API's currently that is feeding data for reporting into our Heroku Environment. Per the API, there are new fields that are being added on a time basis, and some fields contain nested JSON which take a lot of resources when flattening in order to fit our model.

Example of a Nested JSON Structure from SuccessApp:

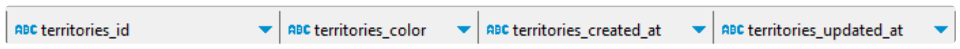


```
"territories": [
    {
        "id": "04b191f8-c86b-46fd-9932-23a592562349",
        "name": "LATAM",
        "color": "indigo-dark",
        "created_at": "2023-04-23 21:00:18",
        "updated_at": "2023-04-23 21:00:18",
        "association": {
            "source_id": "ea6338ef-fe1c-4aa8-9e56-99d24e72bb7a",
            "target_id": "04b191f8-c86b-46fd-9932-23a592562349",
            "id": "5e1e6be8-b9c5-4665-a25d-31baccb2f31d"
        }
    },
    {
        "id": "f79e74c6-1253-473a-90dc-cb3ff5ddc2f7",
        "name": "LATAM-SOLA-BRAZIL SOUTH CROSS",
        "color": "indigo-light",
        "created_at": "2023-04-23 21:00:20",
        "updated_at": "2023-04-23 21:00:20",
        "association": {
            "source_id": "ea6338ef-fe1c-4aa8-9e56-99d24e72bb7a",
            "target_id": "f79e74c6-1253-473a-90dc-cb3ff5ddc2f7",
            "id": "590b85a8-a136-4ae9-ac98-4d406d6614ab"
```

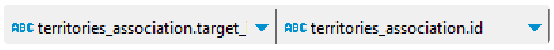
Notice how there are multiple fields under the parent Territories field. On the reporting side, we can't use a nested JSON so flattening it out into it's separate column is the requirement.

This is just one of many parent fields that SuccessApp contained, but below is an example of how territories is flattened inside Heroku.

Heroku Flattened out Column Structure:



ABC territories_id	ABC territories_color	ABC territories_created_at	ABC territories_updated_at

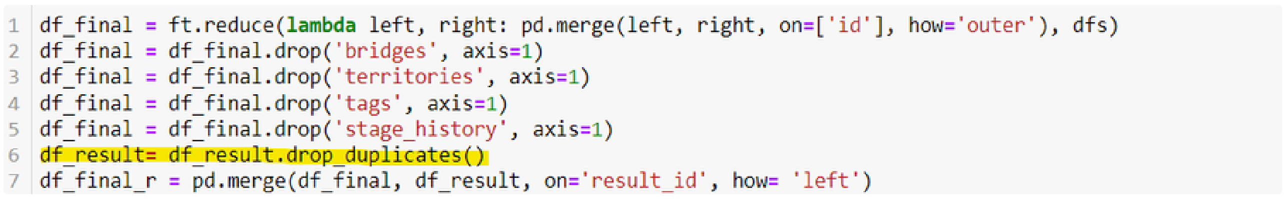


ABC territories_association.target_	ABC territories_association.id

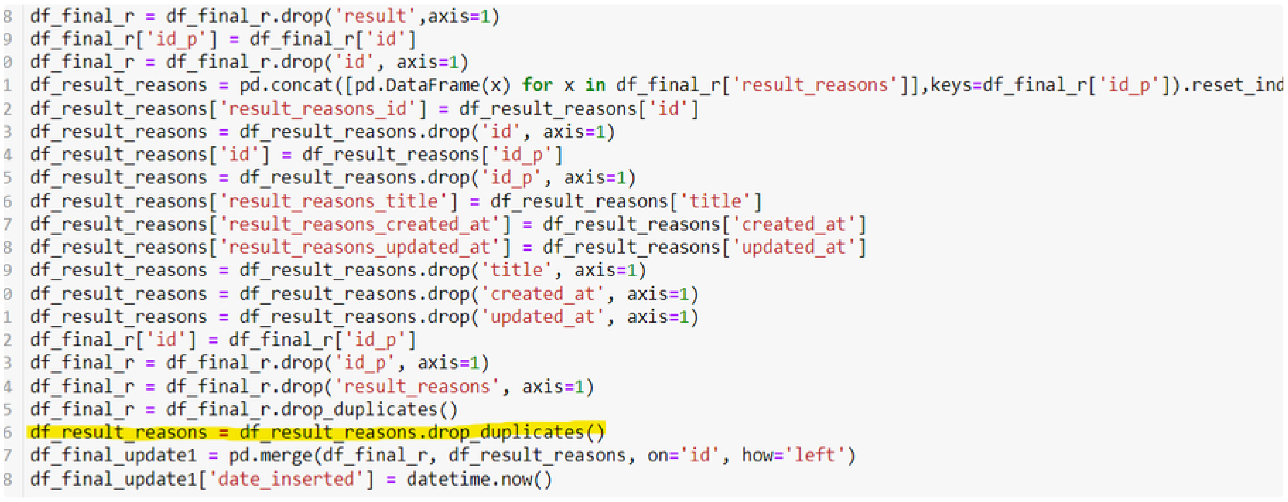
These nested fields are very expensive in memory especially when merging everything into a single data frame so optimization is essential when sending into Heroku. Currently our Heroku environment is on a private-L dyno which supports up to **14GB** of ram.

Optimization:

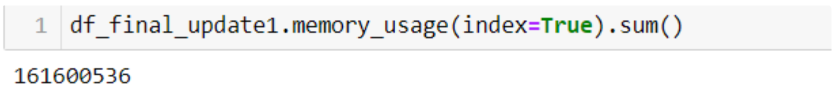
SuccessApp also contains many duplicate records and after joining each parent field dataset together and using the final one to write to Postgres, it totaled up to **135GB** of ram. The solution to optimizing such a huge dataset is to drop duplicate records **BEFORE** each merge.



```
df_final = ft.reduce(lambda left, right: pd.merge(left, right, on=['id'], how='outer'), dfs)
df_final = df_final.drop('bridges', axis=1)
df_final = df_final.drop('territories', axis=1)
df_final = df_final.drop('tags', axis=1)
df_final = df_final.drop('stage_history', axis=1)
df_result= df_result.drop_duplicates()
df_final_r = pd.merge(df_final, df_result, on='result_id', how= 'left')
```



```
df_final_r = df_final_r.drop('result',axis=1)
df_final_r['id_p'] = df_final_r['id']
df_final_r = df_final_r.drop('id', axis=1)
df_result_reasons = pd.concat([pd.DataFrame(x) for x in df_final_r['result_reasons']],keys=df_final_r['id_p']).reset_ind
df_result_reasons['result_reasons_id'] = df_result_reasons['id']
df_result_reasons = df_result_reasons.drop('id', axis=1)
df_result_reasons['id'] = df_result_reasons['id_p']
df_result_reasons = df_result_reasons.drop('id_p', axis=1)
df_result_reasons['result_reasons_title'] = df_result_reasons['title']
df_result_reasons['result_reasons_created_at'] = df_result_reasons['created_at']
df_result_reasons['result_reasons_updated_at'] = df_result_reasons['updated_at']
df_result_reasons = df_result_reasons.drop('title', axis=1)
df_result_reasons = df_result_reasons.drop('created_at', axis=1)
df_result_reasons = df_result_reasons.drop('updated_at', axis=1)
df_final_r['id'] = df_final_r['id_p']
df_final_r = df_final_r.drop('id_p', axis=1)
df_final_r = df_final_r.drop('result_reasons', axis=1)
df_final_r = df_final_r.drop_duplicates()
df_result_reasons = df_result_reasons.drop_duplicates()
df_final_update1 = pd.merge(df_final_r, df_result_reasons, on='id', how='left')
df_final_update1['date_inserted'] = datetime.now()
```



```
df_final_update1.memory_usage(index=True).sum()
```

```
161600536
```

The final merged dataset we get totals to just 161600536 bytes which is only just **0.161600536GB** of ram. Dropping duplicates before a huge merge can be beneficial and should be considered whenever large datasets are involved!