In [168]:

```
# default_exp ht1
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

ht1 - old hittest

> Test a more sensible approach to satellite visibility by using standard astronomy libraries to ask whether a satellite is in view of a ship at a given time.

In [169]:

```
# export
# Supposed to be "hide" but that keeps the module from having imports.

# Mostly test nbdev is set up right
# If you get ModuleNotFound, either symlink nbs/jacobs_vault -> jacobs_vault, or
# in each notebook `import sys; sys.path.append('..')`.
from jacobs_vault.template import *

from datetime import datetime
from dateutil import tz
from skyfield.api import EarthSatellite
from skyfield.api import Topos, load
import math
import pandas as pd
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import numpy as np
```

In [170]:

```
# hide
#from nbdev.showdoc import *
```

In [171]:

```
#export
COLUMNS = ["satellite", "day_dt", "day", "tle_dt", "tle_ts", "line1", "line2"]
# DTYPES = [str, str, int, str, int, str, str]
DTYPES = {'satellite': 'uint16', # observed values are ints in 5..41678, so 0..65535 is good
          'day_dt': 'str',       # here a single date, but generally datetime: PARSE
          'day': 'uint16',       # here a single value 6026, too big for uint8, but 16 is good
          'tle_dt': 'str',       # again, PARSE AS DATETIME
          'tle_ts': 'uint32',    # large ints, but < 4294967295. We could compress more, but... meh
          'line1': 'string',     # 12K unique 80-char TLE strings. Category wd give tiny compression.
          'line2': 'string'}     # In theory "string" is better than "object". Not seeing it here.

DATE_COLS = ['day_dt', 'tle_dt']
```

Load the day's TLE file

Create a function to load a single day's TLE file and return parsed datatypes.

In [172]:

```
#export
DAY_FILE_PATH="data/VAULT_Data/TLE_daily"  # Assumes symlink nbs/data -> actual data folder.

def load_day_file(_day:datetime, folder:str=DAY_FILE_PATH, date_cols=DATE_COLS, verbose=True):
    """Look for and load TLE datafile for {_day}."""
    df_path = "%s/%4d/%02d/%02d.tab.gz"%(folder, _day.year, _day.month, _day.day)
    if verbose:
        print(f'{_day}\t{df_path}')
    df = pd.read_csv(df_path,
                     names=COLUMNS, sep='\t', compression='gzip',
                     dtype=DTYPES,
                     parse_dates=date_cols,
                     infer_datetime_format=True)
    return df
```

Then test it on a single day.

In [173]:

```
df = load_day_file(datetime(2016, 6, 30))
df.count()
df.head()
```

```
2016-06-30 00:00:00     data/VAULT_Data/TLE_daily/2016/06/30.tab.gz
```

Out[173]:

	satellite	day_dt	day	tle_dt	tle_ts	line1	line2
0	1000	2016-06-30	6026	2016-06-27 11:15:21	1467040521	1 01000U 65008B 16179.46899882 .00000021 0...	2 01000 32.1467 333.7511 0009366 165.3909 194...
1	1000	2016-06-30	6026	2016-06-27 11:15:21	1467040521	1 01000U 65008B 16179.46899882 .00000021 0...	2 01000 32.1467 333.7511 0009366 165.3909 194...
2	1000	2016-06-30	6026	2016-06-27 11:15:21	1467040521	1 01000U 65008B 16179.46899882 .00000021 0...	2 01000 32.1467 333.7511 0009366 165.3909 194...
3	10000	2016-06-30	6026	2016-06-30 10:49:53	1467298193	1 10000U 77034A 16182.45131225 -.00000171 0...	2 10000 15.5820 331.7785 0019081 259.0540 28...
4	10002	2016-06-30	6026	2016-06-28 23:10:32	1467169832	1 10002U 77034C 16180.96565494 -.00000126 0...	2 10002 16.1681 333.0471 0296361 5.9346 0...

In [174]:

```
df.satellite.value_counts()
```

Out[174]:

```
29201    106
39694    101
33472     94
28584     87
29203     81
        ...
333        1
18764      1
29003      1
34004      1
16384      1
Name: satellite, Length: 12152, dtype: int64
```

Drop Dupes

Wait, each satellite should only need one TLE entry. These multiples are all *duplicates*. Watch.

In [175]:

```
df = df.drop_duplicates()
df.shape
```

Out[175]:

```
(12152, 7)
```

Memory check

Inspect the resulting dtypes and memory usage.

The parsing was successful. As expected, `line1` and `line2` are large. Using `category` doesn't save much because so many rows are unique. The `datetime` categories are surprisingly large.

In [176]:

```
pd.DataFrame([df.dtypes, df.memory_usage(index=False, deep=True)], index=['Dtype', 'Mem']).T
```

Out[176]:

	Dtype	Mem
satellite	uint16	24304
day_dt	datetime64[ns]	97216
day	uint16	24304
tle_dt	datetime64[ns]	97216
tle_ts	uint32	48608
line1	string	1531152
line2	string	1531152

In [177]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12152 entries, 0 to 15178
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype         
---  ------     --------------  -----         
 0   satellite  12152 non-null  uint16        
 1   day_dt     12152 non-null  datetime64[ns]
 2   day        12152 non-null  uint16        
 3   tle_dt     12152 non-null  datetime64[ns]
 4   tle_ts     12152 non-null  uint32        
 5   line1      12152 non-null  string        
 6   line2      12152 non-null  string        
dtypes: datetime64[ns](2), string(2), uint16(2), uint32(1)
memory usage: 569.6 KB
```

In [178]:

```
print(df.iloc[3]["line1"])
print(df.iloc[3]["line2"])
```

```
1 01001U 65008A   16178.87975142  .00000008  00000-0  00000+0 0  2425
2 01001  32.1427 144.4001 0016649 349.5981  10.4171  9.90724672860590
```

Skyfield

First, test the basic operation works as expected.

In [179]:

```
#export
def test_skyfield():
    lat =  45.0
    lon = -176.0
    earth_position = Topos(lat, lon)

    ts = load.timescale()
    t = ts.utc(datetime(2016, 6, 30).replace(tzinfo=tz.tzutc()))

    line1="1 10000U 77034A   16182.45131225 -.00000171  00000-0  00000+0 0  1275"
    line2="2 10000  15.5820 331.7785 0019081 259.0540  28.2803  0.96674507130362"
    satellite = EarthSatellite(line1, line2, '77034', ts)

    difference = satellite - earth_position

    topocentric = difference.at(t)
    alt, az, distance = topocentric.altaz()

    print(f'{alt.degrees:.1f}º, {az.degrees:.1f}º, {distance.km:.1f}km')
#
test_skyfield()
```

```
51.6º, 179.2º, 38068.6km
```

In [180]:

```
# https://rhodesmill.org/skyfield/earth-satellites.html
```

In [181]:

```
assert (datetime(1971, 6, 1) - datetime(1970, 6, 1)).days == 365
```

In [182]:

```
pd.DataFrame([51.5, 189, 2.3], index=['alt','az','days'])
```

Out[182]:

	0
alt	51.5
az	189.0
days	2.3

Can they see me?

Given at Lat/Lon/Time, what satellites can see me?

We pre-partition the TLE data by Year/Month/Day, so we can quickly load only *today*'s TLE data, and check whether Lat/Lon can see it.

First step: get Alt/Az/dt for each row.

Here we `apply` the `Skyfield.EarthSatellite` function to all TLE rows in the dataframe for today.

Benchmark: this takes about 6s on a laptop. @TODO: speed this up by 10x.

Minor glitch: cannot use faster `raw=True`

In theory `apply(..., raw=True)` should be faster than default `apply` . However, it's not working due to:

> AssertionError: Number of manager items must equal union of block items
>
> ```
>              # manager items: 7, # tot_items:
> ```

- Possible solution: `https://www.nuomiphp.com/eplan/en/254300.html`
- On the other hand, it's an open pandas ticket: `https://github.com/pandas-dev/pandas/issues/34822`

So I `try` the default way first. But the `except` won't work until we track down the block manager issue.

In [183]:

```
#export
def satellite_alt_az_days(_t0: datetime, lat: float, lon: float):
    '''Load tracks for day {_t0} and return altitiude, azimuth, and Δt
[days] for each row.

    '''
    earth_position = Topos(lat, lon)

    ts = load.timescale()
    t = ts.utc(_t0.replace(tzinfo=tz.tzutc()))

    def eval_tle(row):
        '''Extract satellite info from line1/line2/tle_dt.

        Returns alt, az, and (days between dt and each row).
        Inherits {ts}, {t}, and {earth_position} values at function de
finition.

        TODO: Currently only works for `apply(raw=False)`.

        '''
        try:
            satellite = EarthSatellite(row['line1'], row['line2'], 'x'
, ts)
            Δt = abs(_t0 - row['tle_dt'])
        except IndexError:
            # `apply(raw=True)` sends arrays instead of Series
            satellite = EarthSatellite(row[5], row[6], 'x', ts)
            Δt = abs(_t0 - row[3])
        topocentric = (satellite - earth_position).at(t)
        alt, az, distance = topocentric.altaz()
        return pd.Series([alt.degrees, az.degrees, Δt])

    df = load_day_file(_t0).drop_duplicates()
    df_alt_az_days = pd.DataFrame(df.apply(eval_tle, axis=1, raw=False
))
    df_alt_az_days.columns = ["altitude", "azimuth", "days"]
    #df_alt_az_days.reindex()
    return df_alt_az_days
```

Execute for a given day

2016-06-30 for starters. No, wait, do 2017-01 so we can match AIS tracks.

In [184]:

```
df_alt_az_days = satellite_alt_az_days(datetime(2017, 1, 15), 45.0, -1
76.0)
```

```
2017-01-15 00:00:00     data/VAULT_Data/TLE_daily/2017/01/15.tab.gz
```

In [185]:

```
df_alt_az_days.count()
```

Out[185]:

```
altitude    12880
azimuth     12880
days        12880
dtype: int64
```

In [186]:

```
df_alt_az_days.head()
```

Out[186]:

	altitude	azimuth	days
0	-44.702019	269.875412	6 days 06:10:45
7	-37.699756	275.709592	2 days 13:41:57
10	-67.166940	264.336240	0 days 20:11:41
11	-61.699411	214.660075	0 days 03:15:38
12	-16.891041	341.181383	0 days 21:23:03

Second step: Calculate the hit quality

First approximation:

- It's a **hit** if the alt > 0 (above the horizon).
- Smaller time difference -> better quality.

TODO: Kevin, did I capture that logic correctly? I'm confused how a 2-day lag can be "excellent". These aren't *days* are they?

In [187]:

```
#export
# Define cutoffs for TLE track quality
EXCELLENT, GOOD, POOR = 2, 14, 56
# Define horizon in alt degrees. Default 0. Consider 14.
HORIZON = 0

def hit_quality(df_alt_az_days):
    """Return hit/miss and quality as time proximity.

    Parameters
    ----------
    `df_alt_az_days`: Dataframe returned by `satellite_alt_az_days`.

    Returns
    --------
    Dataframe with columns ["hit", "miss"]. Each row will have exactly one filled, with
    a string denoting how recent the pass was, e.g. "excellent", "good", "poor", "stale".
    All "stale" are regarded as "miss".

    """

    def eval_quality(row):
        """Inner function to be `apply`d to a dataframe."""
        ser = None
        days = row[2].days
        altitude = row[0]
        if days <= EXCELLENT:
            if altitude > HORIZON:
                vals = ["excellent", math.nan]
            else:
                vals = [math.nan, "excellent"]
        elif days <= GOOD:
            if altitude > HORIZON:
                vals = ["good", math.nan]
            else:
                vals = [math.nan, "good"]
        elif days <= POOR:
            if altitude > HORIZON:
                vals = ["poor", math.nan]
            else:
                vals = [math.nan, "poor"]
        else:
            vals = [math.nan, "stale"]

        return pd.Series(vals)

    df_hit_quality = pd.DataFrame(df_alt_az_days.apply(eval_quality, a
xis=1))
    df_hit_quality.columns = ["hit", "miss"]
    return df_hit_quality
#
```

In [188]:

```
df_hit_quality = hit_quality(df_alt_az_days)
```

In [189]:

```
df_hit_quality["hit"].value_counts()
```

Out[189]:

```
excellent    1599
good           41
poor            1
Name: hit, dtype: int64
```

In [190]:

```
df_hit_quality["miss"].value_counts()
```

Out[190]:

```
excellent    10963
good           259
poor            11
stale            6
Name: miss, dtype: int64
```

In [191]:

```
#slow
pd.concat([df_hit_quality["hit"].value_counts(), df_hit_quality["miss"
].value_counts()], axis=1, sort=False)
```

Out[191]:

	hit	miss
excellent	1599.0	10963
good	41.0	259
poor	1.0	11
stale	NaN	6

In [192]:

```
df_alt_az_days_visible = df_alt_az_days[df_alt_az_days["altitude"]>HOR
IZON].copy()
```

In [193]:

```
df_alt_az_days_visible.count()
```

Out[193]:

```
altitude    1643
azimuth     1643
days        1643
dtype: int64
```

In [194]:

```
hit_quality(df_alt_az_days_visible)["hit"].value_counts()
```

Out[194]:

```
excellent    1599
good           41
poor            1
Name: hit, dtype: int64
```

In [195]:

```
df_alt_az_days_visible.head(5)
```

Out[195]:

	altitude	azimuth	days
14	4.663401	343.572725	0 days 09:55:28
15	28.755671	188.447474	0 days 14:55:48
18	0.330586	45.753727	0 days 02:37:02
20	6.996020	234.049748	0 days 03:22:40
29	28.677121	188.744730	0 days 08:22:57

Visualize the results

Generate a polar alt/az plot of the qualifying satellites

- Excellent = blue
- Good = red
- Else = yellow

Note the band of satellites at southern bearings -- this ship was in the Northern hemisphere.

In [196]:

```python
#export
def viz(df, show=True, size0=1, alpha=1, mode='svg'):
    """Polar plots a `df_alt_az_days_visible` dataframe.
    Dataframe must have: `color`, `days`, `altitude`, `azimuth`.
    Returns a Plotly Express polar plot figure with:
        * Excellent in blue
        * Good in pink
        * Poor and stale in yellow

    show: if True, also display the figure here
    size0: smallest marker size (used for best hits), out of 10.
    alpha: reduce if the figure is too cluttered
    mode: 'svg' is sharpest, 'webgl' is fastest

    """
    df["color"] = 2 # covers poor and stale
    df.loc[(df["days"].dt.days <= GOOD), "color"] = 1
    df.loc[(df["days"].dt.days <= EXCELLENT), "color"] = 0
    df["size"] = size0 + df["color"]*2
    df["R"] = 90.0 - df["altitude"]
    #fig = px.scatter_polar(df_alt_az_days_visible, r="R", theta="azimuth", color_discrete_sequence=['black'])
    fig = px.scatter_polar(df, r="R", theta="azimuth", color="color",
                           size="size", size_max=10, render_mode=mode)
    if show:
        fig.update_traces(opacity=alpha, showlegend=False).show()
    return fig
```

In [197]:

```
fig = viz(df_alt_az_days_visible)
```

In [198]:

```
fig.write_image(file='images/starmap2.pdf')
!open starmap.pdf
```

Recreate the original 2016-06-30 figure.

In [199]:

```
df_2016 = satellite_alt_az_days(datetime(2016, 6, 30), 45.0, -176.0)
fig1 = viz(df_2016[df_2016["altitude"]>HORIZON].copy())
fig1.write_image(file='images/starmap1.pdf')
```

```
2016-06-30 00:00:00     data/VAULT_Data/TLE_daily/2016/06/30.tab.gz
```

In [200]:

```
df_alt_az_days_visible['intdays'] = [x.days for x in df_alt_az_days_vi
sible.days]
```

In [201]:

```
df_alt_az_days_visible.intdays.value_counts()
```

Out[201]:

```
0      1359
1       183
2        57
3        18
5         8
4         6
6         4
7         3
8         2
181       1
71        1
20        1
Name: intdays, dtype: int64
```

David says try these from 2017:

- 4-jan-2017, 8pm
- 6 jan 3am
- 8 jan 6pm
- 12 jan 5am
- 19 jan 9am
- 26 jan 6pm
- 29 jan 6pm

In [202]:

```
# hide
# slow

dates = [(2017, 1, 4, 20), (2017, 1, 6, 3), (2017, 1, 8, 18), (2017, 1
, 12, 5),
         (2017, 1, 19, 9), (2017, 1, 26, 18), (2017, 1, 29, 18)]
dates = [datetime(*x) for x in dates]
lat, lon = 45.0, -176.0
N_dates = len(dates)
N_cols = 2
N_rows = 1 + N_dates//N_cols



# Tried a make_subplots but it didn't work.
# bigfig = make_subplots(rows=N_rows, cols=N_cols,
#                        specs=[[{'type':'polar'}]*N_cols]*N_rows)

figs = []
for i, _then in enumerate(dates):
    print(f'Making fig {i}.')
    df_alt_az_days = satellite_alt_az_days(_then, lat, lon)
    df_hit_quality = hit_quality(df_alt_az_days)
    hitmiss = pd.concat([df_hit_quality["hit"].value_counts(),
                         df_hit_quality["miss"].value_counts()], axis=
1, sort=False)
    df_alt_az_days_visible = df_alt_az_days[df_alt_az_days["altitude"]
>0].copy()
    fig = viz(df_alt_az_days_visible, show=False)
    figs.append(fig)
    #bigfig.add_trace(fig, row=1+i//N_rows, col=i%N_cols)
    #fig.show()
```

```
Making fig 0.
2017-01-04 20:00:00     data/VAULT_Data/TLE_daily/2017/01/04.tab.gz
Making fig 1.
2017-01-06 03:00:00     data/VAULT_Data/TLE_daily/2017/01/06.tab.gz
Making fig 2.
2017-01-08 18:00:00     data/VAULT_Data/TLE_daily/2017/01/08.tab.gz
Making fig 3.
2017-01-12 05:00:00     data/VAULT_Data/TLE_daily/2017/01/12.tab.gz
Making fig 4.
2017-01-19 09:00:00     data/VAULT_Data/TLE_daily/2017/01/19.tab.gz
Making fig 5.
2017-01-26 18:00:00     data/VAULT_Data/TLE_daily/2017/01/26.tab.gz
Making fig 6.
2017-01-29 18:00:00     data/VAULT_Data/TLE_daily/2017/01/29.tab.gz
```

In [204]:

```
# hide
# slow
for i, fig in enumerate(figs):
    fig.update_traces(opacity=.5) \
    .update_layout(height=400, width=400, title=dates[i].__str__()) \
    .show()
    fig.write_image(f'images/starmap_{dates[i]}.pdf')
```

```
In [ ]:
```