In [1]:
```
import numpy as np
import pandas as pd
import xarray as xr
import matplotlib.pyplot as plt
import datetime
```

In [2]:
```
def p2h(p):
    '''
    to compute isobaric height
    p in hPa
    h in km
    '''
    return -8.08223*np.log(p/1013.25) #height in km

def h2p(h):
    '''
    to compute isobaric pressure
    h in km
    p in hPa
    '''
    return 1013.25*np.exp(-h/8.08223) #pressure in hPa

def vapor_pressure(p, q):
    return (p*q)/(0.622+0.378*q)

def sat_vapor_pressure(p, q, t):
    return 6.112*np.exp(17.67*(t-273.15)/(t-29.65))

def rel_hum(p, q, t):
    e = vapor_pressure(p, q)
    es= sat_vapor_pressure(p, q, t)
    return 100*(e/es)

def q2rh(q, p, t):
    divisor = np.exp(17.67*(t-273.15)/(t-29.65))
    return 0.263*p*q/divisor

def bias(x, x0):
    return np.nanmean(x-x0)

def rmse(x, x0):
    return np.sqrt(np.nanmean((x-x0)**2))

def p05(x):
    return np.percentile(x, 5)

def p25(x):
    return np.percentile(x, 25)

def p50(x):
    return np.percentile(x, 50)

def p75(x):
    return np.percentile(x, 75)

def p95(x):
    return np.percentile(x, 95)
```

```
def calc_theta(t, p):
    '''
    to compute potential temperature (theta in Kelvin)
    t in Kelvin
    p in hPa or mb
    '''
    theta = t*(1000/p)**0.286
    return theta

def calc_m(t, p):
    '''
    to compute the gradient of potential refractive index (m in /meter)
    t in Kelvin
    p in hPa or mb
    '''
    h = 1000*p2h(p)
    theta = calc_theta(t, p)
    grad = np.gradient(theta, h)
    m = -79e-6*p*grad/(t*t)
    return m

def calc_s(u, v, h):
    '''
    to compute the gradient of horizontal speed (s in m/s)
    u,v in m/s
    p in hPa
    '''
    f1 = np.gradient(u, h)
    f2 = np.gradient(v, h)
    s = np.hypot(f1, f2)
    return s

def calc_lpar(u, v, t, p):
    '''
    to compute outer scale for optical turbulence L0^(4/3) in m
    u,v in m/s
    t in Kelvin
    p in hPa
    '''
    h = 1000*p2h(p)
    a = np.where(p < 200)[0] #stratosphere

    s = calc_s(u, v, h)
    f1 = 1.64 + 42*s
    f2 = .506 + 50*s
    #dtdh = np.gradient(t, h)
    #f1 = 4.641589e-2 * 10**(0.362+16.728*s-192.347*dtdh)
    #f2 = 4.641589e-2 * 10**(0.757+13.819*s-57.784*dtdh)
    lpar = f1
    if len(a) > 0:
        lpar[a] = f2[a]

    return lpar

def is_ndarray(x):
    if type(x) == np.ndarray:
        return x
    elif type(x) == list:
        return np.array(x)
    else:
```

```
        return np.array([x])

def calc_cn2(u, v, t, p):
    '''
    to compute refractive index Cn^2
    u,v in m/s
    t in Kelvin
    p in hPa
    '''
    u = is_ndarray(u)
    v = is_ndarray(v)
    t = is_ndarray(t)
    p = is_ndarray(p)

    lpar = calc_lpar(u, v, t, p)
    m = calc_m(t, p)
    cn2 = 2.8*lpar*m*m
    return cn2

def calc_eps(cn2, p, wavelength=500, height=1300):
    '''
    to compute seeing
    '''
    wavelength *= 1e-9
    h = 1000*p2h(p)
    f1 = 5.25*wavelength**(-0.2)
    a = np.where(h >= height)[0]
    f2 = -np.trapz(cn2[a], h[a])
    eps = f1*f2**(0.6)
    return eps

def fried(cn2, p, wavelength=500, height=1300):
    wavelength *= 1e-9
    h = 1000*p2h(p)
    k = 2*np.pi/wavelength
    a = np.where(h >= height)[0]
    f2 = -np.trapz(cn2[a], h[a])
    r0 = (0.433*k*k*f2)**(-0.6)
    return r0

def richardson(u, v, t, p):
    '''
    to compute richardson number
    '''
    h = 1000*p2h(p)
    theta = calc_theta(t, p)
    grad_theta = np.gradient(theta, h)
    grad_s2 = (calc_s(u, v, h))**2
    ri = 9.8*grad_theta/grad_s2/theta
    return ri
```

Praproses data radiosonde Eltari

Pada bagian ini, semua data CSV yang ada di folder `eltari` dibaca dan disatukan. Ada beberapa poin yang perlu digarisbawahi dalam proses ini:

- Kolom `ObsTime` dinyatakan dalam UT. Karena kita hanya tertarik pada seeing malam hari, maka cukup pilih data dengan `ObsTime >= 12`. Batas ini bersesuaian dengan pukul 20 waktu Kupang.

- Bila jumlah data yang terlalu banyak (>500), kita bisa lakukan sampling secara acak dan cukup ambil 500 titik data.
- Standardisasi data:
  - ketinggian `h` dalam kilometer
  - temperatur `t` dalam Kelvin
  - kecepatan angin `speed` dalam m/s
  - komponen kecepatan `u` dan `v` dihitung beradasarkan wind direction `WD`
  - `time` dinyatakan dalam UT

File luaran disimpan di `../data/eltari_2017_2018_b.csv`. Pastikan ada folder `IGRA` di direktori yang sama dengan notebook ini.

In [ ]:
```
## skip this block
root = 'C:/Users/lm8g19/Research/Obnas/case/siteTesting/IGRA'
files = glob.glob(f'{root}/eltari/*12.CSV')

rec = pd.DataFrame()
skipped = []
for path in tqdm.tqdm(files):
    try:
        # pilih data malam hari waktu lokal
        df = pd.read_csv(path, skiprows=6)
        t = int(df[' ObsTime'].values[0][0:2])
        if t < 12:
            continue

        # membuang baris kosong
        df = df.dropna()

        # menghapus spasi pada nama kolom
        col = df.columns.values
        rename = {}
        for c in col:
            rename[c] = c.replace(' ','')
        df = df.rename(columns=rename)

        # sampling 500 data
        if len(df) > 500:
            df = df.sample(n=500).sort_values('Height')

        new_df = pd.DataFrame()

        # standardisasi data
        date = f'{path[-14:-10]}-{path[-10:-8]}-{path[-8:-6]} '
        height = df['Height'].values*0.001
        pressure = df['Press0'].values #h2p(height)

        new_df['p'] = pressure
        new_df['t'] = df['Temp0'].values.astype(float) + 273.15
        theta = np.radians(df['WD'].values)
        speed = df['WS'].values.astype(float)
        new_df['u'] = -speed * np.sin(theta)
        new_df['v'] = -speed * np.cos(theta)
        new_df['speed'] = speed
```

```
        new_df['rh'] = df['Humi0'].values
        new_df['time'] = date + '12:00:00+00' #[date+a for a in df[' ObsT
        new_df['h'] = height
        new_df['report_id'] = path[-14:-6]
        new_df = new_df.sort_values('h')

        rec = pd.concat([rec, new_df], ignore_index=False)
    except:
        skipped.append(date)

print('skipped:', len(date))
rec.to_csv('IGRA/eltari_2017_2018_b.csv', index=False)
```

In [ ]:
```
rec = pd.read_csv('../data/eltari_2017_2018_b.csv')
rec.head(2)
```

Perbandingan antara data radiosonde dan ERA5 di Eltari

Untuk melakukan komparasi, kita perlu kombinasikan dua dataset. Data ERA5 disimpan dalam bentuk NetCDF (`eltari_levels_tuv_2017.nc`) sementara data radiosonde disimpan dalam bentuk CSV (`eltari_2017_2018_b.csv`).

Ada beberapa poin penting yang perlu diperhatikan:

- Data ERA5 yang diunduh mencakup 2x2 grid lintang bujur. Nilai lintang pusat grid adalah $[-10.00, -10.25]$ sementara bujurnya adalah $[124.50, 125.75]$.
- Terdapat 37 pressure level pada ERA5. Interpolasi linier dapat dilakukan untuk memperoleh nilai `u` , `v` , `speed` dan `t` pada pressure level yang tercatat pada dataset radiosonde.

In [ ]:
```
# membaca data ERA5
# skip this block
era5 = xr.load_dataset('era5/eltari_levels_tuv_2017.nc')
era5
```

In [ ]:
```
# membaca data radiosonde
df = pd.read_csv('../data/eltari_2017_2018_b.csv')
ids = np.unique(df.report_id.values)
nskip = 0
stats = pd.DataFrame()
combi = pd.DataFrame()

for i in ids:
    if str(i)[0:4] != '2017':
        continue

    d = df[df.report_id == i]

    if d['p'].min() > 8500:
        nskip += 1
        continue

    time = np.datetime64(d['time'].values[0][0:10] + ' 12')
    levels = d['p'].values
```

```
    # pilih grid yang bersesuaian dengan lokasi eltari
    # lakukan interpolasi pada pressure level yang tercatat dalam data ra
    xr_sel = era5.sel(time=time).isel(longitude=1, latitude=1).interp(lev
    xr_sel['p'] = xr_sel.index.values
    xr_sel['speed'] = np.hypot(xr_sel['u'].values, xr_sel['v'].values)
    s = pd.DataFrame({'time':[d['time'].values[0][0:16]]})

    for j in ['t', 'u', 'v', 'speed']:
        xr_sel[j+'_igra'] = d[j].values
        a = bias(xr_sel[j].values, d[j].values)
        b = rmse(xr_sel[j].values, d[j].values)
        s['bias_'+j] = [a]
        s['rmse_'+j] = [b]

    combi = pd.concat([combi, xr_sel], ignore_index=True)
    stats = pd.concat([stats, s], ignore_index=True)

    #except:
    #    pass

combi['month'] = [a.month for a in combi.time]
print(nskip/len(ids))

# menyimpan hasil kombinasi
combi = combi.dropna()
combi.to_csv('../data/combi_eltari_2018.csv', index=False)
```

In [3]:
```
# mulai dari sini
combi = pd.read_csv('../data/combi_eltari_2018.csv')
combi.head(2)
```

Out[3]:

	longitude	latitude	time	t	u	v	p	speed	
0	123.75	-10.25	2018-01-01 12:00:00	299.536535	0.869067	-0.029372	991.1	0.869564	2
1	123.75	-10.25	2018-01-01 12:00:00	299.530122	0.869048	-0.029237	991.0	0.869540	2

In [4]:
```
# periksa nilai bias dan rmse secara keseluruhan
overall = pd.DataFrame()
for j in ['t', 'u', 'v', 'speed']:
    a = bias(combi[j].values, combi[j+'_igra'].values)
    b = rmse(combi[j].values, combi[j+'_igra'].values)
    overall['bias_'+j] = [a]
    overall['rmse_'+j] = [b]

overall.to_dict()
```

Out[4]:
```
{'bias_t': {0: -0.05359985711408688},
 'rmse_t': {0: 1.253198409823998},
 'bias_u': {0: 0.03055066050367309},
 'rmse_u': {0: 2.69050664409815},
 'bias_v': {0: -0.1672162006006819},
 'rmse_v': {0: 3.6197849185478628},
 'bias_speed': {0: -0.660146203732861},
 'rmse_speed': {0: 3.418497447359415}}
```

Visualisasi data ERA5 dan radiosonde

Setelah kedua dataset dipadukan, maka visualisasi data dapat dilakukan.

Kalau keseluruhan data untuk setiap pressure level dirata-ratakan, akan diperoleh satu profil temperatur maupun kecepatan angin sebagai fungsi ketinggian. Profil ini bisa kita bilang sebagai **aggregated profile**.

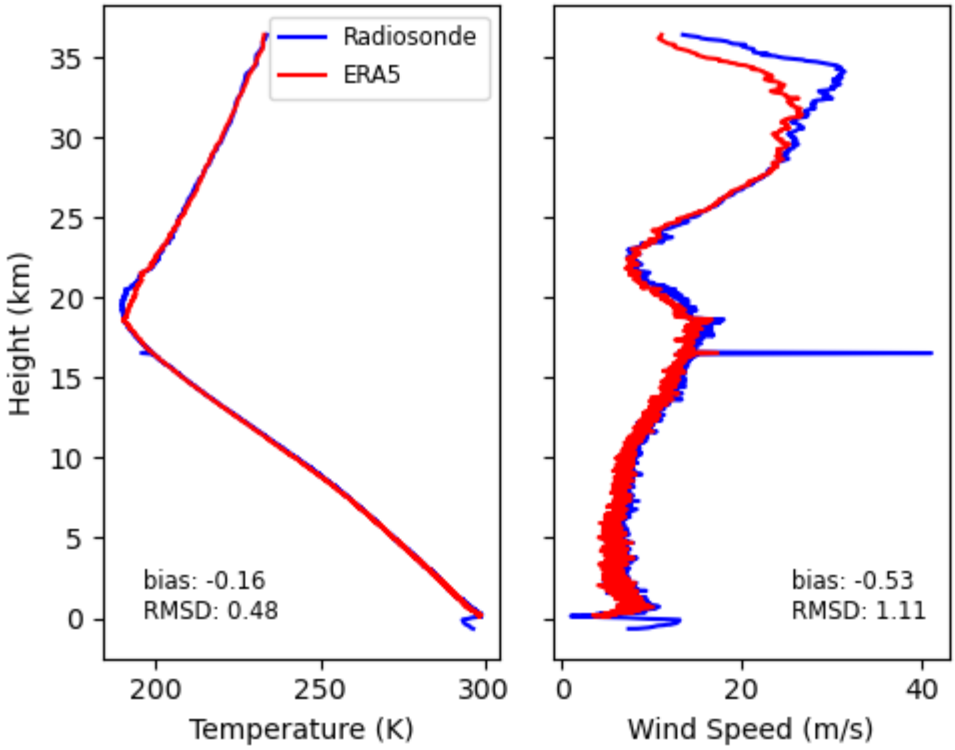
Profil yang diperoleh dari ERA5 dan radiosonde tentu tidak sama persis. Ada bias dan simpangan. Nilai bias dan simpangan yang dihitung berdasarkan **aggregated profile** berbeda dengan nilai bias dan simpangan yang dihitung sebelumnya (untuk keseluruhan data).

statistik	keseluruhan data	aggregated profile
bias(t)	-0.08	-0.16
bias(speed)	-0.73	-0.57
rmsd(t)	1.23	0.43
rmsd(speed)	3.24	0.87

In [5]:
```
# hitung nilai rerata untuk setiap pressure level
agg_all = combi[['p','t','speed','t_igra','speed_igra']].groupby('p').agg
agg_all['h'] = p2h(agg_all.index.values)
```

In [6]:
```
v0 = agg_all['speed_igra'].rolling(window=10).mean()
v1 = agg_all['speed'].rolling(window=10).mean()
t0 = agg_all['t_igra'].rolling(window=10).mean()
t1 = agg_all['t'].rolling(window=10).mean()
h = agg_all['h'].values

bt = bias(t1, t0)
bv = bias(v1, v0)
rt = rmse(t1, t0)
rv = rmse(v1, v0)

fig,ax = plt.subplots(1, 2, figsize=(5,4), dpi=100, sharey=True)

txt = f'bias: {bt:.2f}\nRMSD: {rt:.2f}'
ax[0].plot(t0, h, '-b', label='Radiosonde')
ax[0].plot(t1, h, '-r', label='ERA5')
ax[0].set_xlabel('Temperature (K)')
ax[0].set_ylabel('Height (km)')
ax[0].legend(fontsize='small', loc='upper right')
ax[0].text(0.1, 0.05, txt, ha='left', va='bottom', fontsize='small',
           bbox=dict(color='w', alpha=0.7), transform=ax[0].transAxes)

txt = f'bias: {bv:.2f}\nRMSD: {rv:.2f}'
ax[1].plot(v0, h, '-b', label='IGRA')
ax[1].plot(v1, h, '-r', label='ERA5')
ax[1].set_xlabel('Wind Speed (m/s)')
ax[1].text(0.6, 0.05, txt, ha='left', va='bottom', fontsize='small',
           bbox=dict(color='w', alpha=0.7), transform=ax[1].transAxes)

plt.tight_layout()
```

```
plt.savefig('plot/ERA5_IGRA_eltari.svg')
plt.savefig('plot/ERA5_IGRA_eltari.png')
plt.show()
```



In [7]:

```
## Evaluasi kecocokan ERA5 dan radiosonde tiap bulan

agreement = pd.DataFrame()
months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Ags','Sep','Oct','No

for i in range(0,12):
    d = combi[combi.month == i+1][['p','t','speed','t_igra','speed_igra']
    h = p2h(d.index.values)
    v0 = d['speed_igra'].rolling(window=10).mean()
    v1 = d['speed'].rolling(window=10).mean()
    t0 = d['t_igra'].rolling(window=10).mean()
    t1 = d['t'].rolling(window=30).mean()
    a = pd.DataFrame({
        'm': i+1,
        'month': months[i],
        'bias_t': bias(t1,t0),
        'bias_v': bias(v1,v0),
        'rmsd_t': rmse(t1,t0),
        'rmsd_v': rmse(v1,v0)
    }, index=[0])
    agreement = pd.concat([agreement, a], ignore_index=True)

agreement
```

Out[7]:

	m	month	bias_t	bias_v	rmsd_t	rmsd_v
0	1	Jan	-0.041582	-0.350624	0.771478	1.149444
1	2	Feb	-0.264677	-0.184036	0.754330	0.771248
2	3	Mar	-0.240891	-0.344306	0.732879	1.722539
3	4	Apr	-0.263715	-0.790619	0.927241	1.369748
4	5	May	-0.382082	-0.623850	0.903165	1.415331
5	6	Jun	-0.319783	-0.831585	0.947878	1.566993
6	7	Jul	-0.387962	-0.854355	0.831443	1.321722
7	8	Ags	-0.430661	-0.904481	0.802504	1.476604
8	9	Sep	-0.522592	-0.465386	0.964116	1.450679
9	10	Oct	-1.283092	-0.324187	1.850208	1.643159
10	11	Nov	-0.446210	-0.262107	0.702095	1.035496
11	12	Dec	-0.358851	0.255417	1.052604	1.548941

In [122…

```
df = pd.read_csv('../data/eltari_2017_2018_b.csv')
df = df.dropna(how='any').reset_index(drop=True)
ids = np.unique(df.report_id.values)
prf = pd.DataFrame()
seeing = pd.DataFrame()
window = 3

for i in ids:
    if str(i)[0:4] != '2017':
        continue

    d = df[df.report_id == i].copy().sort_values('p')
    p = d['p'].values
    a = np.append((p[1:] - p[:-1]) != 0, True)
    d = d[a]
    p = d['p'].rolling(window=window).mean().values[window:]
    u = d['u'].rolling(window=window).mean().values[window:]
    v = d['v'].rolling(window=window).mean().values[window:]
    t = d['t'].rolling(window=window).mean().values[window:]
    speed = d['speed'].rolling(window=window).mean().values[window:]
    cn2 = calc_cn2(u, v, t, p)
    ri = richardson(u, v, t, p)

    d1 = pd.DataFrame({
        'time': d['time'].values[window:],
        'u': u,
        'v': v,
        't': t,
        'p': p,
        'speed': speed,
        'cn2': cn2,
        'ri': ri
    })

    d2 = pd.DataFrame({
        'time': d['time'].values[0:1],
        'fried': [fried(cn2, p)],
```

```
        'eps': [206265*calc_eps(cn2, p)]
    })

    prf = pd.concat([prf,d1], ignore_index=True)
    seeing = pd.concat([seeing,d2], ignore_index=True)

# additional info
dtime = pd.DatetimeIndex(prf.time.values)
prf['year'] = dtime.year.values
prf['month'] = dtime.month.values
prf['hour'] = dtime.hour.values

dtime = pd.DatetimeIndex(seeing.time.values)
seeing['year'] = dtime.year.values
seeing['month'] = dtime.month.values
seeing['hour'] = dtime.hour.values
```

In [123…

```
#seeing = pd.read_csv('era5/seeing_2017_2018_eltari.csv')
sel = seeing.month.isin([4,5,6,7,8,9,10])
func = [p05,p25,p50,p75,p95]
agg3 = seeing[['month','fried','eps']].groupby(['month']).agg(func=func)
agg4 = seeing[['hour','fried','eps']].groupby(['hour']).agg(func=func)
agg5 = seeing[['hour','fried','eps']][sel].groupby(['hour']).agg(func=fun

seeing[sel][['fried','eps']].describe(percentiles=[.05,.25,.50,.75,.90,.9
```

Out[123…

	fried	eps
count	101.000000	101.000000
mean	0.080148	1.276784
std	0.012211	0.358073
min	0.023242	0.896565
5%	0.059586	1.038036
25%	0.074741	1.125706
50%	0.081568	1.208728
75%	0.087583	1.319133
90%	0.093261	1.484511
95%	0.094980	1.654623
max	0.109968	4.241942

In [124…

```
prf = prf.dropna()
seeing = seeing.dropna()
prf.to_csv('../data/eltari_turb_profile.csv', index=False)
seeing.to_csv('../data/eltari_seeing.csv', index=False)
```

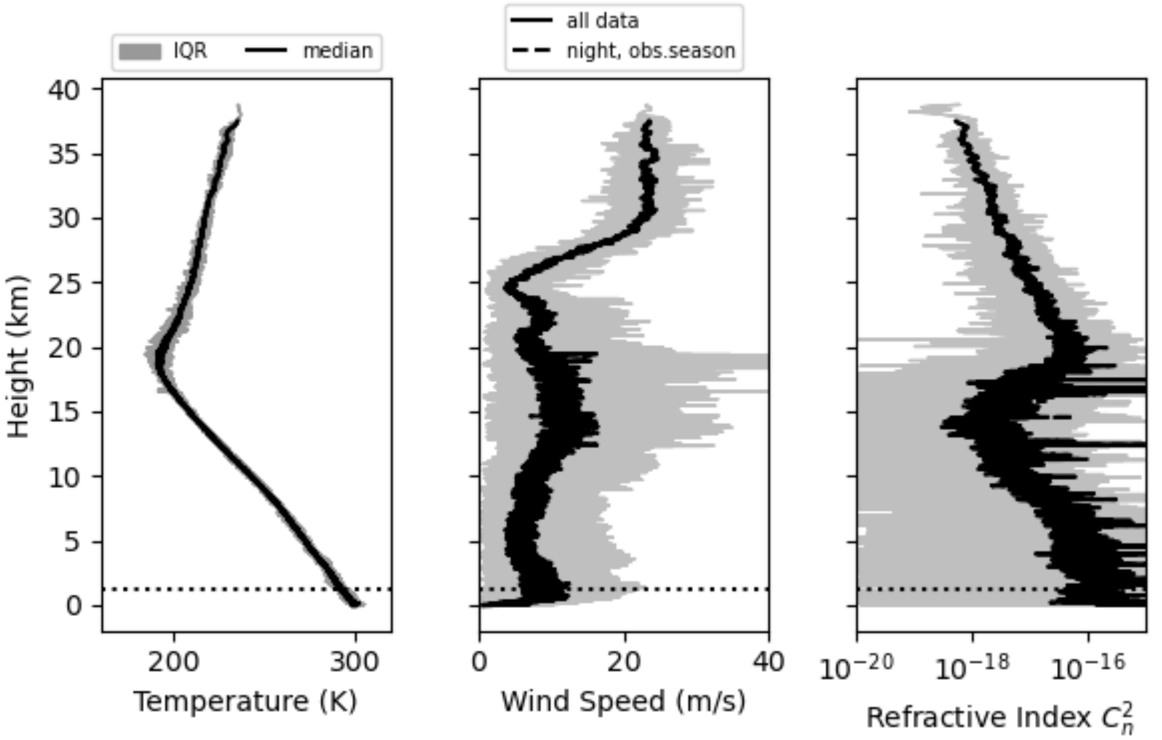
In [125…

```
sel = np.logical_and(prf.month.isin([4,5,6,7,8,9,10]), prf.hour.between(1
func = ['min','max',p25,p50,p75]
agg1 = prf[['p','u','t','speed','cn2','ri']].groupby('p').agg(func=func)
agg2 = prf[['p','u','t','speed','cn2','ri']][sel].groupby('p').agg(func=f
```

In [144…

```
fig,ax = plt.subplots(1, 3, figsize=(6,4), dpi=100, sharey=True)
ax = ax.flatten()
```

```
y = p2h(agg1.index.values)
y2 = p2h(agg2.index.values)

window=15
ax[0].fill_betweenx(y, agg1[('t','p25')], agg1[('t','p75')], color='0.6',
ax[0].plot(agg1[('t','p50')].rolling(window=window).mean(), y, '-k', labe
ax[0].plot(agg2[('t','p50')].rolling(window=window).mean(), y2, '--k')
ax[0].set_xlabel('Temperature (K)')
ax[0].set_ylabel('Height (km)')
ax[0].legend(fontsize='x-small', ncol=2, bbox_to_anchor=(0.5, 1.0), loc='
ax[0].set_xlim(160, 320)

ax[1].fill_betweenx(y, agg1[('speed','p25')], agg1[('speed','p75')], colo
ax[1].plot(agg1[('speed','p50')].rolling(window=window).mean(), y, '-k',
ax[1].plot(agg2[('speed','p50')].rolling(window=window).mean(), y2, '--k'
ax[1].set_xlabel('Wind Speed (m/s)')
ax[1].legend(fontsize='x-small', bbox_to_anchor=(0.5, 1.0), loc='lower ce
ax[1].set_xlim(0, 40)

ax[2].fill_betweenx(y, agg1[('cn2','p25')], agg1[('cn2','p75')], color='0
ax[2].plot(agg1[('cn2','p50')].rolling(window=window).mean(), y, '-k', la
ax[2].plot(agg2[('cn2','p50')].rolling(window=window).mean(), y2, '--k',
ax[2].set_xlabel('Refractive Index $C_n^2$')
ax[2].set_xscale('log')
ax[2].set_xlim(1e-20, 1e-15)
'''
ax[2].fill_betweenx(y, agg1[('ri','p25')], agg1[('ri','p75')], color='0.7
ax[2].plot(agg1[('ri','p50')], y, '-k', label='all data')
ax[2].plot(agg2[('ri','p50')], y, '--k', label='night, obs.season')
ax[2].set_xlabel('Richardson Number')
ax[2].set_xscale('log')
'''

for i in [0,1,2]:
    ax[i].axhline(y=1.3, linestyle='dotted', color='k')

plt.tight_layout()
plt.savefig('plot/seeing_data_eltari_era5.svg')
plt.show()
```



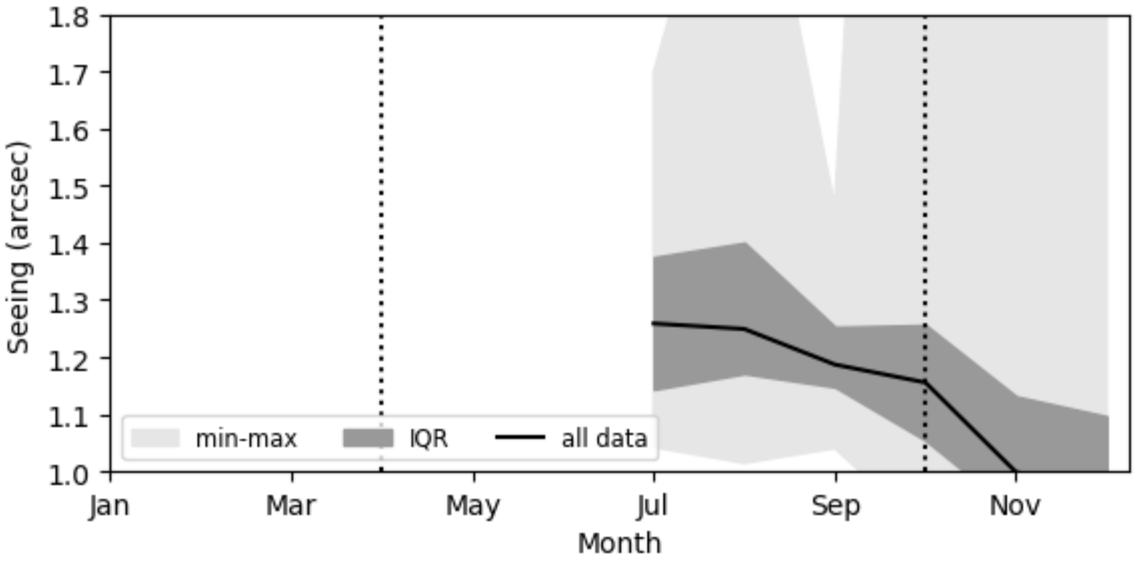
In [145…

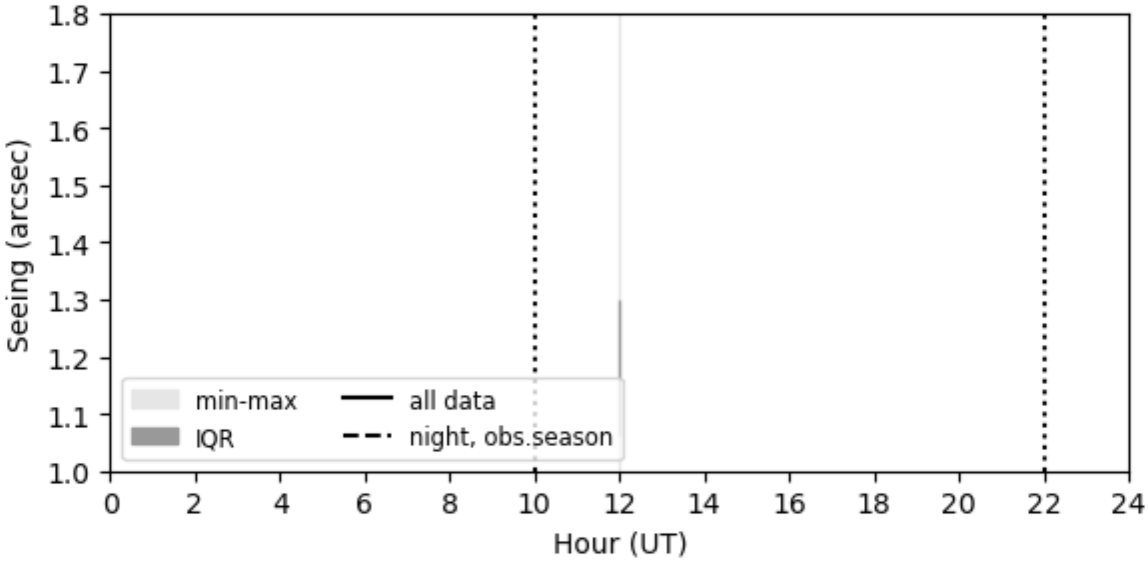
```
sel = seeing.month.isin([4,5,6,7,8,9,10])
func = ['min','max',p25,p50,p75]
agg3 = seeing[['month','fried','eps']].groupby(['month']).agg(func=func)
agg4 = seeing[['hour','fried','eps']].groupby(['hour']).agg(func=func)
agg5 = seeing[['hour','fried','eps']][sel].groupby(['hour']).agg(func=fun
```

In [147…

```
fig,ax = plt.subplots(2, 1, figsize=(6,6), dpi=100, sharey=True)
ax = ax.flatten()

x = agg3.index.values
ax[0].fill_between(x, agg3[('eps','min')], agg3[('eps','max')], color='0.
ax[0].fill_between(x, agg3[('eps','p25')], agg3[('eps','p75')], color='0.
ax[0].plot(x, agg3[('eps','p50')], '-k', label='all data')
ax[0].set_xlabel('Month')
ax[0].legend(fontsize='small', loc='lower left', ncol=3)
ax[0].set_xticks(range(1,12,2))
ax[0].set_xticklabels(['Jan','Mar','May','Jul','Sep','Nov'])
ax[0].axvline(4, linestyle='dotted', color='k')
ax[0].axvline(10, linestyle='dotted', color='k')

#plt.vlines()
x = agg4.index.values
ax[1].fill_between(x, agg4[('eps','min')], agg4[('eps','max')], color='0.
ax[1].fill_between(x, agg4[('eps','p25')], agg4[('eps','p75')], color='0.
ax[1].plot(x, agg4[('eps','p50')], '-k', label='all data')
ax[1].plot(x, agg5[('eps','p50')], '--k', label='night, obs.season')
ax[1].set_xlabel('Hour (UT)')
ax[1].set_xticks(range(0,25,2))
ax[1].legend(fontsize='small', loc='lower left', ncol=2)
ax[1].axvline(22, linestyle='dotted', color='k')
ax[1].axvline(10, linestyle='dotted', color='k')

for i in [0,1]:
    ax[i].set_ylim(1.0, 1.8)
    ax[i].set_ylabel('Seeing (arcsec)')
```

```
plt.tight_layout()
plt.show()
```

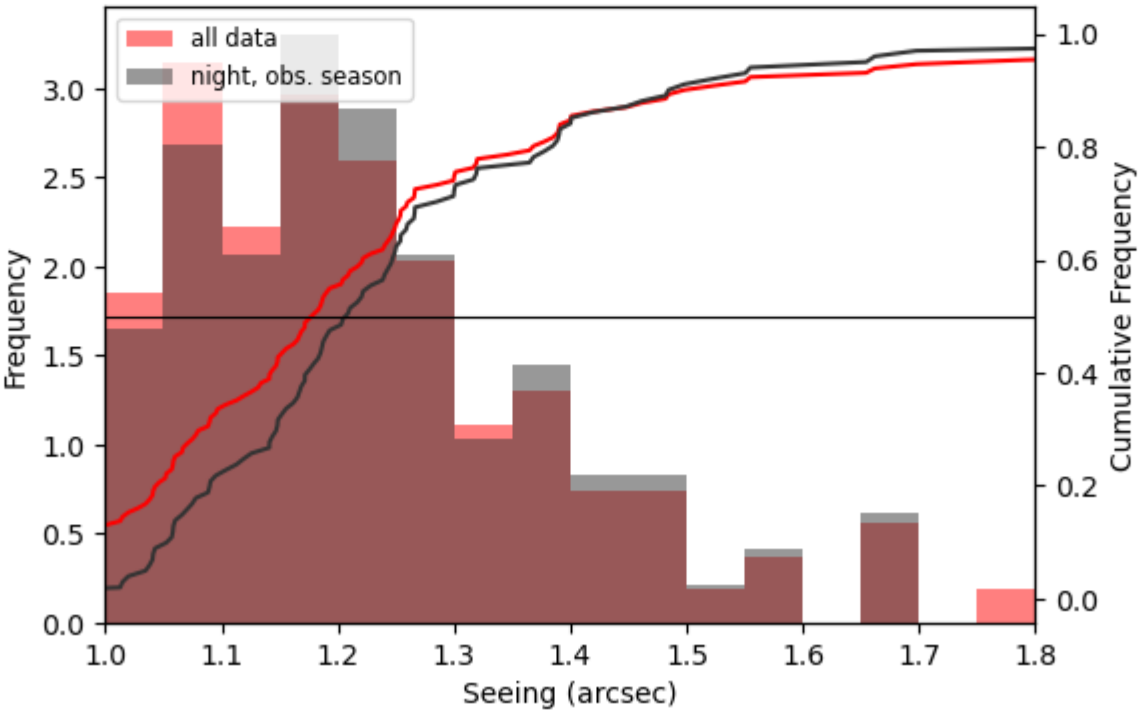




In [151…

```
def plot_dist(df, col='eps', legend_loc='upper left', ax=None,
              title='', save=''):
    params = {'eps': {'bins': [1.0, 1.81, 0.05],
                      'limit': [1.0, 1.8],
                      'label': 'Seeing (arcsec)'}
             }

    param = params[col]
    bins = np.arange(*param['bins'])
    color = ['red', '0.2']
    night = np.logical_and(df['hour'].between(10, 22), df['month'].isin([

    if ax is None:
        fig,ax = plt.subplots(figsize=(6,4), dpi=100, facecolor='w')
    ax.hist(df[col], bins=bins, density=True, histtype='stepfilled',
            label='all data', color=color[0], alpha=0.5)
    ax.hist(df[night][col], bins=bins, density=True, histtype='stepfilled
            label='night, obs. season', color=color[1], alpha=0.5)
    ax.set_xlabel(param['label'])
    ax.set_ylabel('Frequency')
    ax.set_xlim(*param['limit'])
```

```
    if title != '':
        ax.set_title(title, loc='left', fontsize='small', fontweight='bol

    x0 = np.sort(df[col].values)
    l0 = len(x0)
    y0 = np.cumsum(np.ones(l0))/l0
    x1 = np.sort(df[night][col].values)
    l1 = len(x1)
    y1 = np.cumsum(np.ones(l1))/l1

    ax2 = ax.twinx()
    ax2.plot(x0, y0, color=color[0])
    ax2.plot(x1, y1, color=color[1])
    ax2.hlines(0.5, *param['limit'], color='k', lw=0.8)
    ax2.set_ylabel('Cumulative Frequency')

    ax.legend(fontsize='small', loc='upper left')
    if save != '':
        plt.tight_layout()
        plt.savefig(f'plot/seeing_dist_{col}.png')
        plt.savefig(f'plot/seeing_dist_{col}.svg')
    plt.show()

plot_dist(seeing)
```



In [ ]: