太陽X線の分析

説明は次のホームページにあります。

http://swc.nict.go.jp/trend/flare.html (http://swc.nict.go.jp/trend/flare.html)

F10.7 cm電波放射 ¶

10.7 cm（2800 MHz）の太陽電波フラックスは、太陽活動の優れた指標です。しばしばF10.7インデックスと呼ばれ、太陽活動の最も長い実行記録の1つです。 F10.7の電波放射は、彩層で高く、太陽大気のコロナで低く発生します。 F10.7は、太陽黒点の数や、多数の紫外線（UV）および可視の太陽放射照度の記録とよく相関しています。 F10.7は1947年以来、最初にオンタリオ州のオタワで一貫して測定されました。その後、カナダのブリティッシュコロンビア州にあるペンティクトン電波天文台で。多くのソーラーインデックスとは異なり、F10.7電波束は、あらゆる種類の天候において、地表から日常的に簡単かつ確実に測定できます。 「太陽フラックス単位」（s.f.u.: $sfu = 10^{-22} W m^{-2} Hz^{-1}$）で報告されるF10.7は、太陽サイクルの過程で、50 s.f.u.未満から300 s.f.u.まで変化する可能性があります。

https://spaceweather.gc.ca/solarflux/sx-5-en.php (https://spaceweather.gc.ca/solarflux/sx-5-en.php)

In [18]:
```
%pwd
```

Out[18]:
```
'C:¥¥Users¥¥david¥¥SPRING2020-PRACTICE_1'
```

In [2]:
```
import pandas as pd
```

In [20]:
```
data = pd.read_csv("solflux_monthly_average.txt",header=1,sep='¥s+')
```

In [21]:
```
data.head(10)
```

Out[21]:

	Year	Mon	Obsflux	Adjflux	Absflux
0	2004	10	137.56	135.60	122.04
1	2004	11	115.98	113.46	102.12
2	2004	12	95.66	92.69	83.42
3	2005	1	102.86	99.59	89.63
4	2005	2	97.30	94.95	85.45
5	2005	3	90.04	89.14	80.22
6	2005	4	85.92	86.55	77.90
7	2005	5	101.65	103.90	93.50
8	2005	6	94.09	97.06	87.36
9	2005	7	100.64	103.93	93.54

In [22]:
```
data.shape
```

Out[22]:
```
(189, 5)
```

In [23]:
```
#data['MonthYear'] = data['Mon'].str.cat(data['Year'],sep="-")
#pd.to_datetime(data[['Year', 'Mon']])
#pd.concat(data['Mon'],data['Year'])
#data['Date'] = pd.to_datetime(data[['Year', 'Mon']].assign(Day=1))
data['Date'] = pd.to_datetime(data.Year.astype(str) + '-' + data.Mon.astype(str))
```

In [24]:
```
data.head(5)
```

Out[24]:

	Year	Mon	Obsflux	Adjflux	Absflux	Date
0	2004	10	137.56	135.60	122.04	2004-10-01
1	2004	11	115.98	113.46	102.12	2004-11-01
2	2004	12	95.66	92.69	83.42	2004-12-01
3	2005	1	102.86	99.59	89.63	2005-01-01
4	2005	2	97.30	94.95	85.45	2005-02-01

In [25]:
```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (20,15) # この設定があれば、全てのplotに提供される
```

In [26]:
```
%matplotlib inline
```

In [27]:
```
data.set_index('Date',inplace=True)
```

In [28]:
```
data.tail()
```

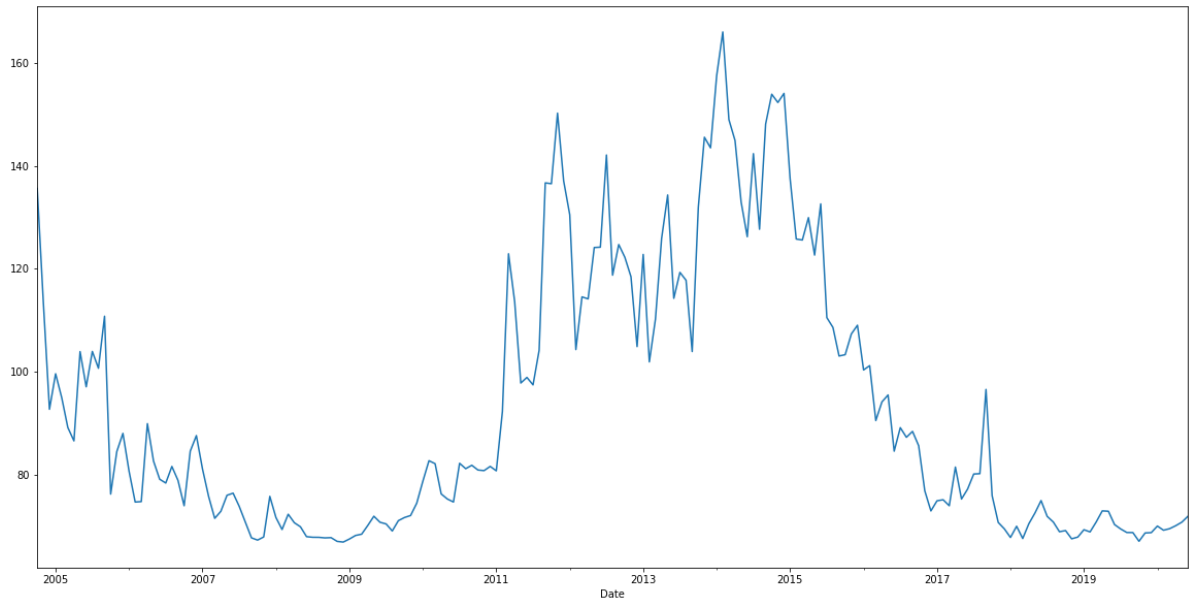
Out[28]:

	Year	Mon	Obsflux	Adjflux	Absflux
Date					
2020-02-01	2020	2	70.90	69.19	62.27
2020-03-01	2020	3	70.19	69.49	62.54
2020-04-01	2020	4	69.56	70.07	63.06
2020-05-01	2020	5	69.21	70.76	63.69
2020-06-01	2020	6	69.70	71.91	64.71

In [29]:
```
#data[data['Date']=='2019-09-01']
```

In [31]:
```
plt.figure(figsize=(20,10))
data['Adjflux'].plot()
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1a6d57eecc0>



In [14]:
```
data.describe()
```

Out[14]:

	Year	Mon	Obsflux	Adjflux	Absflux
count	189.000000	189.000000	189.000000	189.000000	189.000000
mean	2012.126984	6.476190	92.657884	92.582275	83.323598
std	4.573834	3.481824	26.010283	25.740788	23.166747
min	2004.000000	1.000000	65.670000	66.890000	60.200000
25%	2008.000000	3.000000	71.480000	71.050000	63.950000
50%	2012.000000	6.000000	81.420000	81.460000	73.310000
75%	2016.000000	10.000000	109.680000	110.220000	99.200000
max	2020.000000	12.000000	170.130000	166.010000	149.410000

In [ ]:
```
```

太陽物理学

このチュートリアルでは、さまざまな測定値を含む例として、太陽物理学データセットを使用します。ここで使用するデータセットのバージョンは、未加工のテキストファイルとして入手でき、1963年の初めから1時間ごとの測定値が含まれています。

https://spdf.gsfc.nasa.gov/pub/data/omni/low_res_omni/ (https://spdf.gsfc.nasa.gov/pub/data/omni/low_res_omni/)

- omni2_all_years.dat
- omni2.text

Pandasモジュールで次の列を抽出する：

- 列1、2、3は、各測定の年、年間通算日（DOY）、および時刻を示します。
- 列40：黒点数（R）-太陽の表面上のスポットの数。
- 列41：Dstインデックス-地球の表面で測定された毎時の磁気活動インデックス（nT）
- 列51：F10.7インデックス-10.7cmの電波束（つまり、太陽がその波長でどれだけ明るいか）、「太陽束の単位」（sfu）（パンダ列-1）

このデータを調査して、太陽の条件（RおよびDst）、F10.7cmの電波束、および地球の磁気条件（Dst）の間に関係があるかどうかを確認します。

In [3]:
```
df = pd.read_csv("omni2_all_years.dat",
                 delim_whitespace=True,
                 usecols=[0, 1, 2, 39, 40, 50],
                 names=["Year", "DOY", "Hour", "R", "Dst", "F10.7"])
```

In [4]:
```
df.head()
```

Out[4]:

	Year	DOY	Hour	R	Dst	F10.7
0	1963	1	0	33	-6	999.9
1	1963	1	1	33	-5	999.9
2	1963	1	2	33	-5	999.9
3	1963	1	3	33	-3	999.9
4	1963	1	4	33	-3	999.9

In [5]:
```
df.tail()
```

Out[5]:

	Year	DOY	Hour	R	Dst	F10.7
508435	2020	366	19	999	99999	999.9
508436	2020	366	20	999	99999	999.9
508437	2020	366	21	999	99999	999.9
508438	2020	366	22	999	99999	999.9
508439	2020	366	23	999	99999	999.9

これでデータが読み込まれたので、少し修正してより使いやすくします。 最初に、インデックスを整数のシーケンスとしての現在の状態から、Python datetimeオブジェクトに基づくより機能的なpandas.DatetimeIndexに変更します。

pandas.to_datetime（）関数を使用して、「Year」、「DOY」、および「Hour」列から新しいインデックスを作成し、それをdfの.indexプロパティに直接割り当てて、不要な列を削除します。

In [6]:
```
df.index = pd.to_datetime(df["Year"] * 100000 + df["DOY"] * 100 + df["Hour"], format="%Y%j%H")
df = df.drop(columns=["Year", "DOY", "Hour"])
```

In [7]:
```
import numpy as np
```

df ["Year"] *100000 + df ["DOY"]* 100 + df ["Hour"]は、列を結合して、「%Y %j%H "フォーマット指定子。 df.head（）が表示されるはずです。

このデータセットでは、データのギャップは9で埋められています。 これらの出現箇所をNaNで置き換えることができます。

In [8]:
```
df = df.replace({"R":999,
                 "Dst":99999,
                 "F10.7":999.9}, np.nan)
```

これにより、データセット内のレコード数は、最初にサンプリングされた時間と最後にサンプリングされた時間の間の時間数と同じであることがわかります。 私たちは、約50万件のレコードを生成する55年を超える毎時サンプルを処理しています。

In [9]:
```
print("Dataframe shape: ", df.shape)
dt = (df.index[-1] - df.index[0])
print("Number of hours between start and end dates: ", dt.total_seconds()/3600 + 1)
```

```
Dataframe shape:  (508440, 3)
Number of hours between start and end dates:  508440.0
```

In [10]:
```
h, d, y = 24, 365, 55
print(f"{h} hours/day * {d} days/year * {y} years = {h*d*y} hours")
```

```
24 hours/day * 365 days/year * 55 years = 481800 hours
```

In [11]:
```
print("{} hours/day * {} days/year * {} years = {} hours".format(h, d , y, h*d*y))
print("%d hours/day * %d days/year * %d years = %d hours"%(h, d , y, h*d*y))
```

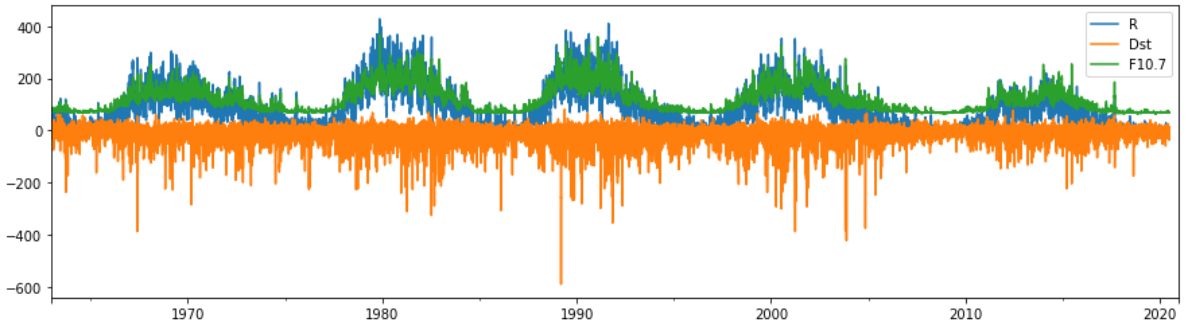
```
24 hours/day * 365 days/year * 55 years = 481800 hours
24 hours/day * 365 days/year * 55 years = 481800 hours
```

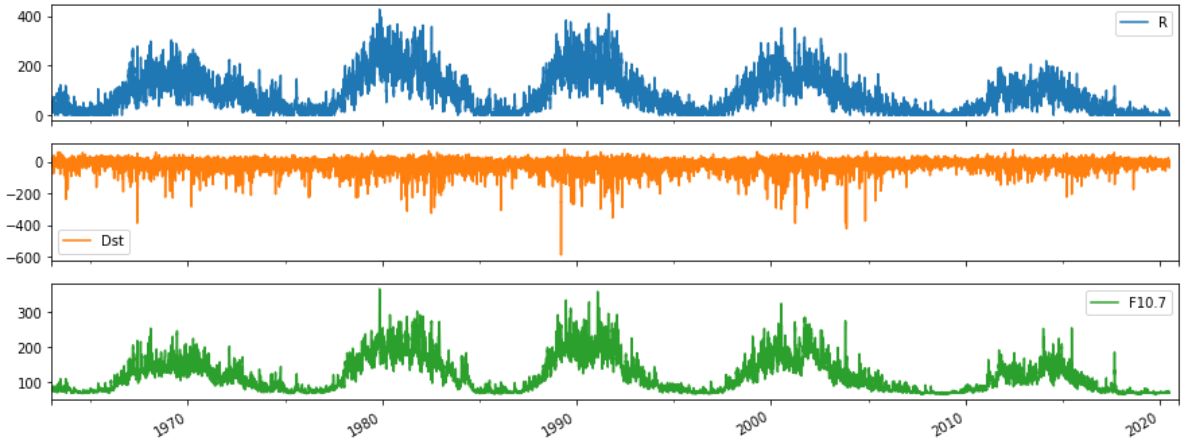
データフレームの内容をプロットする

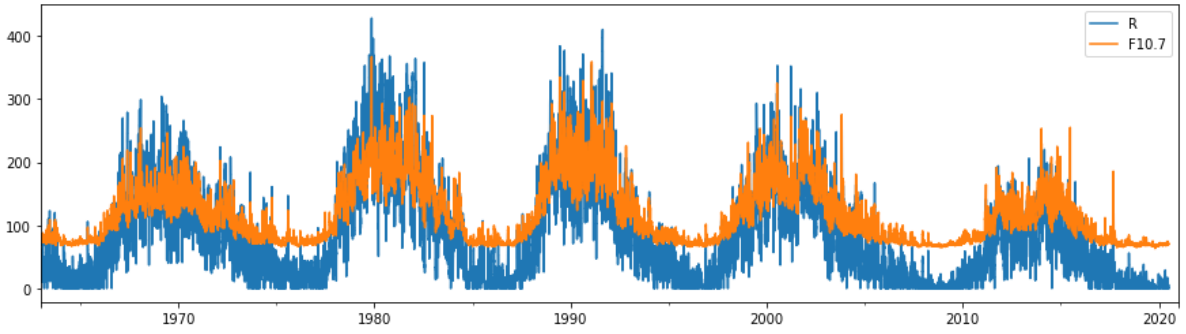
これでデータは「分析可能な」形式になり、検査を開始する必要があります。 まず、.plot（）メソッドを使用します。 以下のそれぞれを試して、得られるものを比較してください。

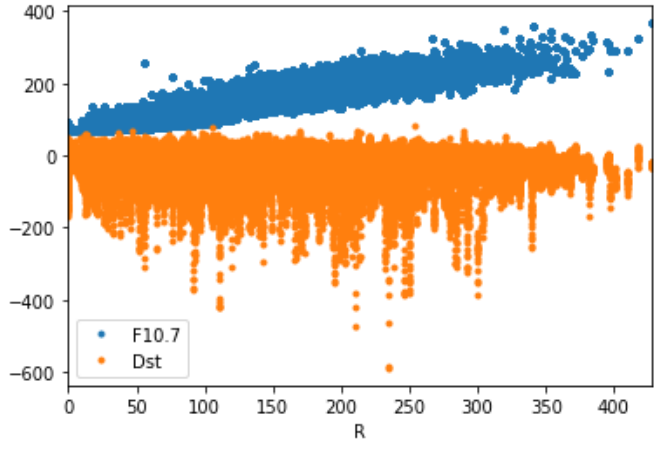
In [12]:
```
df.plot(figsize=(15,4))
df.plot(subplots=True, figsize=(15,6))
df.plot(y=["R", "F10.7"], figsize=(15,4))
df.plot(x="R", y=["F10.7", "Dst"], style='.')
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x234252072b0>









これにより、4つの異なるプロットがすばやく実現されました。

1. 1つの軸にすべての時系列をプロットする
2. それらを個別のサブプロットにプロットして、それらをより明確に表示します（x軸を共有）
3. 選択した列をプロットする
4. 他の1つに対して2つの変数をプロットする

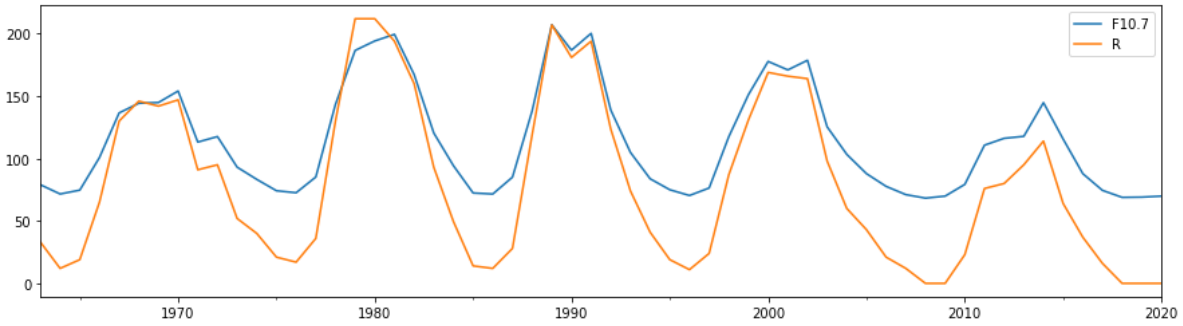
これで、データの感触をつかむことができます。 F10.7とRはよく相関しており、それぞれ時間の経過に応じて5つのピークが等間隔に配置されています。 すべての測定値に多くのノイズがあり、Dstとの関係を確認することは困難です。 では、トレンドや関係をさらに深く見るために何ができるでしょうか？

リサンプリング、ローリング計算、および差分

データのノイズを減らすために、データを平滑化できます。 これを行うにはさまざまな方法があるため、使用する方法と必要な平滑化の程度について選択する必要があります。 pandasは、.resample（）メソッドでリサンプリングすることにより、データのリズムを減らす便利な方法を提供します。

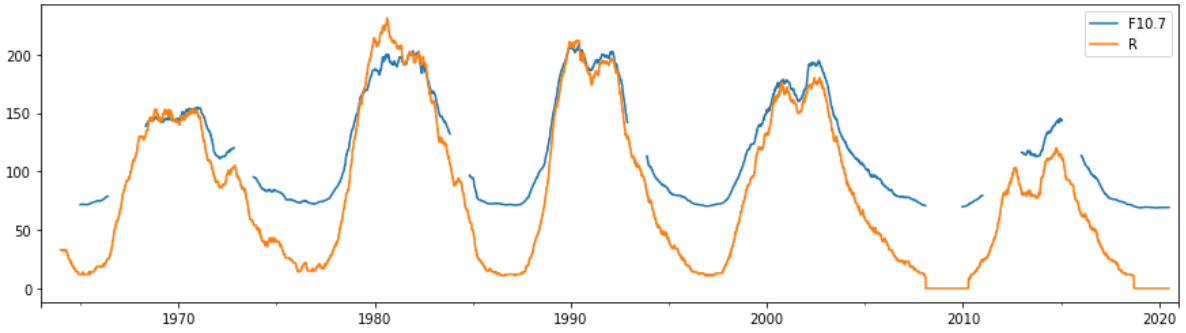
In [17]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
df[["F10.7", "R"]].resample("1y").median().plot(figsize=(15,4))
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2341f1204a8>

<Figure size 1440x720 with 0 Axes>
```



In [18]:
```
plt.figure(figsize=(20,15))
df[["F10.7", "R"]].rolling(24*365).median().plot(figsize=(15,4))
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2341f20a518>

<Figure size 1440x1080 with 0 Axes>
```



ここでは、df [["F10.7"、 "R"]]で関心のある列を含むデータフレームを抽出し、年ベースの「リサンプラー」オブジェクトを生成しました。 各年の間隔で中央値を取る。

.resample（）は、高周波ノイズを含まない低いケイデンスデータセットを提供します。 これと同様のローリングウィンドウ計算は、入力と同じデータのリズムを返しますが、計算は各データポイントについて指定された幅のローリングウィンドウで実行されます。 これを行うには.rolling（）メソッドを使用できます。 ここでは、移動中央値フィルターを作成します。

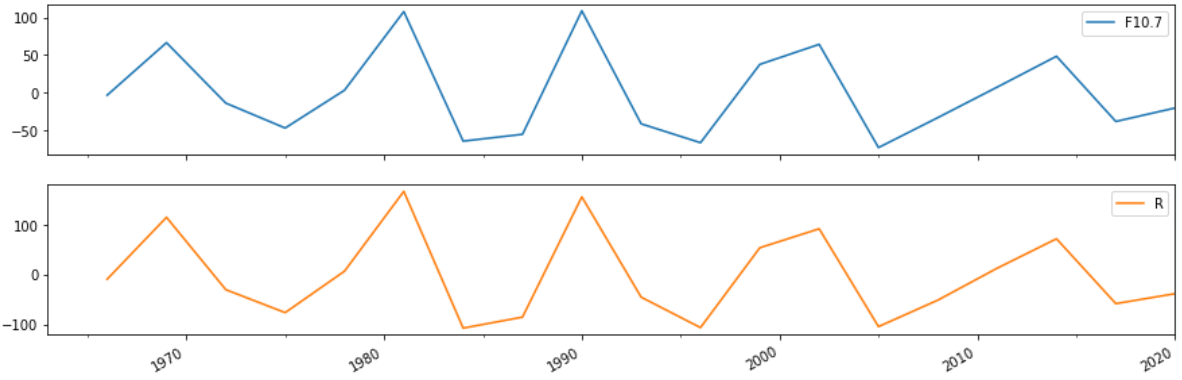
ローリング計算はウィンドウのサイズを引数として取り、リサンプリングは周波数指定子を引数として取ります。 注意：デフォルトでは、計算された各ウィンドウ内でギャップが許可されていないため、F10.7時系列にいくつかのギャップの外観を確認できます。この動作は、min_periods引数で変更できます。

ドキュメントを参照して、リサンプラーやローリングオブジェクトで実行できる他の計算を確認してください。

差分は、時系列アルゴリズムの一部になることができる便利なツールであることがよくあります。 例として、周期信号をより明確に分離するために平滑化と差分を使用する方法をご覧ください。

In [22]:
```
plt.figure(figsize=(20,15))
df[["F10.7", "R"]].resample("3y").median().diff().plot(subplots=True, figsize=(15,5))
```

Out[22]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000002340306FBE0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000023403F26860>],
      dtype=object)

<Figure size 1440x1080 with 0 Axes>
```



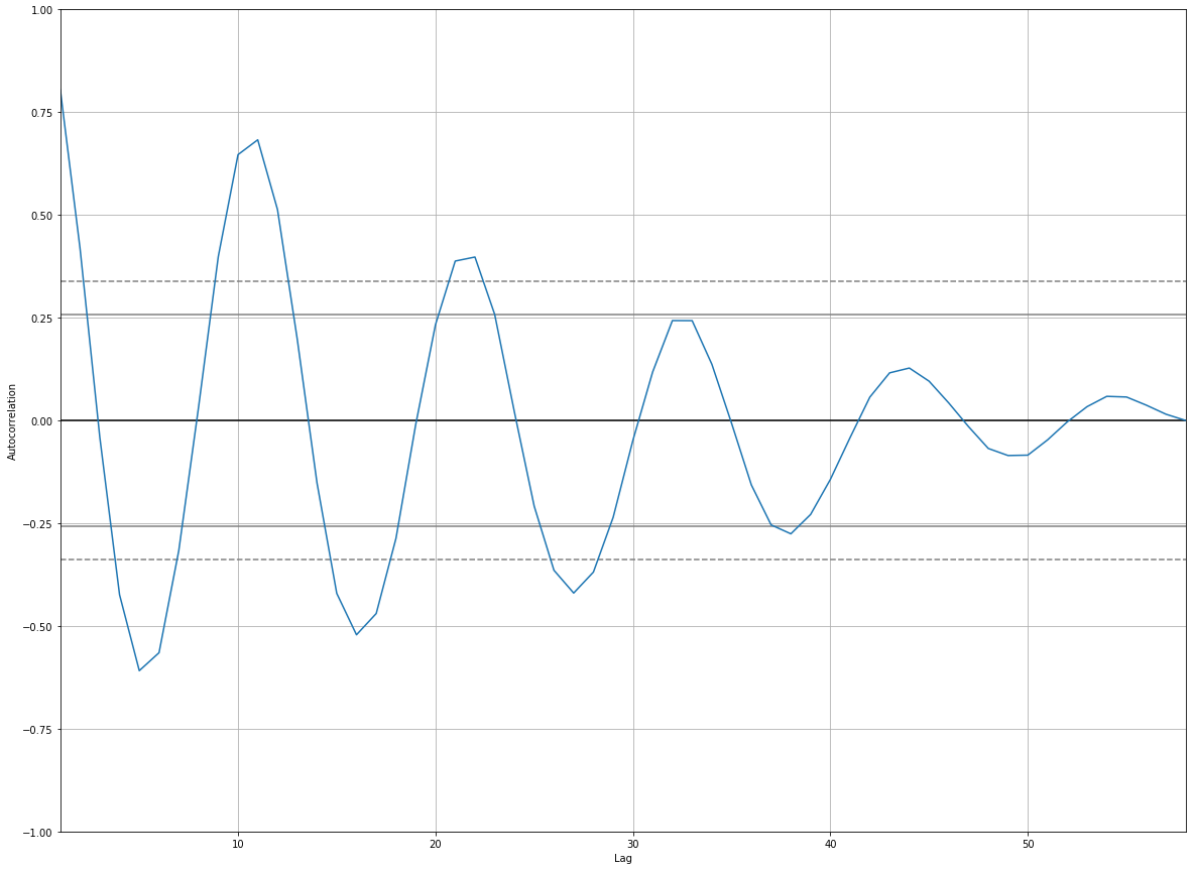
サイクルの各周期の最大値と最小値の中心は、この曲線の最大値と最小値によって定義できます。

周期性と相関の特定

RとF10.7には約10年の周期があることが目で確認できます。 この周期性を特定するための便利な高レベルツールは、pandas.plotting.autocorrelation_plot（）です。

In [24]:
```
plt.figure(figsize=(20,15))
pd.plotting.autocorrelation_plot(df["R"].resample("1y").median())
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x234053b5da0>



In [28]:
```
Dst_count = df["Dst"].where(df["Dst"]<-100).resample("1y").count()
Dst_count = Dst_count.reindex(df.index, method="bfill")
```

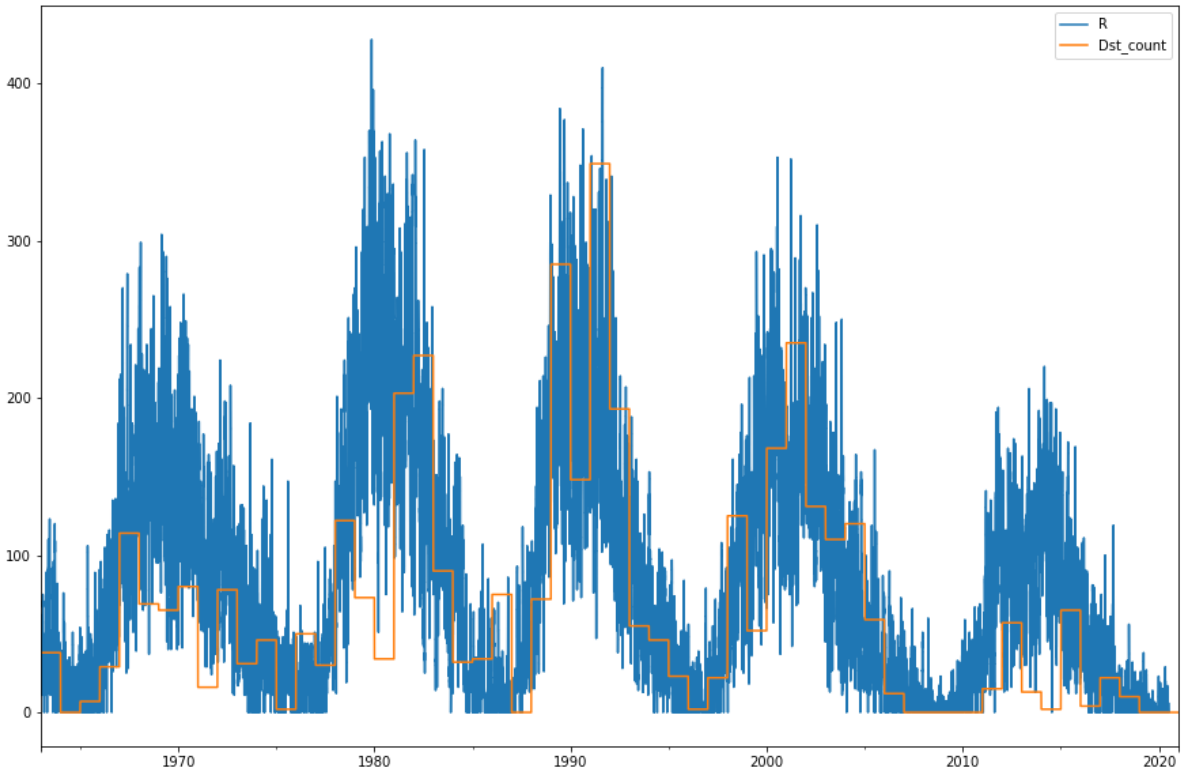
これにより、自己相関プロットが作成されます。ある範囲の遅延時間における時系列とそれ自体との相関関係です。ダウンサンプリングされた年次時系列に適用して、計算を大幅に高速化しました。時系列のリズムは1年であるため、「ラグ」軸は年数で測定されます。最初のピーク（0の遅れの後）は約11年です。これは、系列が11年の遅れ時間でそれ自体とよく相関していることを意味します。これはよく知られた太陽活動サイクルです。

Dstインデックスをもう一度見て、Rへの接続があるかどうかを確認してみましょう。調査している量のコンテキストを検討することは役に立ちます。黒点番号Rは太陽活動を示し、Dstは地磁気活動を示します。これは、地球の周りの時間変化する大規模な電流によって生成される磁場です。 Dstを平滑化してノイズを減らし、Rとの相関があるかどうかを確認することもできますが、これから何かを証明するのは難しいことをここで説明できます。 Dstの変動は、実際には「地磁気嵐」と呼ばれる個別のイベントで発生する傾向があり、Dstが0nTを大幅に下回り、0時間に回復するまでに数時間または数日かかります。 Dstが-100nTを下回った場合として、大きな嵐を分類できます。これを使用して、大きな嵐の発生を検索してみましょう！

Dstが-100を下回る場所をdf ["Dst"]。where（df ["Dst"] <-100）でマスクして、この条件を満たす年ごとのエントリ数をカウントできます。

In [29]:
```
plt.figure(figsize=(20,15))
df["Dst_count"] = Dst_count
df.plot(y=["R", "Dst_count"], figsize=(15,10));
```

```
<Figure size 1440x1080 with 0 Axes>
```



高い黒点数（太陽周期のピーク）と大嵐の発生率には相関があるようです。 ただし、このストームレートにはさらに多くのバリエーションがあります。黒点がたくさんあるからといって、嵐が多いとは限らず、黒点が少ない場合でも嵐が発生する可能性があります。

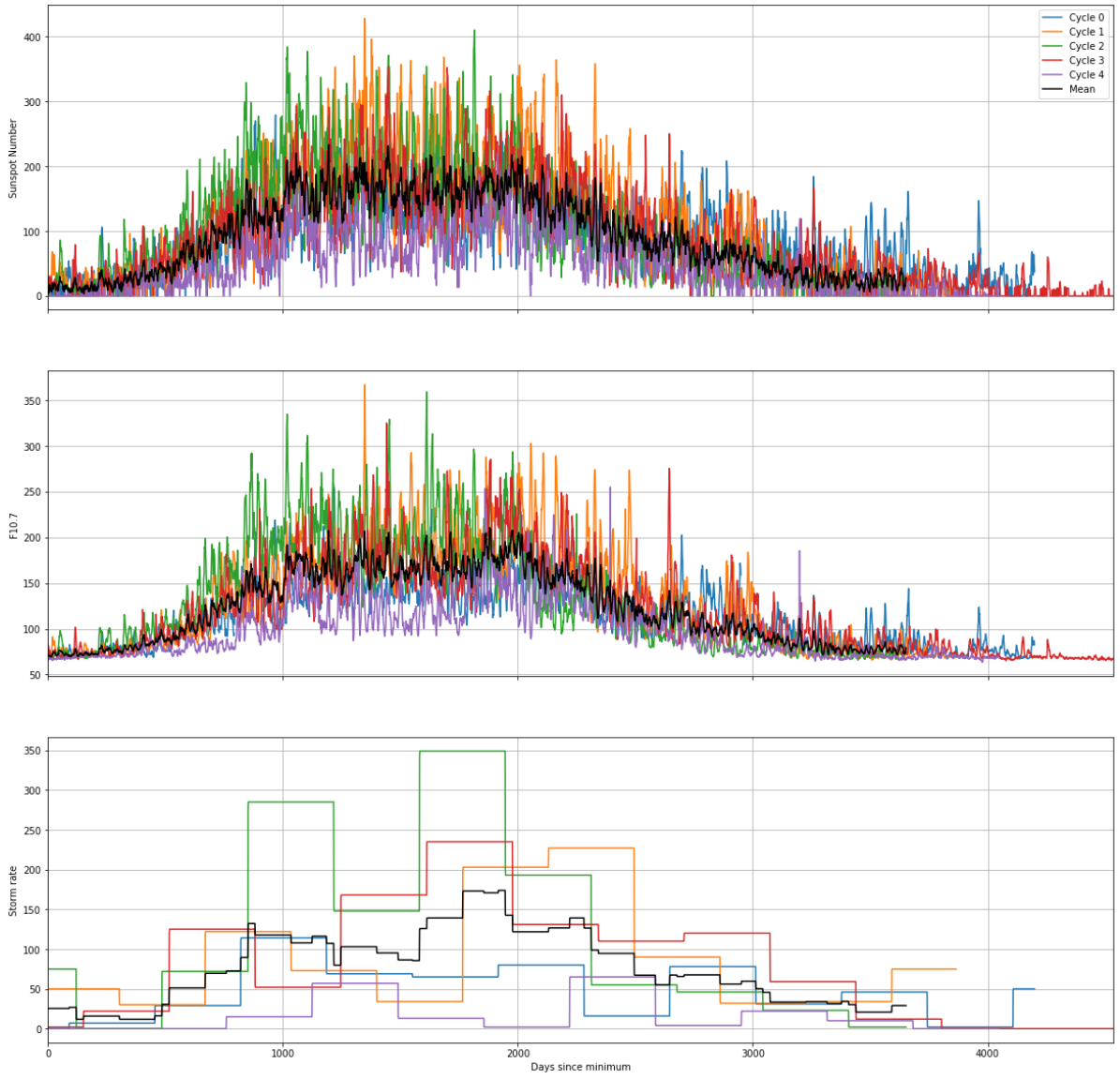
分割と積み重ねのサイクル

時系列を構成要素のサイクルに分割し、それらを積み重ねてみましょう。 これには、pandasとmatplotlibを使用したより複雑な作業が必要です。 この時点で、日次レートにダウンサンプリングすることもできます。これにより、プロットが少し明確になり、生成が速くなります。

In [31]:
```
# https://en.wikipedia.org/wiki/List_of_solar_cycles
minima = ["1964-10", "1976-03", "1986-09", "1996-08", "2008-12", "2019-12"]
df_daily = df.resample("1D").mean()

def split_into_cycles(df):
    """Returns a list of dataframes, one for each solar cycle"""
    cycles = []
    # Split by solar cycle
    for start, end in zip(minima[0:-1], minima[1:]):
        cycle = df[start:end]
        # Convert from dates to days from minimum
        cycle.index = (cycle.index - cycle.index[0]).days
        # Extend so that each cycle lasts a full 5000 days (filled with nan)
        ix = pd.Int64Index(np.arange(0,5000))
        cycle.reindex(ix)
        cycles.append(cycle)
    return cycles

cycles = split_into_cycles(df_daily)
```

これで、それぞれが異なるサイクルを含む5つのデータフレームを含むリスト、cyclesができました。 各データフレームで、インデックスを最小からの日数に変更し、.reindex（）を使用してそれらをすべて同じ長さに修正して、それらに対して算術演算を実行できるようにしました。 以下は、各パラメータのプロットを作成し、サイクルを互いに重ね合わせます。 この例では、最初にmatplotlibを使用して（sharex = Trueを使用して各プロットのx軸をリンクします）、図とその軸を作成し、次にpandasプロットコマンドに指示して、それぞれがプロットする軸にポイントします ax kwargを使用します。 積み上げ時系列の平均も計算します。

In [35]:
```
fig, axes = plt.subplots(3, 1, figsize=(20,20), sharex=True)
for i, cycle in enumerate(cycles):
    cycle["R"].plot(ax=axes[0], label=f"Cycle {i}")
    cycle["F10.7"].plot(ax=axes[1])
    cycle["Dst_count"].plot(ax=axes[2])
N_cycles = len(cycles)
(sum(cycles)["R"]/N_cycles).plot(ax=axes[0], color="black", label="Mean")
(sum(cycles)["F10.7"]/N_cycles).plot(ax=axes[1], color="black")
(sum(cycles)["Dst_count"]/N_cycles).plot(ax=axes[2], color="black")
axes[0].legend()
axes[0].set_ylabel("Sunspot Number")
axes[1].set_ylabel("F10.7")
axes[2].set_ylabel("Storm rate")
axes[2].set_xlabel("Days since minimum")
for ax in axes:
    ax.grid()
```



これは、周期が互いにどのように異なるかを確認するのに役立ちます。たとえば、最新の周期は、太陽条件と地磁気嵐の速度の両方で、平均より一貫して低いです。 サイクルの平均を構築することにより、実際には各サイクルにわたって同様のパターンを強化し、ランダムノイズの影響を低減しています。 これは、重ね合わせたエポック分析と呼ばれる手法の基礎であり、ノイズの多い時系列間の周期性と類似性を識別するのに役立ちます。

概要

Pandasの力を使用して時系列を調査する最初のステップをどのように実行できるかを調査しました。 メソッドをつなぎ合わせてデータフレームで複雑な操作を1行で実行し、結果を簡単にプロットする方法を示しました。

In [ ]: