main

September 14, 2022

1 Ozone Layer Over Time

Based on data from https://ozonewatch.gsfc.nasa.gov/ https://ozonewatch.gsfc.nasa.gov/data/omi/

I was curious about the ozone layer and the hole in the ozone layer I've heard mentioned throughout my life and decided to try taking a quick look at the data. This took longer then I'd hoped to put together, but was a fairly rewarding project to parse one source of ozone layer data.

[1]:
```
# set up dependencies
%matplotlib inline
import numpy as np  # useful for many scientific computing in Python
import pandas as pd # primary data structure library
import folium # !conda install -c conda-forge folium=0.5.0 --yes
import os
import math
from branca.element import Figure
```

[2]:
```
# download the data:
# this section will
#  1. go to a page listing all the years of data available
#  2. get the links to each year's directory
#  3. check each year for data files to download, and make sure not to download␣
 ↪files that are already downloaded
#  4. download the data files to the local data directory
#
# Node: this involves downloading several thousand files representing literal␣
 ↪years worth of climate data, it will take a while.
# Prepare for it to take 2 or more hours.
from bs4 import BeautifulSoup
import requests

text = requests.get("https://ozonewatch.gsfc.nasa.gov/data/omi/").text
soup = BeautifulSoup(text, features="html.parser")
links = soup.find_all("a")
for l in links:
    if l['href'][0] == 'Y': # <a href="Y2004/">Y2004/</a>
        text = requests.get("https://ozonewatch.gsfc.nasa.gov/data/omi/"+␣
 ↪l['href']).text
```

```
        soup = BeautifulSoup(text, features="html.parser")
        sub_links = soup.find_all("a")
        for s in sub_links:
            if s['href'][0] == 'L' and not os.path.exists('./data/{0}'.
 ↪format(s['href'])): # <a href="L3_ozone_omi_20220101.txt">
                text_data_file = requests.get("https://ozonewatch.gsfc.nasa.gov/
 ↪data/omi/{0}{1}".format(l['href'], s['href']))
                open('./data/{0}'.format(s['href']), 'wb').write(text_data_file.
 ↪content)
```

[3]:
```
# load the data from the files - loading all the data at once can cause a crash
data = {}
file_list = os.listdir('./data/')

for f in file_list:
    if 'L3_ozone_omi_2004' in f: # get records only for the year 2004
    # if f[0] == 'L': # get all records - takes a long time and may crash
        with open('./data/{0}'.format(f)) as data_file:
            data[f] = data_file.read()
```

[4]:
```
# parse the files into points

def parse_data(data_str):
    # start by parsing the data string's header
    first  = data_str.split("\n")[0]  # " Day: 275 Oct  1, 2004    OMI TO3    ␣
 ↪STD OZONE    GEN:12:096 Asc LECT: 01:49 pm "
    second = data_str.split("\n")[1]  # " Longitudes:  360 bins centered on 179.
 ↪5  W  to 179.5  E   (1.00 degree steps)  "
    third  = data_str.split("\n")[2]  # " Latitudes :  180 bins centered on  89.
 ↪5  S  to  89.5  N   (1.00 degree steps)  "

    day = first[10:22]
    long_start = float(second[35:40])
    long_start = -long_start if second[42:43] == 'W' else long_start
    long_stop  = float(second[48:53])
    long_stop  = -long_stop if second[55:56] == 'W' else long_stop
    long_step  = float(second[60:64])
    long_bins  = int(second[14:17])

    lat_start = float(third[35:40])
    lat_start = -lat_start if third[42:43] == 'S' else lat_start
    lat_stop  = float(third[48:53])
    lat_stop  = -lat_stop if third[55:56] == 'S' else lat_stop
    lat_step  = float(third[60:64])
    lat_bins  = int(third[14:17])
```

```
    # next, get all the points into an array
    # lines are 76 characters long, start with a space, and have up to 25 bins
    # bins have 3 characters representing 1 int
    # lat comes after the last bin, with 3 characters of spaces
    # the actual lat value starts 9 characters after the last bin and is 6␣
↪characters long
    # "   lat =  -89.5"

    points = []
    block_size = math.ceil( (long_bins * 3) / 75 )
    lat_range  = np.arange(lat_start, lat_stop+lat_step, lat_step)
    long_range = np.arange(long_start, long_stop+long_step, long_step)
    max_score = 0
    min_score = 0
    max_point = {}

    for lat_pos in range(0, lat_bins):
        block = data_str.split("\n")[3 + (lat_pos * block_size):3 +␣
↪((lat_pos+1) * block_size)]
        lat = lat_range[lat_pos]
        for long_pos in range(0, long_bins):
            long = long_range[long_pos]
            line = block[math.floor(long_pos/25)]
            point = {
                'Latitude':  lat,
                'Longitude': long,
                'score': int(line[1+(long_pos % 25)*3:1+((long_pos % 25)+1)*3]),
                'color': '#ffffff'
            }
            if point['score'] > max_score:
                max_score = point['score']
                max_point = point
            if point['score'] < min_score:
                min_score = point['score']

            points.append(point)

    return {
        "points": points,
        "max": max_score,
        "min": min_score,
        "day": day,

        "long_start": long_start,
        "long_stop":  long_stop,
        "long_step":  long_step,
        "long_bins":  long_bins,
```

```
        "lat_start":  lat_start,
        "lat_stop":   lat_stop,
        "lat_step":   lat_step,
        "lat_bins":   lat_bins,
    }

points = {}
for k in data:
    points[k[13:21]] = parse_data(data[k])
```

[5]:
```
## color the points
# 1. red (#FF0000) (score_a)
# 2. yellow (#FFFF00) (score_b)
# 3. green (#00FF00) (score_c)
# 4. turquoise (#00FFFF) (score_d)
# 5. blue (#0000FF) (score_e)
def score_to_color(score, score_a, score_b, score_c, score_d, score_e):
    color = ""

    if score <= 0: # score should never be 0 or negative, show error state
        color = "#000000"

    elif score <= score_a: # red
        color = "#FF0000"

    elif score < score_b: # red to yellow (#FF0000 to #FFFF00)
        a = 255 * ((score - score_a) / (score_b - score_a)) # value between 0␣
 ↪and 256
        b = math.floor(a % 16)
        a = math.floor(a / 16)
        color = "#FF" + (hex(a)[-1]) + (hex(b)[-1]) + "00"

    elif score < score_c: # yellow to green (#FFFF00 to #00FF00)
        a = 255 * (1 - (score - score_b) / (score_c - score_b)) # value between␣
 ↪256 and 0
        b = math.floor(a % 16)
        a = math.floor(a / 16)
        color = "#" + (hex(a)[-1]) + (hex(b)[-1]) + "FF00"

    elif score < score_d: # green to turquoise (#00FF00 to #00FFFF)
        a = 255 * ((score - score_c) / (score_d - score_c)) # value between 0␣
 ↪and 256
        b = math.floor(a % 16)
        a = math.floor(a / 16)
        color = "#00FF" + (hex(a)[-1]) + (hex(b)[-1])
```

```
    elif score < score_e: # turquoise to blue (#00FFFF to #0000FF)
        a = 255 * (1 - (score - score_d) / (score_e - score_d)) # value between␣
 ↪256 and 0
        b = math.floor(a % 16)
        a = math.floor(a / 16)
        color = "#00" + (hex(a)[-1]) + (hex(b)[-1]) + "FF"

    else: # blue for scores equal to or greater then max score
        color = "#0000FF"

    return color

# a safe level of ozone isn't something I've been able to determine exactly.␣
 ↪but the agreed upon normal in dobson units (DU) seems to be 300
# https://theozonehole.com/dobsonunit.htm
patch_holes = True

for k in points:
    p = points[k]
    for i in range(len(p['points'])):
        point = p['points'][i]
        score = point['score']

        if score == 0 and patch_holes:
            if i > 0 and i < len(p['points']) - 1:
                if p['points'][i-1] != 0 and p['points'][i+1] != 0:
                    score = (p['points'][i-1]['score'] +␣
 ↪p['points'][i+1]['score']) / 2

        point['color'] = score_to_color(score, 100, 225, 350, 476, 600)
        point['text'] = str(point['score'])
```

[6]:
```
# Generate a map of the ozone layer during a single day - takes about a minute

# define the world map
fig = Figure(width=500, height=500)
world_map = folium.Map(width=500, height=500, zoon_start=1) #location=[56.130,␣
 ↪-106.35], zoom_start=4) # centered around Canada with a low zoom level

# bounds parameter: bounds (list of points (latitude, longitude)) - Latitude␣
 ↪and Longitude of line (Northing, Easting)
#for p in points:
p = points['20041001']
for point in p['points']:
    upper_left  = (point['Latitude']+p['lat_step']/2,␣
 ↪point['Longitude']-p['long_step']/2)
```

```
    upper_right = (point['Latitude']+p['lat_step']/2,␣
↪point['Longitude']+p['long_step']/2)
    lower_right = (point['Latitude']-p['lat_step']/2,␣
↪point['Longitude']-p['long_step']/2)
    lower_left  = (point['Latitude']-p['lat_step']/2,␣
↪point['Longitude']+p['long_step']/2)
    folium.Rectangle(
        bounds=[upper_left, upper_right, lower_right, lower_left],
        fill=True,
        fill_color=point['color'],
        color=point['color'],
        opacity=0.1,
        popup=point['text'],
    ).add_to(world_map)

fig.add_child(world_map)
fig # display world map
```

[6]: <branca.element.Figure at 0x7f6af274dca0>

The above map shows a wide range of scores for ozone over the globe. It also shows a large hole in the data set over the north pole. It turns out the data set I used actually has quite a collection of holes that move in various predictable patterns. Some days have more data missing then others, 2019-09-09 being a particularly bad example. And some days are missing entirely. I tried finding out why this is, and found some mention of instrument issues possibly the orbital path of the satelite being part of it, but I'll admit I wasn' sure what to make of it. But most of the data is still there, and it's enough to learn a lot about the ozone layer.

From what I can tell, there are other publicly available measurements of the ozone layer that I could combine this with, but I've not looked into those yet.

There's also the question of what these values actually mean, which I'll try to explain. The values are in Dobson units (DU), a measure of the concentration of ozone in an area of the globe. Roughly speeking, 300 DU of atmospheric ozone is quivalent to a 3mm layer of pure gas. Which means if the ozone layer wasn't diluted and was still at standard tempurature and pressure, it'd surround the earth in a layer less than a centimeter thick.

So, what's a good and bad DU value? I didn't see an exact answer to that, but there are some clues. Year round polar values in the 1960s were 300-500 DU, so we probably don't want lower then 300. Well, it turns out to be about UV light and how much of it the ozone layer filters out. We care about UV light because it's a potentially harmful type of radiation. And that's further complicated by the ozone layer filtering out different types of UV light at different rates. Apparently the ozone layer absorbs over 99% percenovert of UV-C rays, about 90% of the UV-B rays, and about 50 percent of the UV-A rays. lowering the amount of ozone would lower how much UV is being absorbed.

To give an idea of how bad lower ozone levels and thus higher UV levels could be, even without factoring in the elevated risks of cancer, I've included an exerpt from "NASA - New Simulation Shows Consequences of a World Without Earth's Natural Sunscreen": *By 2040, global ozone con-*

centrations fall below 220 DU, the same levels that currently comprise the "hole" over Antarctica. (In 1974, globally averaged ozone was 315 DU.) The UV index in mid-latitude cities reaches 15 around noon on a clear summer day (a UV index of 10 is considered extreme today.), giving a perceptible sunburn in about 10 minutes. Over Antarctica, the ozone hole becomes a year-round fixture.

[7]:
```
# generate ozone statistics - this may take several minutes. save time by␣
↪specifying a particular year instead

def generate_stats(points):
    def blank_section():
        return {
            "min": 0,
            "max": 0,
            "mean": 0,
            "total": 0,
            "missing": 0,
            "sum": 0
        }

    def add_point(section, score):
        section['total'] += 1
        section['sum']   += score
        if score == 0:
            section['missing'] += 1
        elif section['min'] > score or section['min'] == 0:
            section['min'] = score
        elif section['max'] < score:
            section['max'] = score

    def get_mean(section):
        if (section['total'] - section['missing']) != 0:
            section['mean'] = (section['sum'] / (section['total'] -␣
↪section['missing']))

    stats = {
        "all": blank_section(),
        "polar_north": blank_section(),
        "equitorial_north": blank_section(),
        "equitorial_south": blank_section(),
        "polar_south": blank_section(),
        "east": blank_section(),
        "west": blank_section()
    }

    for point in points['points']:
        score = point['score']
```

```
        add_point(stats['all'], score)

        if point['Latitude'] > 45:
            add_point(stats['polar_north'], score)
        elif point['Latitude'] > 0:
            add_point(stats['equitorial_north'], score)
        elif point['Latitude'] > -45:
            add_point(stats['equitorial_south'], score)
        else:
            add_point(stats['polar_south'], score)

        if point['Longitude'] > 0:
            add_point(stats['east'], score)
        else:
            add_point(stats['west'], score)

    for s in stats:
        get_mean(stats[s])

    points['stats'] = stats
    return stats

stats = []
file_list = os.listdir('./data/')
file_list.sort()
for f in file_list:
    if f[0] == 'L': # get all records
    # if 'L3_ozone_omi_2004' in f: # get records only for the year 2004
        with open('./data/{0}'.format(f)) as data_file:
            s = generate_stats(parse_data(data_file.read()))
            s['date'] = f[13:21]
            stats.append(s)

stats[0]
```

[7]:
```
{'all': {'min': 138,
  'max': 432,
  'mean': 274.2993180470449,
  'total': 64800,
  'missing': 4092,
  'sum': 16652163},
 'polar_north': {'min': 226,
  'max': 384,
  'mean': 285.7871913580247,
  'total': 16200,
  'missing': 3240,
  'sum': 3703802},
```

```
 'equitorial_north': {'min': 242,
  'max': 344,
  'mean': 271.8202219329242,
  'total': 16200,
  'missing': 69,
  'sum': 4384732},
 'equitorial_south': {'min': 239,
  'max': 432,
  'mean': 286.8393753485778,
  'total': 16200,
  'missing': 63,
  'sum': 4628727},
 'polar_south': {'min': 138,
  'max': 425,
  'mean': 254.19263565891472,
  'total': 16200,
  'missing': 720,
  'sum': 3934902},
 'east': {'min': 138,
  'max': 425,
  'mean': 275.0824738871132,
  'total': 32400,
  'missing': 2051,
  'sum': 8348478},
 'west': {'min': 140,
  'max': 432,
  'mean': 273.5164201719424,
  'total': 32400,
  'missing': 2041,
  'sum': 8303685},
 'date': '20041001'}
```

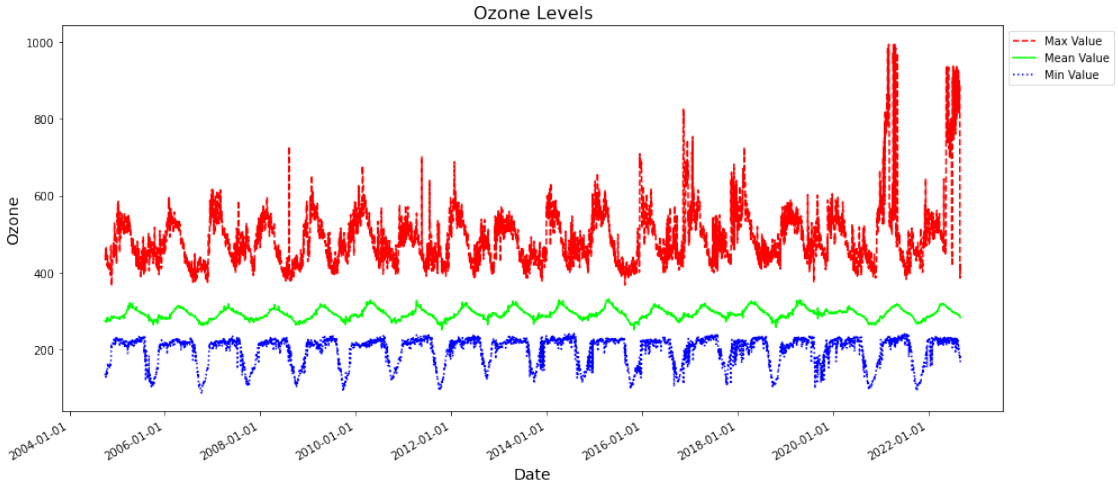
[8]:
```
# graph ozone statistics

import matplotlib.pyplot as plt
import matplotlib.dates as mdates

df = []
for s in stats:
    df.append({
        "Date": pd.Timestamp(s['date']),
        "Min Value (All)":  s['all']['min'],
        "Max Value (All)":  s['all']['max'],
        "Mean Value (All)": s['all']['mean'],
        "Min Value (East)":  s['east']['min'],
        "Max Value (East)":  s['east']['max'],
        "Mean Value (East)": s['east']['mean'],
```

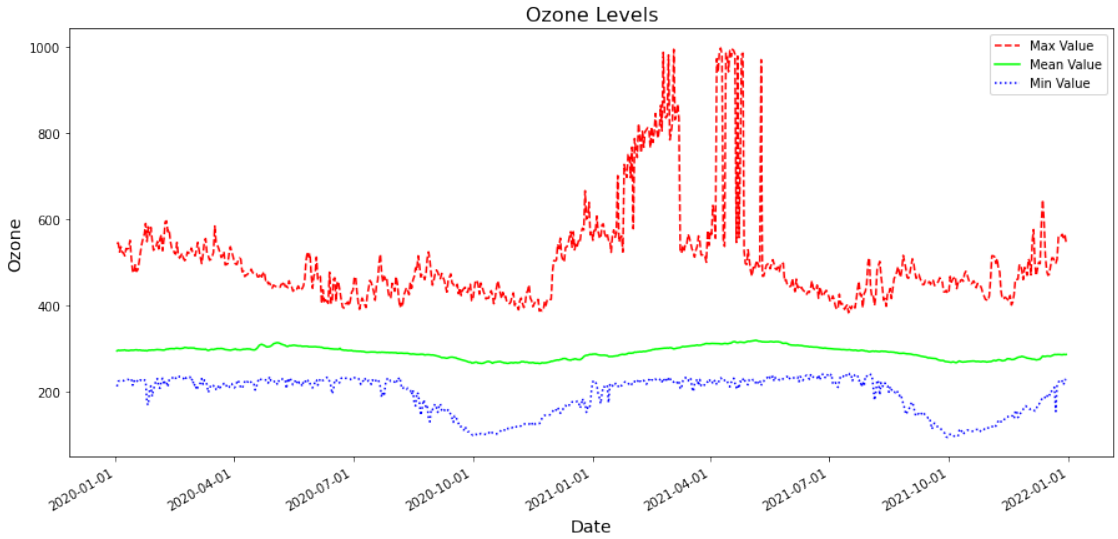
```
        "Min Value (West)":  s['west']['min'],
        "Max Value (West)":  s['west']['max'],
        "Mean Value (West)": s['west']['mean'],
        "Min Value (Polar North)":  s['polar_north']['min'],
        "Max Value (Polar North)":  s['polar_north']['max'],
        "Mean Value (Polar North)": s['polar_north']['mean'],
        "Min Value (Equitorial North)":  s['equitorial_north']['min'],
        "Max Value (Equitorial North)":  s['equitorial_north']['max'],
        "Mean Value (Equitorial North)": s['equitorial_north']['mean'],
        "Min Value (Equitorial South)":  s['equitorial_south']['min'],
        "Max Value (Equitorial South)":  s['equitorial_south']['max'],
        "Mean Value (Equitorial South)": s['equitorial_south']['mean'],
        "Min Value (Polar South)":  s['polar_south']['min'],
        "Max Value (Polar South)":  s['polar_south']['max'],
        "Mean Value (Polar South)": s['polar_south']['mean'],
    })
df = pd.DataFrame(df)
df = df.sort_values(by='Date')

fig, ax = plt.subplots(figsize=(15,7))
fig.autofmt_xdate()
xfmt = mdates.DateFormatter('%Y-%m-%d')
ax.xaxis.set_major_formatter(xfmt)
ax.plot(df['Date'], df['Max Value (All)'],  label="Max Value", ␣
 ↪linestyle="dashed", color='#FF0000')
ax.plot(df['Date'], df['Mean Value (All)'], label="Mean Value",␣
 ↪linestyle="solid",  color='#00FF00')
ax.plot(df['Date'], df['Min Value (All)'],  label="Min Value", ␣
 ↪linestyle="dotted", color='#0000FF')
plt.ylabel('Ozone', fontsize=14)
plt.xlabel('Date', fontsize=14)
ax.set_title("Ozone Levels", color='black', fontsize=16)
ax.legend(bbox_to_anchor=(1.0, 1.0))

df_a = df[(df['Date'] > '2020-01-01') & (df['Date'] < '2021-12-31')]
fig, ax = plt.subplots(figsize=(15,7))
fig.autofmt_xdate()
xfmt = mdates.DateFormatter('%Y-%m-%d')
ax.xaxis.set_major_formatter(xfmt)
ax.plot(df_a['Date'], df_a['Max Value (All)'],  label="Max Value", ␣
 ↪linestyle="dashed", color='#FF0000')
ax.plot(df_a['Date'], df_a['Mean Value (All)'], label="Mean Value",␣
 ↪linestyle="solid",  color='#00FF00')
ax.plot(df_a['Date'], df_a['Min Value (All)'],  label="Min Value", ␣
 ↪linestyle="dotted", color='#0000FF')
plt.ylabel('Ozone', fontsize=14)
```

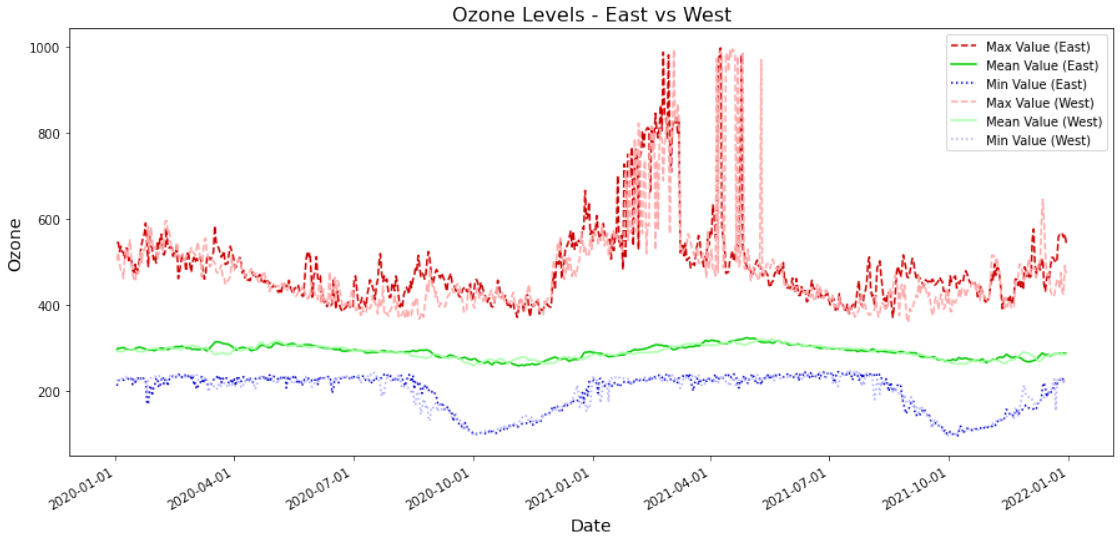
```
plt.xlabel('Date', fontsize=14)
ax.set_title("Ozone Levels", color='black', fontsize=16)
ax.legend(bbox_to_anchor=(1.0, 1.0))

fig, ex = plt.subplots(figsize=(15,7))
fig.autofmt_xdate()
xfmt = mdates.DateFormatter('%Y-%m-%d')
ex.xaxis.set_major_formatter(xfmt)
ex.plot(df_a['Date'], df_a['Max Value (East)'],  label="Max Value (East)", ␣
 ↪linestyle="dashed", color='#CC0000')
ex.plot(df_a['Date'], df_a['Mean Value (East)'], label="Mean Value (East)",␣
 ↪linestyle="solid",  color='#00CC00')
ex.plot(df_a['Date'], df_a['Min Value (East)'],  label="Min Value (East)", ␣
 ↪linestyle="dotted", color='#0000CC')
ex.plot(df_a['Date'], df_a['Max Value (West)'],  label="Max Value (West)", ␣
 ↪linestyle="dashed", color='#FFAAAA')
ex.plot(df_a['Date'], df_a['Mean Value (West)'], label="Mean Value (West)",␣
 ↪linestyle="solid",  color='#AAFFAA')
ex.plot(df_a['Date'], df_a['Min Value (West)'],  label="Min Value (West)", ␣
 ↪linestyle="dotted", color='#AAAAFF')
plt.ylabel('Ozone', fontsize=14)
plt.xlabel('Date', fontsize=14)
ex.set_title("Ozone Levels - East vs West", color='black', fontsize=16)
ex.legend(bbox_to_anchor=(1.0, 1.0))

fig, hx = plt.subplots(figsize=(15,7))
fig.autofmt_xdate()
xfmt = mdates.DateFormatter('%Y-%m-%d')
hx.xaxis.set_major_formatter(xfmt)
hx.plot(df_a['Date'], df_a['Max Value (Polar North)'],  label="Max Value (Polar␣
 ↪North)",  linestyle="dashed", color='#FF0000')
hx.plot(df_a['Date'], df_a['Mean Value (Polar North)'], label="Mean Value␣
 ↪(Polar North)", linestyle="solid",  color='#FF0000')
hx.plot(df_a['Date'], df_a['Min Value (Polar North)'],  label="Min Value (Polar␣
 ↪North)",  linestyle="dotted", color='#FF0000')
hx.plot(df_a['Date'], df_a['Max Value (Equitorial North)'],  label="Max Value␣
 ↪(Equitorial North)",  linestyle="dashed", color='#00FF00')
hx.plot(df_a['Date'], df_a['Mean Value (Equitorial North)'], label="Mean Value␣
 ↪(Equitorial North)", linestyle="solid",  color='#00FF00')
hx.plot(df_a['Date'], df_a['Min Value (Equitorial North)'],  label="Min Value␣
 ↪(Equitorial North)",  linestyle="dotted", color='#00FF00')
hx.plot(df_a['Date'], df_a['Max Value (Equitorial South)'],  label="Max Value␣
 ↪(Equitorial South)",  linestyle="dashed", color='#000000')
hx.plot(df_a['Date'], df_a['Mean Value (Equitorial South)'], label="Mean Value␣
 ↪(Equitorial South)", linestyle="solid",  color='#000000')
```

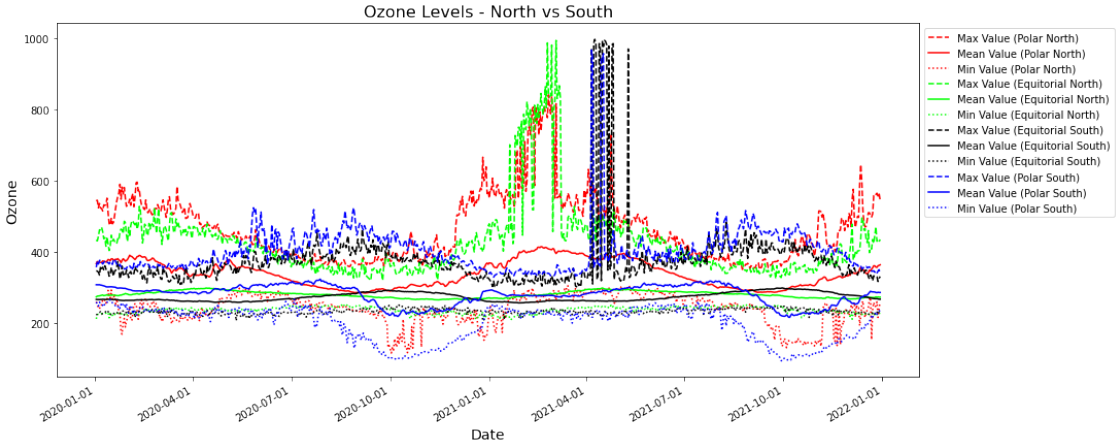
```
hx.plot(df_a['Date'], df_a['Min Value (Equitorial South)'],  label="Min Value␣
 ↪(Equitorial South)",  linestyle="dotted", color='#000000')
hx.plot(df_a['Date'], df_a['Max Value (Polar South)'],  label="Max Value (Polar␣
 ↪South)",  linestyle="dashed", color='#0000FF')
hx.plot(df_a['Date'], df_a['Mean Value (Polar South)'], label="Mean Value␣
 ↪(Polar South)", linestyle="solid",  color='#0000FF')
hx.plot(df_a['Date'], df_a['Min Value (Polar South)'],  label="Min Value (Polar␣
 ↪South)",  linestyle="dotted", color='#0000FF')
plt.ylabel('Ozone', fontsize=14)
plt.xlabel('Date', fontsize=14)
hx.set_title("Ozone Levels - North vs South", color='black', fontsize=16)
hx.legend(bbox_to_anchor=(1.0, 1.0))
```

[8]: <matplotlib.legend.Legend at 0x7f6a45d83160>









This is somewhat interesting, but could be expected from the first image.

A view of the full date range is too crowded to make out all the detail, but it's clear there are some yearly patterns at play. In Particular, the mean levels seem to be in a fairly stable and consistent yearly cycle at around 300.

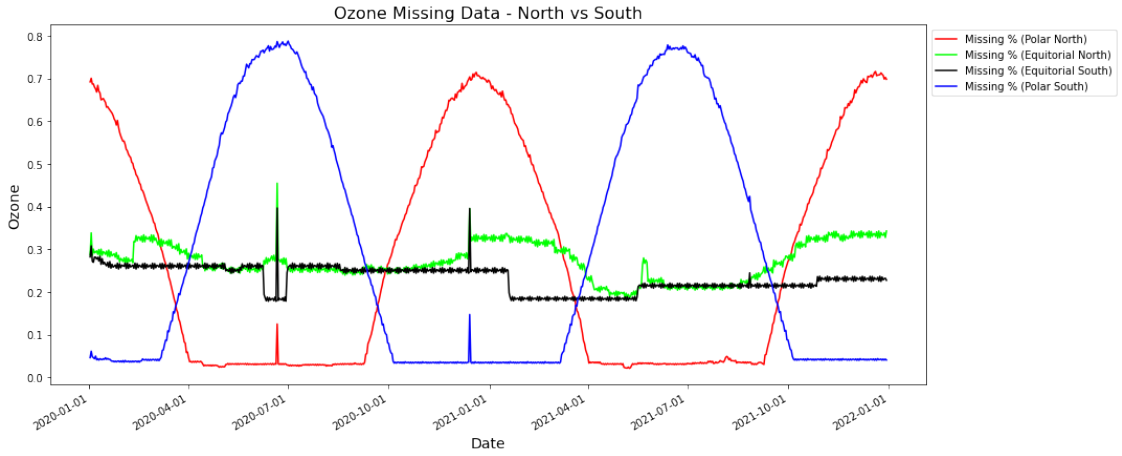
Ozone levels across east and west hemispheres are nearly identical and are close to the overall ozone levels.

Ozone levels in the south pole at their lowest are dipping to around 100, while averaging a bit above 200. Everything else averages between 200 and 400, and has max values somehow jumping to nearly 1000.

We'll have to take a look at how significant the cropping's effects are on the north vs south axis with some charts of just how much data is missing from each.

[9]:
```
df_m = []
for s in stats:
    df_m.append({
        "Date": pd.Timestamp(s['date']),
        "Missing % (All)":   s['all']['missing']/s['all']['total'],
        "Missing % (East)":  s['east']['missing']/s['east']['total'],
        "Missing % (West)":  s['west']['missing']/s['west']['total'],
        "Missing % (Polar North)":  s['polar_north']['missing']/
 ↪s['polar_north']['total'],
        "Missing % (Equitorial North)":  s['equitorial_north']['missing']/
 ↪s['equitorial_north']['total'],
        "Missing % (Equitorial South)":  s['equitorial_south']['missing']/
 ↪s['equitorial_south']['total'],
        "Missing % (Polar South)":  s['polar_south']['missing']/
 ↪s['polar_south']['total'],
    })
df_m = pd.DataFrame(df_m)
df_m = df_m.sort_values(by='Date')
df_m = df_m[(df_m['Date'] > '2020-01-01') & (df_m['Date'] < '2021-12-31')]

fig, hx = plt.subplots(figsize=(15,7))
fig.autofmt_xdate()
xfmt = mdates.DateFormatter('%Y-%m-%d')
hx.xaxis.set_major_formatter(xfmt)
hx.plot(df_m['Date'], df_m['Missing % (Polar North)'],       label="Missing %␣
 ↪(Polar North)",       linestyle="solid", color='#FF0000')
hx.plot(df_m['Date'], df_m['Missing % (Equitorial North)'],  label="Missing %␣
 ↪(Equitorial North)",  linestyle="solid", color='#00FF00')
hx.plot(df_m['Date'], df_m['Missing % (Equitorial South)'],  label="Missing %␣
 ↪(Equitorial South)",  linestyle="solid", color='#000000')
hx.plot(df_m['Date'], df_m['Missing % (Polar South)'],       label="Missing %␣
 ↪(Polar South)",       linestyle="solid", color='#0000FF')
plt.ylabel('Ozone', fontsize=14)
plt.xlabel('Date', fontsize=14)
hx.set_title("Ozone Missing Data - North vs South", color='black', fontsize=16)
hx.legend(bbox_to_anchor=(1.0, 1.0))
```

[9]:
```
<matplotlib.legend.Legend at 0x7f6a456fabe0>
```



More then 50% of of a region's data is missing at points. This seems to match some of the anomalies in the East vs West graph and North vs South graph.

1.1 Conclusion

The ozone layer's thickness and the lingering seasonal antartic hole in it are complex systems that we've been able to measure and map out. Over all, levels seem to be in stable seasonal cycles that vary mostly along north and south rather then east and west. This particular data set has some notable holes in it, but still provides a clear and useful picture of levels across the globe.

1.2 Further Work

I could try to clean up the data further and see just how well I could fill in the holes in the data. Given the cyclical nature of a lot of the data and that there seems to be a pattern in how levels flow across the globe it might be possible to make fairly accurate predictive models.

What would be easier though would be to first check the other ozone layer data sources to see how well their data matches up with this source and if they can fill in the data for us rather then trying to make guesses.

I could also try to detect how the ozone layer is changing outside of the seasonal cycles to try and gauge if it's healing or still diminishing. The average levels shown here suggest it's holding fairly consistent, but a closer look might show a gradual increase or decrease of a few points each year. I chose not to attempt this yet given the large regional holes in the data set, particularly in polar regions made it harder to accurately measure the true average.

[ ]: