Analysis and Visualization of Solar Flares

Sudheer Nimmagadda-16265787

Sameer Yarlagadda-16271011

Motivation and Introduction:

A flare is defined as a sudden, rapid, and intense variation in brightness.
A solar flare occurs when magnetic energy that has built up in the solar atmosphere is suddenly released.
Radiation is emitted across virtually the entire electromagnetic spectrum, from radio waves at the long wavelength end, through visible light to x-rays and gamma rays.
The amount of energy released could power the whole world for 10 million years! On the other hand, it is less than one-tenth of the total energy emitted by the Sun every second. The first solar flare recorded in astronomical literature was on September 1, 1859.

How can you view a solar flare:

Typically, a person cannot view a solar flare by simply staring at the Sun.
Flares are in fact difficult to see because the Sun is already so bright. Instead, specialized scientific instruments are used to detect the light emitted during a flare.
Radio and optical emission from flares can be observed with telescopes on Earth.
Energetic emission such as x-rays and gamma-rays require telescopes located in space, since these emissions, thankfully, do not penetrate Earth's atmosphere.
Eg: Geostationary Operational Environmental Satellite-GOES, Solar Dynamics Observatory (SDO).

Classification of Solar Flares:

Solar flares are classified as A, B, C, M or X according to the peak flux (in watts per square metre, W/m2) of 1 to 8 Angstroms X-rays near Earth, as measured by XRS instrument on-board the GOES-15 satellite which is in a geostationary orbit over the Pacific Ocean.

A&B class: The A & B-class are the lowest class of solar flares.

C class: C-class solar flares are minor solar flares that have little to no effect on Earth.

M Class: M-class solar flares are the medium large solar flares. They cause small to moderate radio blackouts.

X Class: X-class solar flares are the biggest and strongest of them all. Strong to extreme radio blackouts occur on the daylight side of the Earth during the solar flare.

Dataset Description and Mapping between SDO and GOES:

GOES Database:

- Consider flares with a Geostationary Operational Environmental Satellite (GOES) X-ray flux peak magnitude above the M1.0 level (major flares)
- **Positive event** to be an active region that flares with a **peak magnitude above the M1.0** level, as defined by the GOES database.
- A **negative event** would be an active region that does not have such an event within a 24-hour time span.

- For collection active region for the negative class, we will also use information about all regions where X-ray flux peak magnitude above the C1.0 level.

SDO Database:

The Solar Dynamic's Observatory's Helioseismic and Magnetic Imager(HMI) instrument used to continuously map the vector magnetic field of the sun. Using this data, we can characterize active regions on the sun.

The SHARP data series provide maps in patches that encompass automatically tracked magnetic concentrations for their entire lifetime.we focused on 18 parameters calculated using the SHARP vector magnetic field data. Unsigned flux, Mean shear angle, Mean current helicity, Gradient of total field, Gradient of vertical field, Gradient of horizontal field etc.

- USFLUX: total unsigned flux.
- MEANGAM: mean angle of field from radial.
- MEANGBT: mean gradient of total field.
- MEANGBZ: mean gradient of vertical field.
- MEANGBH: mean gradient of the horizontal field.
- MEANJZD: mean vertical current density.
- TOTUSJZ: total unsigned vertical current.
- MEANALP: mean characteristic twist parameter.
- MEANJZH: mean current helicity.

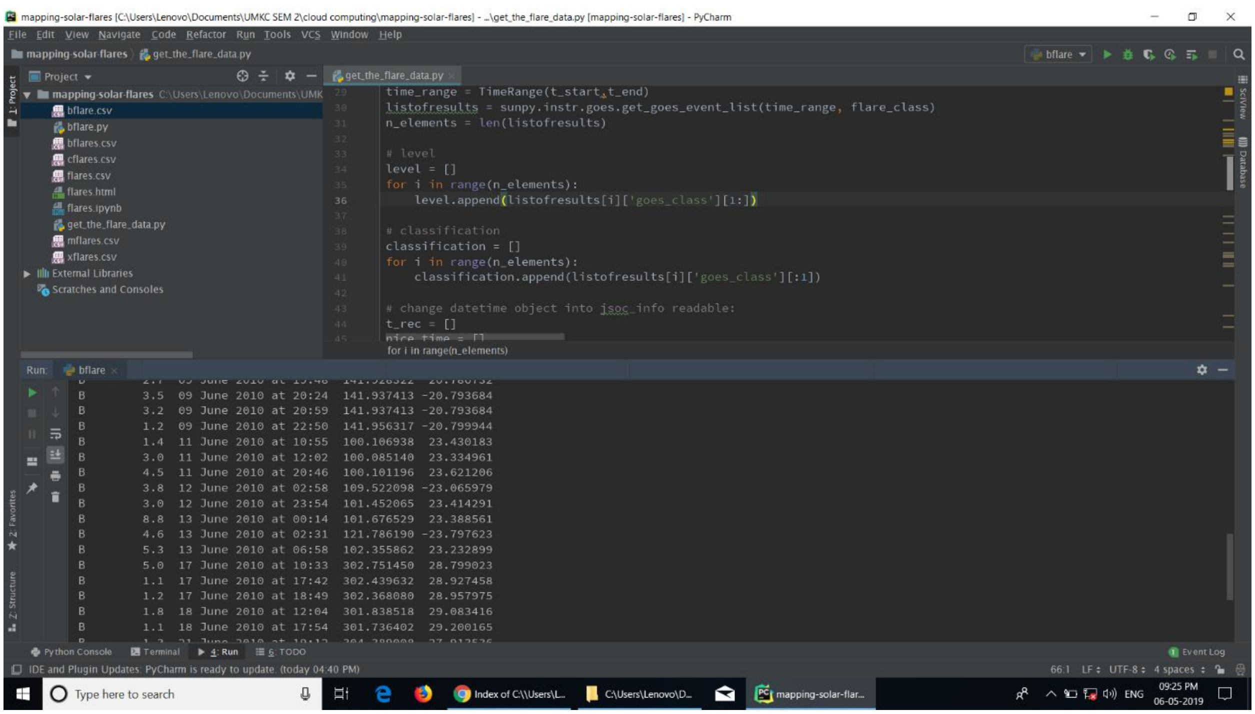
- TOTUSJH: total unsigned vertical current.
- ABSNJZH is the absolute value of the net current helicity.
- SAVNCPP: sum of the modulus of the net current per polarity.
- MEANPOT: mean photospheric magnetic free energy.
- TOTPOT: total photospheric magnetic free energy density.
- MEANSHR: mean shear angle.
- SHRGT45: fraction of area with shear greater than 45 degrees.
- R_VALUE: sum of flux near polarity inversion line.
- AREA_ACR: area of strong field pixels in the active region.

Modules:

1. **Visualization of Solar Flares Using D3.js:**

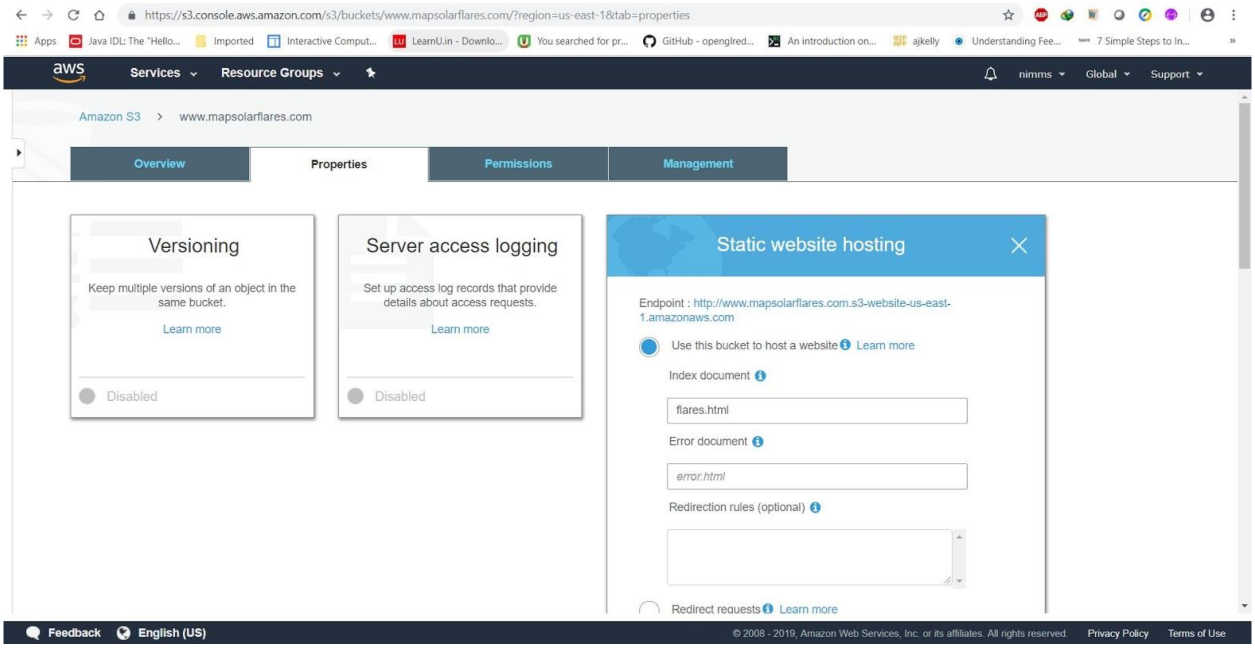
   Here we query from the GOES x-ray flux database for a list of active regions (along with parameters such as flare class, peak time) and to query the SDO JSOC database for longitude and Latitude of each flare active region. This will create a macro list of flare active regions with following parameters such as class, level, time, latitude, longitude.

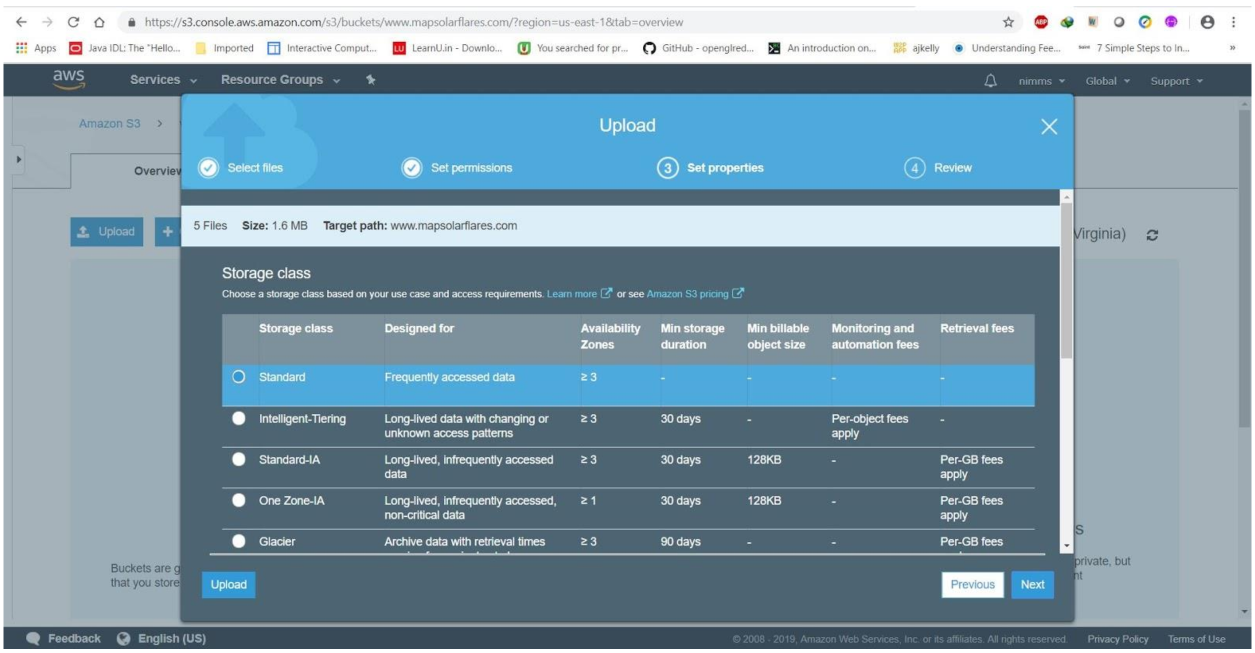
   The output is .csv file and can be read by the .html file to create a visualization of solar flares.

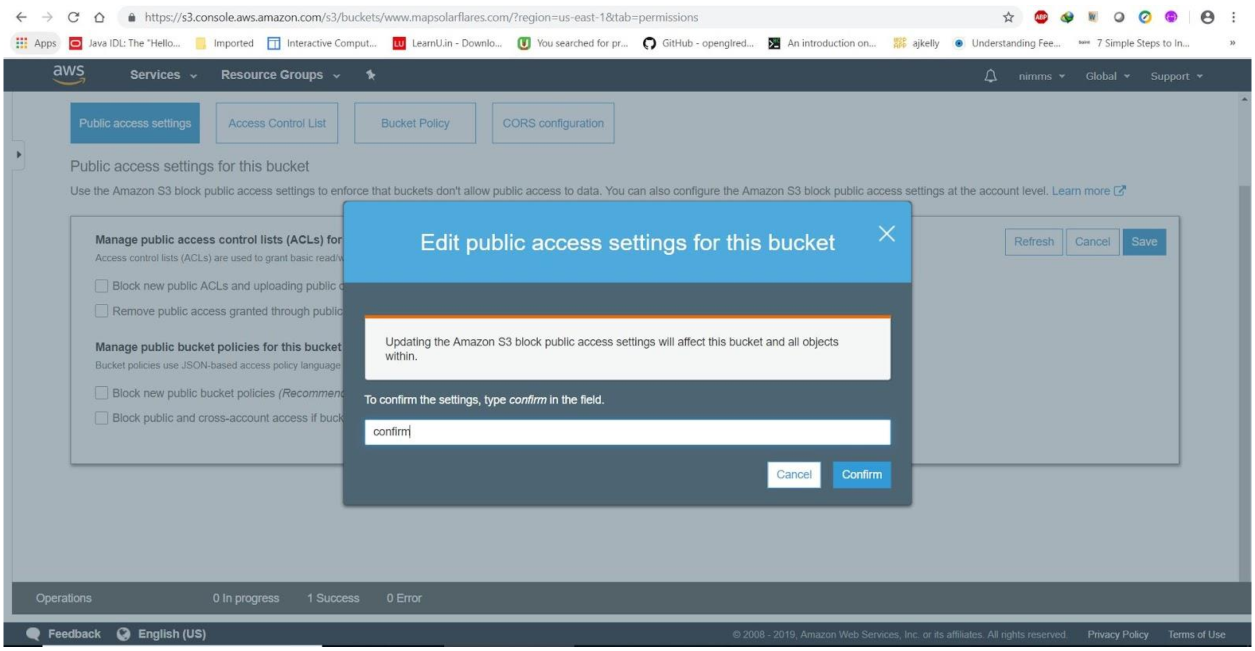


Inputs for the above code are start time, end time, flare class(eg X1), output file(Xflares.csv)

We hosted this visualization using AWS Static website hosting S3 bucket.

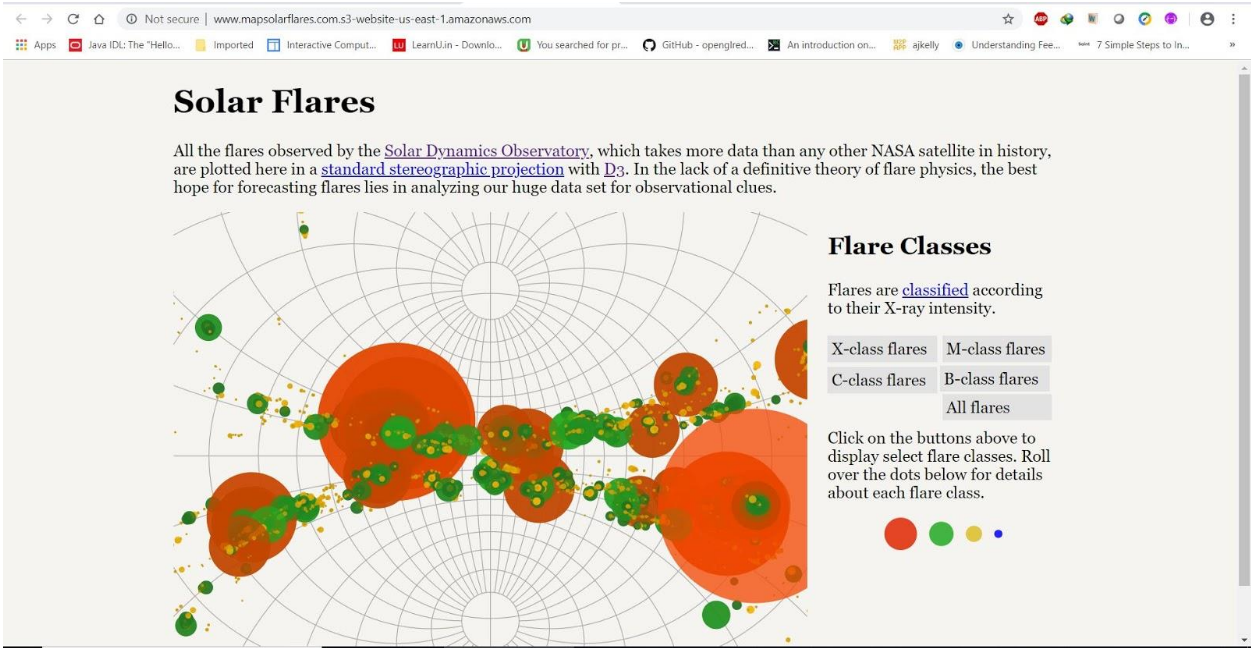





https://s3.console.aws.amazon.com/s3/buckets/www.mapsolarflares.com/?region=us-east-1&tab=permissions
Services
Resource Groups
Public access settings
Access Control List
Bucket Policy
CORS configuration
Public access settings for this bucket
Use the Amazon S3 block public access settings to enforce that buckets don't allow public access to data. You can also configure the Amazon S3 block public access settings at the account level. Learn more
Manage public access control lists (ACLs) for
Block new public ACLs and uploading public
Remove public access granted through public
Manage public bucket policies for this bucket
Block new public bucket policies (Recommend
Block public and cross-account access if buck
Refresh
Cancel
Save
Edit public access settings for this bucket
Updating the Amazon S3 block public access settings will affect this bucket and all objects within.
To confirm the settings, type confirm in the field.
confirm
Cancel
Confirm
Operations
0 In progress
1 Success
0 Error
Feedback
English (US)
© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

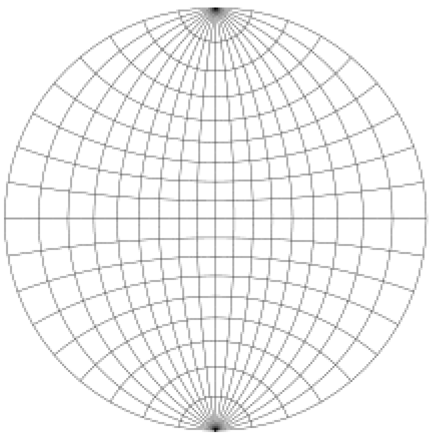
Stereographic projection of flares:

http://www.mapsolarflares.com.s3-website-us-east-1.amazonaws.com/



How stereographic projection is plotted:

Sphere is projected onto a plane using a mapping function. The projection is defined on the entire sphere, except at one point: the projection point. Where it is defined, the mapping is smooth and bijective.



Applications:

Planetary Science: The stereographic is the only projection that maps all circles on a sphere to circles on a plane. This property is valuable in planetary mapping where craters are typical features. The set of circles passing through the point of projection have unbounded radius, and therefore degenerate into lines.

Analysis of Solar Flares:

Active regions: The regions where the strength of magnetic field is high.

When does a flare occur: A flare occurs when the magnetic energy which was built up in the solar atmosphere is released suddenly.

There are a large number of different characteristics that can be used for magnetic field complexity description.

For HMI/SDO magnetograms, automated active region tracking exist known as SHARP (Space weather HMI Active Region Patch). For each active region, key features known as SHARP parameters were calculated. Calculation of these features are based on SDO magnetograms.

Here data is collected from two instruments: GOES and SDO. For handling Solar data there is special package “sunpy” in python.

```
DOWNLOAD = False
DATA_PATH = 'D:\\Cloud computing\\solar_flares'
if DOWNLOAD:
    time_range = TimeRange('2010/06/01 00:10', '2018/12/01 00:20')
    #time_range = TimeRange(t_start,t_end)
    goes_events = goes.get_goes_event_list(time_range,'C1')
    goes_events = pd.DataFrame(goes_events)
else:
    goes_events = pd.read_csv(os.path.join(DATA_PATH,'1.csv'), index_col=[0])

goes_events['noaa_active_region'] = goes_events['noaa_active_region'].replace(0,np.nan)
goes_events.dropna(inplace=True)
goes_events.drop(['goes_location','event_date','end_time','peak_time'], axis=1, inplace=True)
goes_events.start_time = goes_events.start_time.astype('datetime64[ns]')
```

```
      goes_class  noaa_active_region          start_time
8530        C8.1             12699.0 2018-02-07 13:31:00
8531        C1.7             12699.0 2018-02-07 14:35:00
8532        C4.6             12699.0 2018-02-10 13:02:00
8533        C1.5             12699.0 2018-02-12 00:15:00
8534        C1.9             12700.0 2018-03-02 10:56:00
8535        C4.6             12703.0 2018-03-30 07:57:00
8536        C2.0             12712.0 2018-05-23 18:03:00
8537        C2.7             12712.0 2018-05-28 17:04:00
8538        C1.0             12712.0 2018-06-06 10:44:00
8539        C2.1             12715.0 2018-06-21 01:09:00
```

Active region detection:

Different approaches are present to detect active regions. One of them is done by NOAA, correcting manually each day. Active regions have NOAA numbers. SDO has automated system for AR detections, the regions are known as HARPs. Also, SDO compute plenty of parameters of magnetic field complexity. So harp regions with features are used, but information about goes flux there is only for NOAA regions. There is no coincidence between HARP and NOAA regions, but they can be mapped. Below the code for mapping between the HARP and NOAA regions.

```
if os.path.isfile(os.path.join(DATA_PATH,'all_harps_with_noaa_ars.txt')):
    num_mapper = pd.read_csv(os.path.join(DATA_PATH,'all_harps_with_noaa_ars.txt'), sep=' ',index_col=[0])
else:
    num_mapper = pd.read_csv('http://jsoc.stanford.edu/doc/data/hmi/harpnum_to_noaa/all_harps_with_noaa_ars.txt',sep=' ')
    num_mapper.to_csv(os.path.join(DATA_PATH,'all_harps_with_noaa_ars.txt'), sep=' ')

print(num_mapper.tail())
```

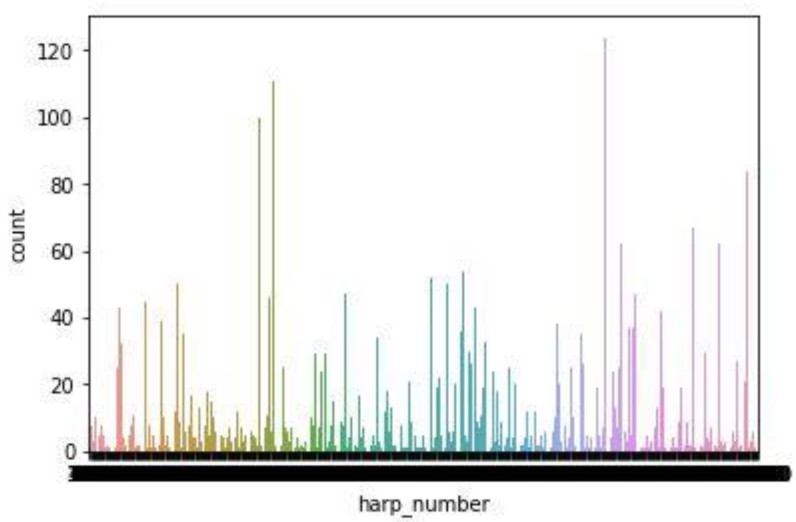
```
      HARPNUM NOAA_ARS
1314     7304    12721
1315     7305    12722
1316     7310    12723
1317     7312    12724
1318     7313    12725
```

```
def convert_noaa_to_harpnum(noaa_ar):

    idx = num_mapper[num_mapper['NOAA_ARS'].str.contains(str(int(noaa_ar)))]
    return None if idx.empty else int(idx.HARPNUM.values[0])
goes_events['harp_number'] = goes_events['noaa_active_region'].apply(convert_noaa_to_harpnum)
goes_events.dropna(inplace=True)

goes_events['flux'] =  goes_events['goes_class'].apply(lambda x: 1e06*goes.flareclass_to_flux(x).value)
print(goes_events.head())
```

	goes_class	noaa_active_region	start_time	harp_number	flux
0	M2.0	11081.0	2010-06-12 00:30:00	54.0	20.0
1	C1.0	11080.0	2010-06-12 03:57:00	51.0	1.0
2	C6.1	11081.0	2010-06-12 09:02:00	54.0	6.1
3	M1.0	11079.0	2010-06-13 05:30:00	49.0	10.0
4	C1.2	11081.0	2010-06-13 06:08:00	54.0	1.2

Count plot for Harp Number:

```
import seaborn as sns
sns.countplot(x='harp_number', data = goes_events)
```



Loading data:

Active regions with main features are taken from SDO database. In Python, a special package 'drms' is used for downloading meta information with all the 18 parameters(T_REC,USFLUX etc..)

```python
import drms
c = drms.Client()
list_keywords = ['T_REC,CRVAL1,CRLN_OBS,USFLUX,MEANGBT,MEANJZH,MEANPOT,SHRGT45,TOTUSJH,MEANGBH,MEANALP,MEANGAM,MEANGBZ,MEANJZD,
harp_list = pd.unique(goes_events.harp_number)
N=len(harp_list)
k=0
for harp in harp_list:
    str_query = f'hmi.sharp_cea_720s[{str(int(harp))}]'

    if os.path.isfile(os.path.join(DATA_PATH+'\\keys_regions',str_query+'.csv')):
        #print(f'Harp number {harp} already exist\n')
        k=k+1
    else:
        print(f'load region with Harp number {harp}')
        keys = c.query(str_query, key=list_keywords)
        keys.to_csv(os.path.join(DATA_PATH+'\\keys_regions',str_query+'.csv'))
```

hmi.sharp_cea_720s[49]	5/9/2019 2:00 AM	Microsoft Excel C...	147 KB
hmi.sharp_cea_720s[51]	5/9/2019 2:00 AM	Microsoft Excel C...	107 KB
hmi.sharp_cea_720s[54]	5/9/2019 2:00 AM	Microsoft Excel C...	111 KB
hmi.sharp_cea_720s[86]	5/9/2019 2:00 AM	Microsoft Excel C...	378 KB
hmi.sharp_cea_720s[92]	5/9/2019 2:00 AM	Microsoft Excel C...	398 KB
hmi.sharp_cea_720s[104]	5/9/2019 2:00 AM	Microsoft Excel C...	400 KB
hmi.sharp_cea_720s[115]	5/9/2019 2:00 AM	Microsoft Excel C...	418 KB
hmi.sharp_cea_720s[156]	5/9/2019 2:00 AM	Microsoft Excel C...	196 KB
hmi.sharp_cea_720s[185]	5/9/2019 2:00 AM	Microsoft Excel C...	353 KB
hmi.sharp_cea_720s[187]	5/9/2019 2:00 AM	Microsoft Excel C...	370 KB
hmi.sharp_cea_720s[190]	5/9/2019 2:00 AM	Microsoft Excel C...	267 KB
hmi.sharp_cea_720s[211]	5/9/2019 2:01 AM	Microsoft Excel C...	375 KB
hmi.sharp_cea_720s[224]	5/9/2019 2:01 AM	Microsoft Excel C...	280 KB
hmi.sharp_cea_720s[226]	5/9/2019 2:01 AM	Microsoft Excel C...	410 KB
hmi.sharp_cea_720s[245]	5/9/2019 2:01 AM	Microsoft Excel C...	376 KB
hmi.sharp_cea_720s[252]	5/9/2019 2:01 AM	Microsoft Excel C...	329 KB
hmi.sharp_cea_720s[274]	5/9/2019 2:01 AM	Microsoft Excel C...	242 KB
hmi.sharp_cea_720s[284]	5/9/2019 2:01 AM	Microsoft Excel C...	393 KB
hmi.sharp_cea_720s[297]	5/9/2019 2:01 AM	Microsoft Excel C...	339 KB
hmi.sharp_cea_720s[318]	5/9/2019 2:01 AM	Microsoft Excel C...	197 KB
hmi.sharp_cea_720s[325]	5/9/2019 2:01 AM	Microsoft Excel C...	189 KB
hmi.sharp_cea_720s[327]	5/9/2019 2:01 AM	Microsoft Excel C...	284 KB
hmi.sharp_cea_720s[345]	5/9/2019 2:01 AM	Microsoft Excel C...	374 KB
hmi.sharp_cea_720s[362]	5/9/2019 2:01 AM	Microsoft Excel C...	223 KB

Defining target:

- **Positive event** to be an active region that flares with a **peak magnitude above the M1.0** level, as defined by the GOES database.

- A **negative event** would be an active region that does not have such an event within a 24-hour time span.

```
def compute_target(full_df, goes_events=goes_events, horizont = 24, level = 10):
    harp = full_df['HARP'][0]
    big_events = goes_events[(goes_events['harp_number']==harp) &(goes_events.flux>level)]
    target = pd.Series(full_df.index.map(lambda x: np.sum((x>big_events.start_time - np.timedelta64(horizont,'h'))
            & (x<big_events.start_time))))
    full_df['target'] = target.values
    return full_df
full_df=compute_target(full_df)
```

Computing the previous flux:

Previous flux is calculated up to considered moment.

```
def compute_prev_flux(full_df, goes_events=goes_events):
    harp = full_df['HARP'][0]

    goes_harp = goes_events[(goes_events['harp_number']==harp)]
    prev_flux = pd.Series(full_df.index.map(lambda x: goes_harp.loc[goes_harp.start_time<x].flux.sum()))
    full_df['prev_flux'] = prev_flux.values
    return full_df
full_df=compute_prev_flux(full_df)
```

	USFLUX	MEANGBT	MEANJZH	...	HARP	target	prev_flux
2010-07-20 16:00:00	2.873286e+22	86.112	0.001684	...	92.0	0	1.4
2010-07-20 18:00:00	2.987362e+22	81.151	0.001855	...	92.0	0	1.4
2010-07-20 20:00:00	3.020606e+22	82.845	0.002830	...	92.0	0	1.4
2010-07-20 22:00:00	3.090511e+22	86.301	0.003532	...	92.0	0	1.4
2010-07-21 00:00:00	3.245146e+22	85.072	0.003725	...	92.0	0	1.4

```
DOWNLOAD = True
if DOWNLOAD and os.path.isfile(os.path.join(DATA_PATH, 'solar_train.pkl')):
    print('Download prepared train data')
    train_df = pd.read_pickle(os.path.join(DATA_PATH, 'solar_train.pkl'))
else:
    df_list=[]
    for harp in tqdm(harp_list):
        df_ = extract_features_from_csv(harp)
        if df_.shape[0]==0:
            continue
        df_ = compute_target(df_, goes_events=goes_events)
        df_ = compute_prev_flux(df_)
        df_['Time'] = df_.index
        df_list.append(df_)
    train_df = pd.concat(df_list, ignore_index=True).sort_values(by = 'Time')
    train_df.to_pickle(os.path.join(DATA_PATH, 'solar_train.pkl'))

print(train_df.target.value_counts())


train_df['bin_target'] = train_df.target.map(lambda x: 0 if x==0 else 1)
print(train_df['bin_target'].value_counts())
```

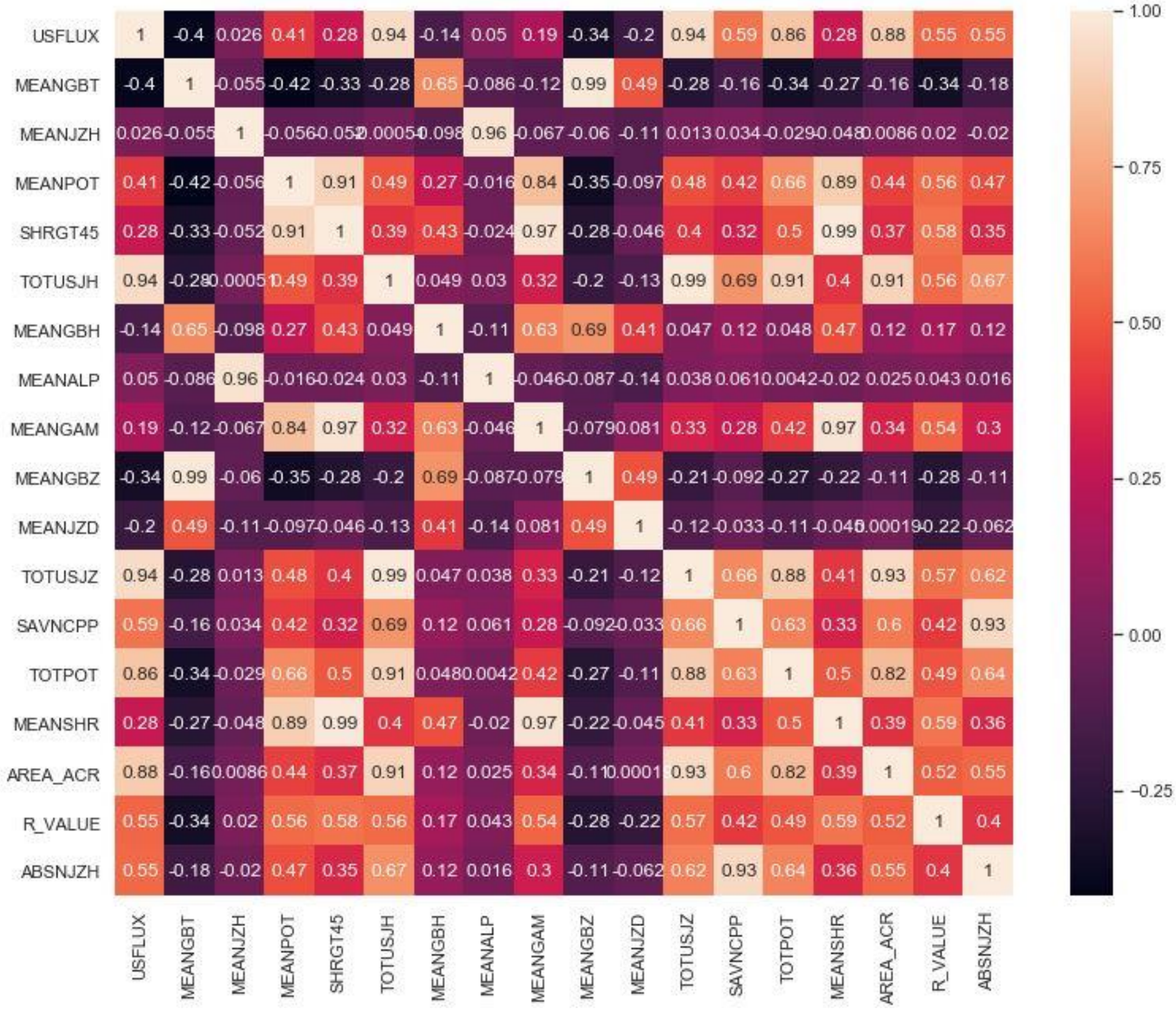
As strength of flare increases, the number of flares decreases implies strongest flares are very rare.

```
0     58409
1      2080
2       563
3       225
4        91
5        73
6        30
7        13
8         8
9         7
10        2
11        1
Name: target, dtype: int64
0    58409
1     3093
Name: bin_target, dtype: int64
```

```
Download prepared train data
0    58409
1     3093
Name: bin_target, dtype: int64
```

Correlation between 18 magnetic parameters:

```
key_cols = str.split('USFLUX,MEANGBT,MEANJZH,MEANPOT,SHRGT45,TOTUSJH,MEANGBH,MEANALP,MEANGAM,MEANGBZ,MEANJZD,TOTUSJZ,SAVNCPP,
sns.set(rc={'figure.figsize':(12.7,10.27)})
sns.heatmap(train_df[key_cols].corr(), annot=True)
```


USFLUX 1 -0.4 0.026 0.41 0.28 0.94 -0.14 0.05 0.19 -0.34 -0.2 0.94 0.59 0.86 0.28 0.88 0.55 0.55
MEANGBT -0.4 1 -0.055 -0.42 -0.33 -0.28 0.65 -0.086 -0.12 0.99 0.49 -0.28 -0.16 -0.34 -0.27 -0.16 -0.34 -0.18
MEANJZH 0.026 -0.055 1 -0.056 -0.052 0.00051 -0.098 0.96 -0.067 -0.06 -0.11 0.013 0.034 -0.029 -0.048 0.0086 0.02 -0.02
MEANPOT 0.41 -0.42 -0.056 1 0.91 0.49 0.27 -0.016 0.84 -0.35 -0.097 0.48 0.42 0.66 0.89 0.44 0.56 0.47
SHRGT45 0.28 -0.33 -0.052 0.91 1 0.39 0.43 -0.024 0.97 -0.28 -0.046 0.4 0.32 0.5 0.99 0.37 0.58 0.35
TOTUSJH 0.94 -0.28 0.00051 0.49 0.39 1 0.049 0.03 0.32 -0.2 -0.13 0.99 0.69 0.91 0.4 0.91 0.56 0.67
MEANGBH -0.14 0.65 -0.098 0.27 0.43 0.049 1 -0.11 0.63 0.69 0.41 0.047 0.12 0.048 0.47 0.12 0.17 0.12
MEANALP 0.05 -0.086 0.96 -0.016 -0.024 0.03 -0.11 1 -0.046 -0.087 -0.14 0.038 0.061 0.0042 -0.02 0.025 0.043 0.016
MEANGAM 0.19 -0.12 -0.067 0.84 0.97 0.32 0.63 -0.046 1 -0.079 0.081 0.33 0.28 0.42 0.97 0.34 0.54 0.3
MEANGBZ -0.34 0.99 -0.06 -0.35 -0.28 -0.2 0.69 -0.087 -0.079 1 0.49 -0.21 -0.092 -0.27 -0.22 -0.11 -0.28 -0.11
MEANJZD -0.2 0.49 -0.11 -0.097 -0.046 -0.13 0.41 -0.14 0.081 0.49 1 -0.12 -0.033 -0.11 -0.045 0.00019 -0.22 -0.062
TOTUSJZ 0.94 -0.28 0.013 0.48 0.4 0.99 0.047 0.038 0.33 -0.21 -0.12 1 0.66 0.88 0.41 0.93 0.57 0.62
SAVNCPP 0.59 -0.16 0.034 0.42 0.32 0.69 0.12 0.061 0.28 -0.092 -0.033 0.66 1 0.63 0.33 0.6 0.42 0.93
TOTPOT 0.86 -0.34 -0.029 0.66 0.5 0.91 0.048 0.0042 0.42 -0.27 -0.11 0.88 0.63 1 0.5 0.82 0.49 0.64
MEANSHR 0.28 -0.27 -0.048 0.89 0.99 0.4 0.47 -0.02 0.97 -0.22 -0.045 0.41 0.33 0.5 1 0.39 0.59 0.36
AREA_ACR 0.88 -0.16 0.0086 0.44 0.37 0.91 0.12 0.025 0.34 -0.11 0.0001 0.93 0.6 0.82 0.39 1 0.52 0.55
R_VALUE 0.55 -0.34 0.02 0.56 0.58 0.56 0.17 0.043 0.54 -0.28 -0.22 0.57 0.42 0.49 0.59 0.52 1 0.4
ABSNJZH 0.55 -0.18 -0.02 0.47 0.35 0.67 0.12 0.016 0.3 -0.11 -0.062 0.62 0.93 0.64 0.36 0.55 0.4 1
USFLUX MEANGBT MEANJZH MEANPOT SHRGT45 TOTUSJH MEANGBH MEANALP MEANGAM MEANGBZ MEANJZD TOTUSJZ SAVNCPP TOTPOT MEANSHR AREA_ACR R_VALUE ABSNJZH
1.00
0.75
0.50
0.25
0.00
−0.25

Model Selection:

Logistic Regression:

```python
train_df['Year'] = train_df.Time.dt.year

train_part = train_df.loc[~train_df['Year'].isin([2017,2018])]
test_part = train_df.loc[train_df['Year'].isin([2017,2018])]

print(train_part.bin_target.value_counts())
print(test_part.bin_target.value_counts())
tcv = TimeSeriesSplit(n_splits=10)
logit_pipe = Pipeline([('scaler', StandardScaler()), ('logit', LogisticRegression(class_weight='balanced', random_state=17))])
score = cross_val_score(logit_pipe, train_part[key_cols], train_part['bin_target'], cv=tcv, scoring = 'roc_auc')
print('Validation score:', score)
logit_pipe.fit(train_part[key_cols], train_part['bin_target'])

print(' Training data ')
y_pred = logit_pipe.predict_proba(train_part[key_cols])
class_names = ['No Flare', 'Flare']
print(pd.DataFrame(confusion_matrix(train_part['bin_target'],y_pred[:,1]>0.5), index=class_names, columns=class_names))

print('\n Test data ')
y_pred = logit_pipe.predict_proba(test_part[key_cols])
class_names = ['No Flare', 'Flare']
print(pd.DataFrame(confusion_matrix(test_part['bin_target'],y_pred[:,1]>0.5), index=class_names, columns=class_names))
```

```
 Training data
          No Flare  Flare
No Flare     47483   8688
Flare          446   2534

 Test data
          No Flare  Flare
No Flare      1927    311
Flare           10    103
```

Training data:

True positives: 2534

False Positives: 8688

True negatives: 47483

False negatives: 446

Test data:

True positives: 10

False Positives: 311

True negatives: 1927

False negatives: 103

Random Forest:

```
rf = RandomForestClassifier(n_estimators = 100, max_depth=3, class_weight='balanced')
score = cross_val_score(rf, train_part[key_cols], train_part['bin_target'], cv=tcv, scoring = 'roc_auc')
print('Validation score:', score)

rf = RandomForestClassifier(n_estimators = 500,  max_depth=3, class_weight='balanced')
rf.fit(train_part[key_cols], train_part['bin_target'])

print('Training data ')
rf_pred = rf.predict_proba(train_part[key_cols])
class_names = ['No Flare', 'Flare']
print(pd.DataFrame(confusion_matrix(train_part['bin_target'],rf_pred[:,1]>0.5), index=class_names, columns=class_names))

print('Test data ')
rf_pred = rf.predict_proba(test_part[key_cols])
class_names = ['No Flare', 'Flare']
print(pd.DataFrame(confusion_matrix(test_part['bin_target'],rf_pred[:,1]>0.5), index=class_names, columns=class_names))
```

```
Training data
          No Flare  Flare
No Flare     45136  11035
Flare          348   2632
Test data
          No Flare  Flare
No Flare      1810    428
Flare           10    103
```

Training data:

True positives: 2632

False Positives: 11035

True negatives: 45136

False negatives: 348

Test data:

True positives: 103

False Positives: 428

True negatives: 1810

False negatives: 10

Random Forest yield better results as based on confusion matrix and accuracy.

Conclusion:

Here we analyzed and visualized solar flares based on their magnetic properties by linking GOES and SDO databases. In future we can more analyze and predict whether these really harm the earth's atmosphere by taking time series data into consideration.

References:

1. https://en.wikipedia.org/wiki/List_of_GOES_satellites
2. http://jsoc.stanford.edu/doc/data/hmi/sharp/sharp.htm#data_segments
3. http://jsoc.stanford.edu/new/HMI/HARPS.html
4. https://sdo.gsfc.nasa.gov/
5. http://hmi.stanford.edu/magnetic/
6. http://jsoc.stanford.edu/jsocwiki/HARPDataSeries
7. https://en.wikipedia.org/wiki/Stereographic_projection
8. https://en.wikipedia.org/wiki/Solar_flare#Classification
9. https://docs.sunpy.org/projects/drms/en/latest/
10. http://docs.sunpy.org/en/latest/generated/gallery/index.html#acquiring-data