Pandas Time Series

Today we are going to look at CO_2 data and try to learn about time series. I will introduce it and then you will be in charge of analyzing CO_2 data from Mauna Lao. We are going to start by looking at the data from the Scripps Pier.

In [1]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
from scipy import stats
from matplotlib.backends.backend_pdf import PdfPages
```

First go get CO2 data for Scripps. http://scrippsco2.ucsd.edu/data/atmospheric_co2/ljo and the La Jolla Pier.

- I did flask daily values.
- Download and Open the CSV.
- rename the columns and called them Date, HR, Excel, Year, Flask, Flag, CO2.
- Then we can skip the rows and columns we don't want. You might have to change this. The number wiggles

I kept all the information in place and used skiprows to skip what we don't want.
http://pandas.pydata.org/pandas-docs/stable/ T

Start by just reading in the data. do you get the data? Then get more clever as you go. Here is the read_csv information. https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html or http://pandas.pydata.org/pandas-docs/stable/io.html

You just need to try and do to learn.

In [6]:
```
df_scripps=pd.read_csv('daily_merge_co2_ljo.csv',skiprows=70)
```

Flashback. Remember I am just printing head() to save space on the printouts

In [7]:
```
print (df_scripps.head())
```

```
      Date      HR    Excel      Year  Flask  Flag     CO2
0  3/21/57   00:00  20900.0  1957.216      1    -2  314.75
1  3/26/57   00:00  20905.0  1957.230      1    -2  316.47
2   4/6/57   00:00  20916.0  1957.260      1    -2  315.98
3  4/10/57   00:00  20920.0  1957.271      1    -2  315.49
4  4/23/57   00:00  20933.0  1957.307      1    -2  315.37
```

But we only want to the Date, Hr, Flags, and CO2. So just grab those columns using use_cols

I just learned a neat trick and the reason why my printouts might look different then yours sometimes. Sometimes when I print it actually just shows df.info() which can be useful. so I am going to use this call to get us smaller descriptive outputs.

In [10]:
```
df_scripps=pd.read_csv('daily_merge_co2_ljo.csv',skiprows=70,usecols=[0,1,5,6])
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1541 entries, 0 to 1540
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Date    1541 non-null   object
```

```
 1   HR      1541 non-null   object
 2   Flag    1541 non-null   int64
 3   CO2     1541 non-null   float64
dtypes: float64(1), int64(1), object(2)
memory usage: 48.3+ KB
None
```

This does the same thing but using column names.

In [13]:
```
df_scripps=pd.read_csv('daily_merge_co2_ljo.csv',skiprows=70
                       ,usecols=['Date','HR','Flag','CO2'])
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1541 entries, 0 to 1540
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1541 non-null   object
 1   HR      1541 non-null   object
 2   Flag    1541 non-null   int64
 3   CO2     1541 non-null   float64
dtypes: float64(1), int64(1), object(2)
memory usage: 48.3+ KB
None
```

Now lets turn the Date into a datetime index. It is critical to get a datetime index. I can't stress that enough. So you can make sure it

Remember you need a date time index!

Remember what I said. The date time index is critical and can really mess you up if you don't have it!

Don't forget this!!!

First we are making Date the index column

In [14]:
```
df_scripps=pd.read_csv('daily_merge_co2_ljo.csv',skiprows=70
                       ,usecols=[0,1,5,6],index_col='Date')
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1541 entries, 3/21/57 to 9/6/23
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   HR      1541 non-null   object
 1   Flag    1541 non-null   int64
 2   CO2     1541 non-null   float64
dtypes: float64(1), int64(1), object(1)
memory usage: 48.2+ KB
None
```

We now have the Dates as an index but we don't have a datetime index. We need that! parse_dates tells it it is a date

CRITICAL!

In [15]:
```
df_scripps=pd.read_csv('daily_merge_co2_ljo.csv',skiprows=70,usecols=[0,1,5,6]\
                       ,index_col='Date',parse_dates=True)
```

```
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1541 entries, 2057-03-21 to 2023-09-06
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   HR      1541 non-null   object 
 1   Flag    1541 non-null   int64  
 2   CO2     1541 non-null   float64
dtypes: float64(1), int64(1), object(1)
memory usage: 48.2+ KB
None
```

Now we have a datetime index! See how it says DatetimeIndex

Now we have that. We could have also.

I am now showing you some different tricks to do the same thing. These might come in handy later

In [16]:
```
df_scripps=pd.read_csv('daily_merge_co2_ljo.csv',skiprows=70,usecols=[0,1,5,6],\
                       parse_dates=True,index_col='Date')
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1541 entries, 2057-03-21 to 2023-09-06
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   HR      1541 non-null   object 
 1   Flag    1541 non-null   int64  
 2   CO2     1541 non-null   float64
dtypes: float64(1), int64(1), object(1)
memory usage: 48.2+ KB
None
```

But you can also just read in and deal with it later. First we can set the Date to a Datetime. Then set that to an index.

In [17]:
```
df_scripps=pd.read_csv('daily_merge_co2_ljo.csv',skiprows=70,usecols=[0,1,5,6])
df_scripps.Date=pd.to_datetime(df_scripps.Date)
df_scripps.set_index('Date',inplace=True)
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1541 entries, 2057-03-21 to 2023-09-06
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   HR      1541 non-null   object 
 1   Flag    1541 non-null   int64  
 2   CO2     1541 non-null   float64
dtypes: float64(1), int64(1), object(1)
memory usage: 48.2+ KB
None
```

We could also do it all at once.

In [18]:
```
df_scripps=pd.read_csv('daily_merge_co2_ljo.csv',skiprows=70,usecols=[0,1,5,6])
df_scripps=df_scripps.set_index(pd.to_datetime(df_scripps.Date))
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1541 entries, 2057-03-21 to 2023-09-06
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
```

```
 0   Date    1541 non-null   object
 1   HR      1541 non-null   object
 2   Flag    1541 non-null   int64
 3   CO2     1541 non-null   float64
dtypes: float64(1), int64(1), object(2)
memory usage: 60.2+ KB
None
```

But back to reading in. We can also add the hours to the date and set the index and dateime all at once. BUT YOU NEED the double brackets on parse dates for this to work because it is a list you are using

In [19]:
```
df_scripps=pd.read_csv('daily_merge_co2_ljo.csv',skiprows=70
                       ,usecols=[0,1,5,6],parse_dates=[['Date','HR']]
                       ,index_col='Date_HR')
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1541 entries, 2057-03-21 00:00:00 to 2023-09-06 17:00:00
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Flag    1541 non-null   int64  
 1   CO2     1541 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 36.1 KB
None
```

This is a fun trick. We can use python to grab the csv file for us off of the website! No need to download. But something is wrong with the file and we will lose the first row of data. I needed to adjust skip rows and then you can see the problem with the names. It changes each year. But this is great because you always get the most up to date file. Do not type the url. Go right click on the file and say copy link and paste in.

In [20]:
```
url='https://scrippsco2.ucsd.edu/assets/data/atmospheric/stations/merged_in_situ_and_flask/
df_scripps=pd.read_csv(url,skiprows=71,usecols=[0,1,5,6])
print (df_scripps)
```

```
      1957-03-21   00:00   -2   314.75
0     1957-03-26   00:00   -2   316.47
1     1957-04-06   00:00   -2   315.98
2     1957-04-10   00:00   -2   315.49
3     1957-04-23   00:00   -2   315.37
4     1957-04-30   00:00   -2   316.11
...          ...     ...  ...      ...
1535  2023-06-22   15:04    0   419.11
1536  2023-07-19   13:16    0   418.02
1537  2023-08-08   12:42    0   416.97
1538  2023-08-24   15:34    4   414.65
1539  2023-09-06   17:00    0   415.99

[1540 rows x 4 columns]
```

Definately funny

Here is how to do it in one fell swoop!

You can see the files on this website. http://scrippsco2.ucsd.edu/data/atmospheric_co2/ljo

This is the most up to date data. Last sample was September 6.

In [21]:
```
url='https://scrippsco2.ucsd.edu/assets/data/atmospheric/stations/merged_in_situ_and_flask/
df_scripps=pd.read_csv(url,skiprows=71,usecols=[0,1,5,6],names=['Date','HR','Flag','CO2']\
                       ,parse_dates=[['Date','HR']]\
                       ,index_col='Date_HR',header=None)
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1541 entries, 1957-03-21 00:00:00 to 2023-09-06 17:00:00
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Flag    1541 non-null   int64  
 1   CO2     1541 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 36.1 KB
None
```

In [22]:
```
url='https://scrippsco2.ucsd.edu/assets/data/atmospheric/stations/merged_in_situ_and_flask/
df_scripps=pd.read_csv(url,skiprows=71,usecols=[0,1,5,6],parse_dates=[[0,1]]\
                       ,index_col=[0],header=None,names=['Date','HR','Flag','CO2'])
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1541 entries, 1957-03-21 00:00:00 to 2023-09-06 17:00:00
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Flag    1541 non-null   int64  
 1   CO2     1541 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 36.1 KB
None
```

Now we can start thinking about the data. Data setup is critical! That is why we repeated doing it so many times. But now lets look at the data!

In [23]:
```
df_scripps
```

Out[23]:

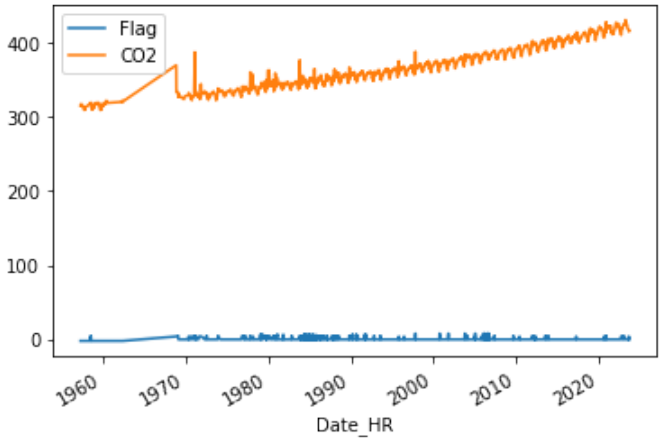
Date_HR	Flag	CO2
1957-03-21 00:00:00	-2	314.75
1957-03-26 00:00:00	-2	316.47
1957-04-06 00:00:00	-2	315.98
1957-04-10 00:00:00	-2	315.49
1957-04-23 00:00:00	-2	315.37
...	...	...
2023-06-22 15:04:00	0	419.11
2023-07-19 13:16:00	0	418.02
2023-08-08 12:42:00	0	416.97
2023-08-24 15:34:00	4	414.65
2023-09-06 17:00:00	0	415.99

1541 rows × 2 columns

You are now an expert at getting data in. You can figure it out! Lets plot the date. We can do the pandas quick plotting.

In [24]:
```
df_scripps.plot()
```

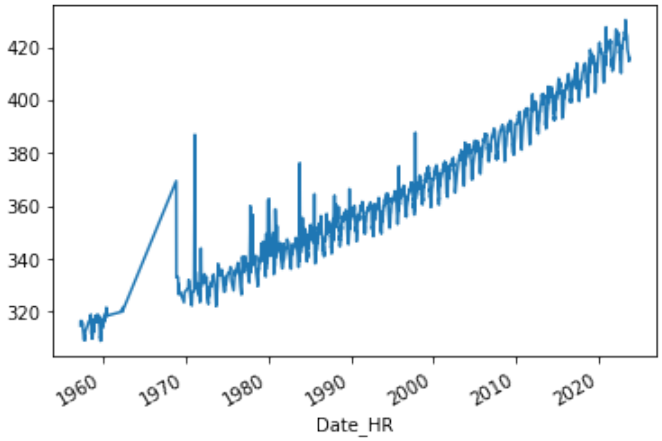
Out[24]:
```
<AxesSubplot:xlabel='Date_HR'>
```



Lets just plot the CO2 data

In [25]:
```
df_scripps.CO2.plot()
```

Out[25]:
```
<AxesSubplot:xlabel='Date_HR'>
```

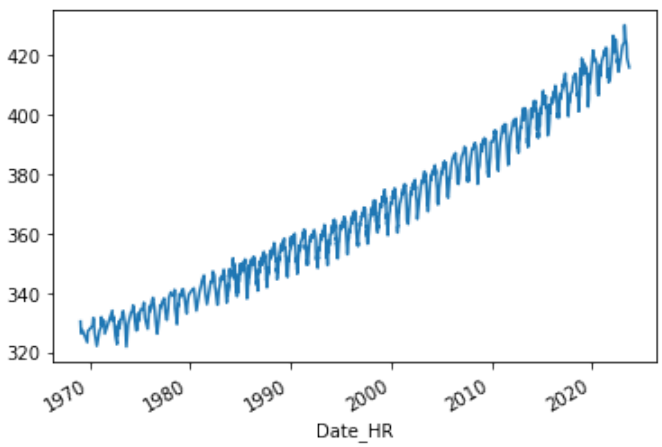


There is obviosly some bad data points. Go look at the csv file header and figure out which values are good and only keep the good data. Good data has a flag of 0. I am going to drop all the other data.

I just figured out this next step. Since we are filtering the data by flag we need to add the .copy(). This saves us a warning later on about SettingWithCopyWarning.

In [32]:
```
df_scripps=df_scripps[df_scripps.Flag==0].copy()   #this .copy is critical for stopping an
df_scripps.CO2.plot()
```

Out[32]:
```
<AxesSubplot:xlabel='Date_HR'>
```

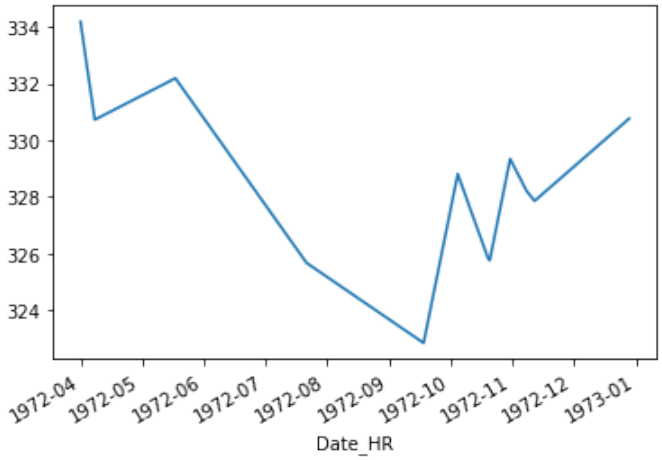


Now that is much nicer!

Now can you plot just the data from your birth year?

In [37]:
```
df_scripps['1972'].CO2.plot()
```

Out[37]:
```
<AxesSubplot:xlabel='Date_HR'>
```

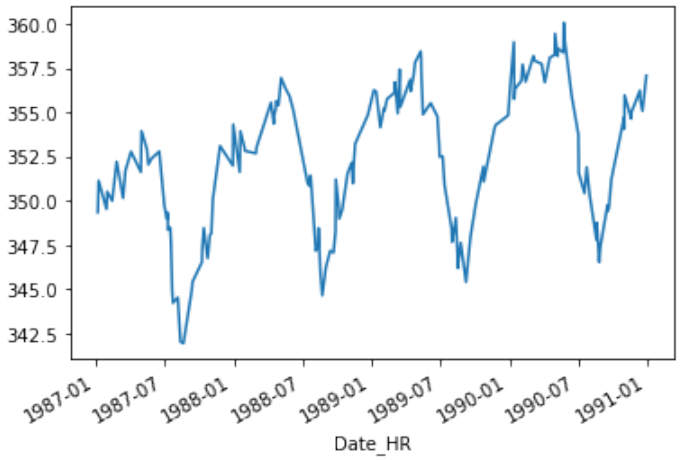


Remember all your slicing? We can also now slice by Date!!!

Now just plot it for the years you were in high school

In [38]:
```
df_scripps['1987':'1990'].CO2.plot()
```

Out[38]:
```
<AxesSubplot:xlabel='Date_HR'>
```



Now I am going to go complicated on you. This is the first time I am trying it and I am not sure it will work. Can we make a boxplot of CO2 for each year? I will make a new column that is the year and the I will make a boxplot by year. Here is the documentation http://pandas-docs.github.io/pandas-docs-travis/visualization.html. This is an important note. To make a new column you have to use the [] notation and not the . notation. See this example.

In [39]:
```
df_scripps['Year']=df_scripps.index.year
print (df_scripps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1278 entries, 1969-01-29 15:47:00 to 2023-09-06 17:00:00
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Flag     1278 non-null   int64
 1   CO2      1278 non-null   float64
 2   Year     1278 non-null   int64
```

```
 3   CO2_10x  1278 non-null   float64
dtypes: float64(2), int64(2)
memory usage: 89.9 KB
None
```

You can also make new columns by using math. What if you wanted to make a new column that was the CO2 times 10 minus 5? I would just make a new column first. If you need a new dataframe you could do that also.

In [40]:
```
# to make a new column
df_scripps['CO2_10x']=df_scripps['CO2']*10.0-5.0
print (df_scripps.head())

#to make a new dataframe
df=pd.DataFrame()
df['CO2_10x']=df_scripps.CO2*10.0-5.0
print (df.head())
```

```
                     Flag     CO2  Year  CO2_10x
Date_HR
1969-01-29 15:47:00     0  330.50  1969   3300.0
1969-02-18 16:28:00     0  326.50  1969   3260.0
1969-02-21 14:12:00     0  326.58  1969   3260.8
1969-04-01 15:00:00     0  327.62  1969   3271.2
1969-06-18 15:11:00     0  326.00  1969   3255.0
                     CO2_10x
Date_HR
1969-01-29 15:47:00   3300.0
1969-02-18 16:28:00   3260.0
1969-02-21 14:12:00   3260.8
1969-04-01 15:00:00   3271.2
1969-06-18 15:11:00   3255.0
```

Flashback to earlier Pandas handouts

Now remember back to boxplots..... Can you make a boxplot of CO2 "by" decade? I couldn't figure it out how to do it without some googling.

Can you make sense of this? https://stackoverflow.com/questions/17764619/pandas-dataframe-group-year-index-by-decade

do you rememeber //?

In [41]:
```
print(1//10,2//20,5//10,10//10,12//10)
```

```
0 0 0 1 1
```

So now to get the decade you can

- //10
- multiply by 10

In [42]:
```
df_scripps.index.year//10*10
```

Out[42]:
```
Int64Index([1960, 1960, 1960, 1960, 1960, 1960, 1960, 1970, 1970, 1970,
            ...
            2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020],
           dtype='int64', name='Date_HR', length=1278)
```

Now you can just make and set a new column

In [43]:
```
df_scripps['decade']=df_scripps.index.year//10*10
```

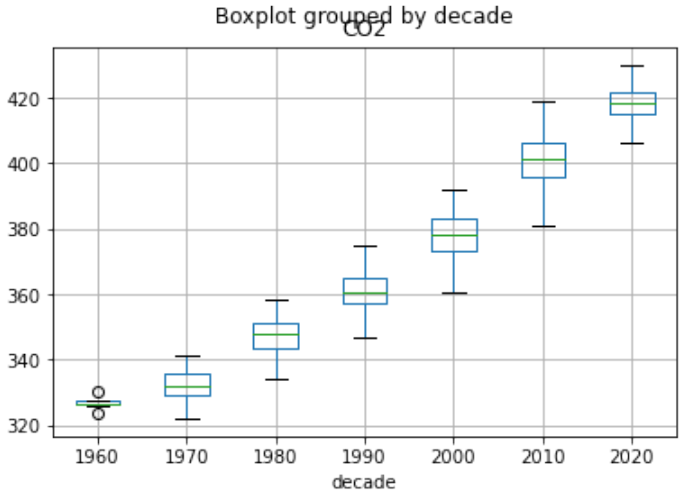
If you get a warning you can do this for SettingWithCopyWarning

In [44]:
```
df_scripps.loc[:,'decade']=df_scripps.index.year//10*10
```

Now make the boxplot!

In [48]:

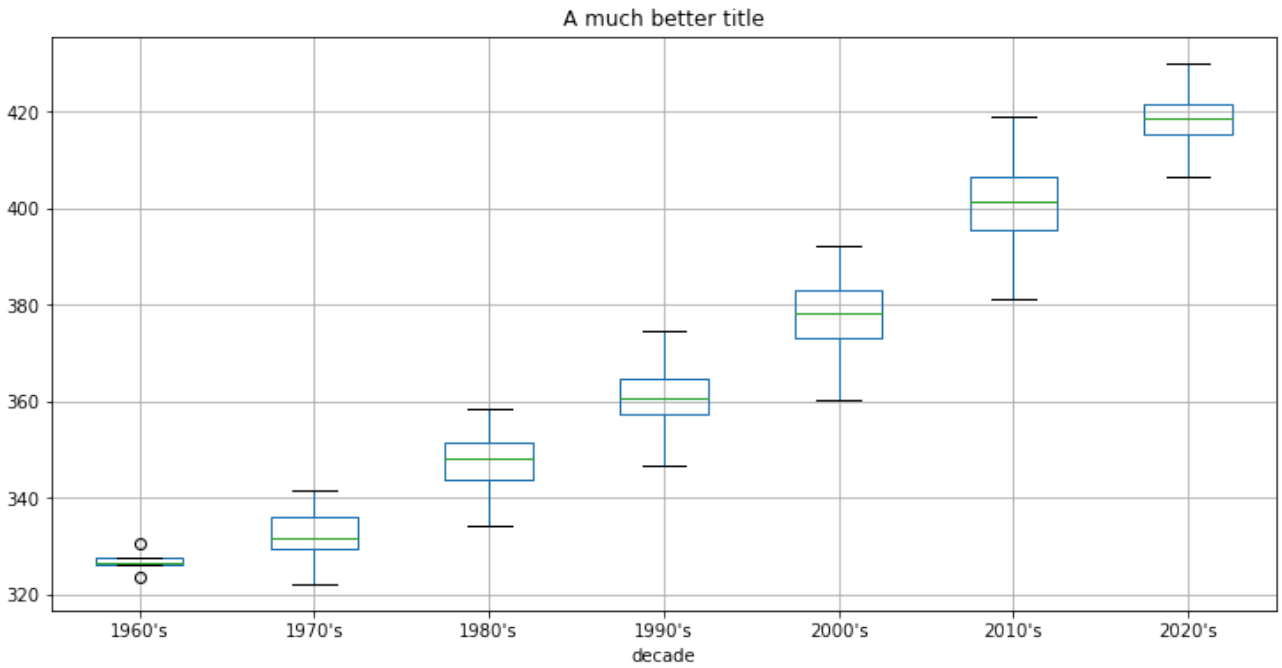
Out[48]: <AxesSubplot:title={'center':'CO2'}, xlabel='decade'>



Making really pretty boxplots in Pandas isn't always perfect. Here are a few hints to use our fig and ax nomenclature. Fixing the ticks is harder. You have to add a new label for each tick and then blanks for the ones you don't want. just some tricks to put in your back pocket. Don't spend much time here. Just file it away for future reference,

In [46]:
```
fig,ax=plt.subplots()
df_scripps.boxplot(column='CO2',by='decade',ax=ax)
fig.set_size_inches(12,6)
ax.set_title('A much better title')
ax.set_xticklabels(["1960's","1970's","1980's","1990's","2000's","2010's","2020's"])
fig.suptitle('')#This has to go last.
```

Out[46]: Text(0.5, 0.98, '')

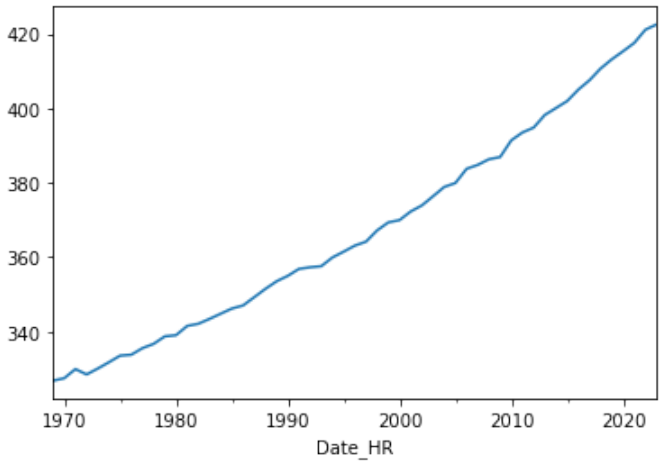


the next thing you might need to do is to resample by a larger time frame like a year. so what is the yearly mean? By resample we mean take all the data and find a subset of it. I am now just looking at the annual mean and plotting it all in one fell swoop. You need the np.mean b/c we are using an numpy function on our

data. Here is a link to show you the options. https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases

In [49]:
```
df_scripps.CO2.resample('A').mean().plot()
```

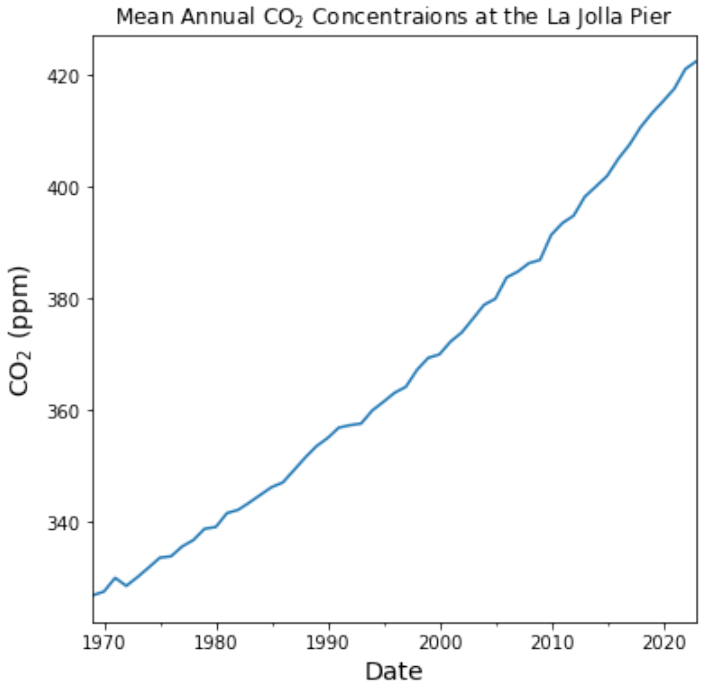
Out[49]:
```
<AxesSubplot:xlabel='Date_HR'>
```



Now here is how we do our fig,ax to make it look nice within pandas.

In [50]:
```
fig,ax=plt.subplots()
fig.set_size_inches(6,6)
df_scripps.CO2.resample('A').mean().plot(ax=ax)
ax.set_ylabel('CO$_2$ (ppm)',fontsize=14)
ax.set_xlabel('Date',fontsize=14)
ax.set_title('Mean Annual CO$_2$ Concentraions at the La Jolla Pier')
```

Out[50]:
```
Text(0.5, 1.0, 'Mean Annual CO$_2$ Concentraions at the La Jolla Pier')
```

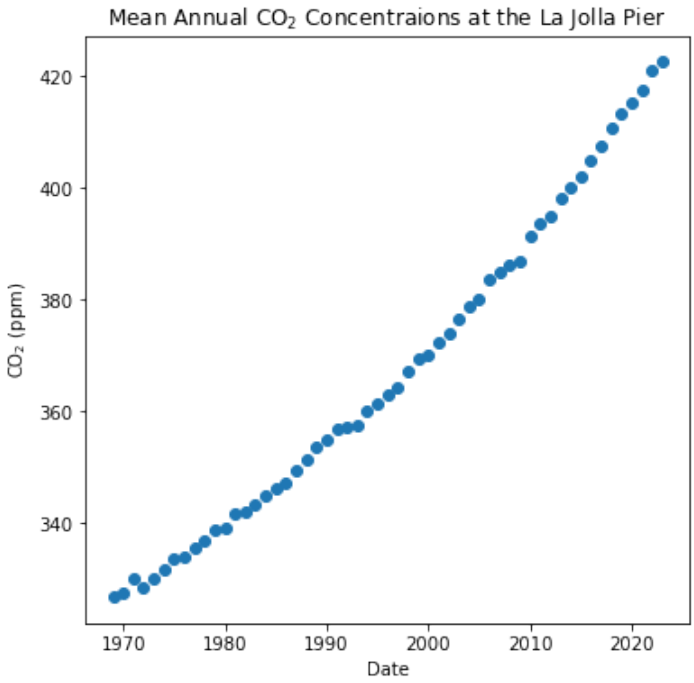


Here is how you would plot the data in scatter

In [51]:
```
fig,ax=plt.subplots()
fig.set_size_inches(6,6)
x=df_scripps.CO2.resample('A').mean().index.year
y=df_scripps.CO2.resample('A').mean()
```

```
ax.scatter(x,y)
ax.set_ylabel('CO$_2$ (ppm)')
ax.set_xlabel('Date')
ax.set_title('Mean Annual CO$_2$ Concentraions at the La Jolla Pier')
```

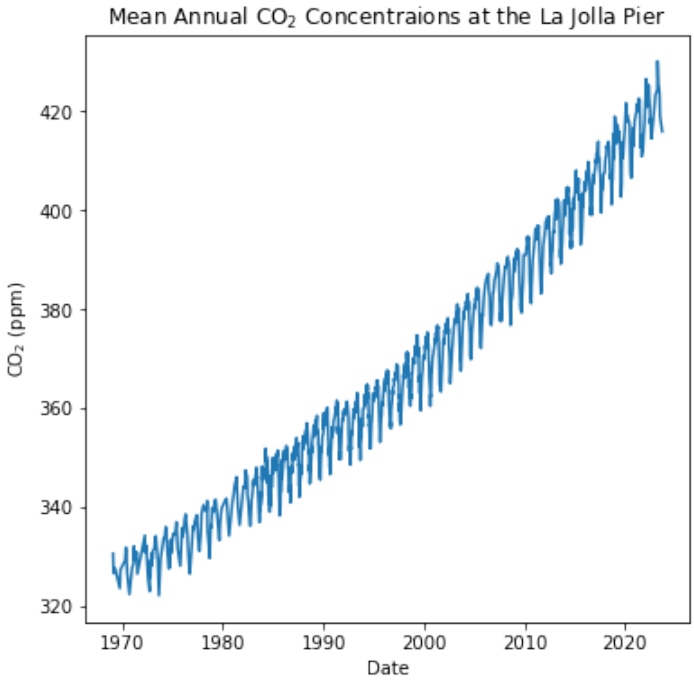
Out[51]: Text(0.5, 1.0, 'Mean Annual CO$_2$ Concentraions at the La Jolla Pier')



We could also plot all the data with plot outside of pandas.

In [52]:
```
fig,ax=plt.subplots()
fig.set_size_inches(6,6)
x=df_scripps.CO2.index
y=df_scripps.CO2
ax.plot(x,y)
ax.set_ylabel('CO$_2$ (ppm)')
ax.set_xlabel('Date')
ax.set_title('Mean Annual CO$_2$ Concentraions at the La Jolla Pier')
```

Out[52]: Text(0.5, 1.0, 'Mean Annual CO$_2$ Concentraions at the La Jolla Pier')



Now back to using Pandas. You can look online. There are so many ways to resample your data!

In [53]:
```
print( df_scripps.CO2.resample('Q').mean().head())  #THis just shows the quartely amounts!
#If there is an NAN it won't let me plot straight away.  not sure why.
# Just showing you there are more resample methods
```

```
Date_HR
1969-03-31    327.860
1969-06-30    326.810
1969-09-30        NaN
1969-12-31    325.345
1970-03-31        NaN
Freq: Q-DEC, Name: CO2, dtype: float64
```

For the how part you can use any numpy array function. I have not found a list. So you can try many.

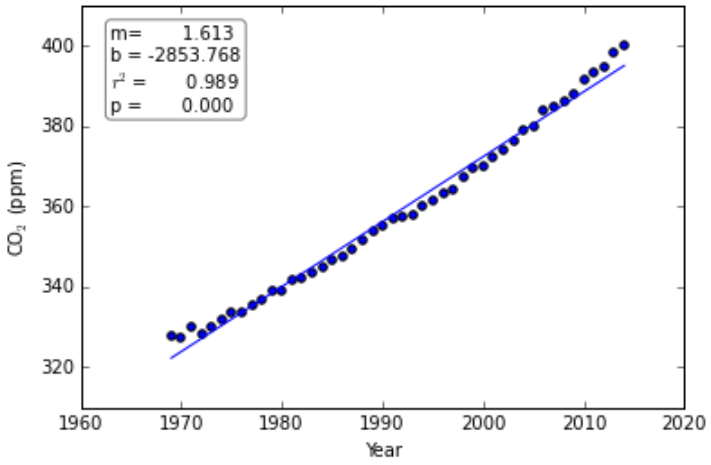
Linear Regression!

For now we will go back and use the annual mean values we did from resample and do linear regression

- Go back and set X and Y using resample for annual mean data.
- plot the x and y
- do linregress on the x and y
- make your graph look nice
- The intercept is non-sensical but do the other parameters make sense?

In [27]:
```
# Your code here!
```

Out[27]:
```
<matplotlib.text.Text at 0xf5b5358>
```

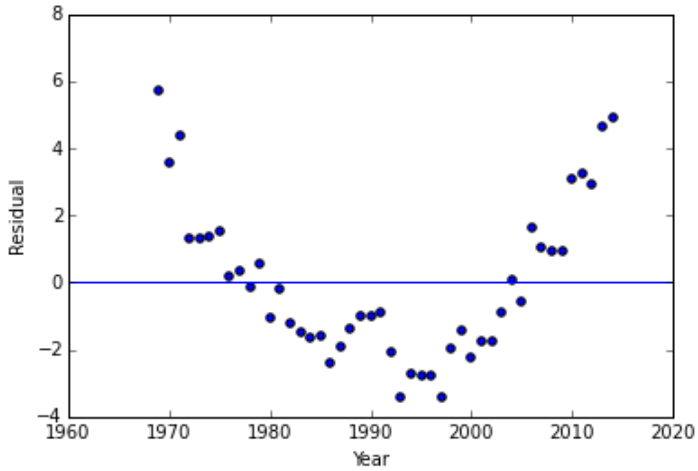


the r^2 is great but the curve looks non-linear. You can try to look at the graph "down the line" and you will see the line is straight but the points curve around it. So the error looks systematic and not random. could we find the residuals and plot them? So could we plot how far off the value is from the line for each year? So the residual is the difference in y-space for each value for each year. residuas=observed-predicted. I would make the residual by doing the math to look at the difference.

- Residual is Observed minus expected (observed-expected)
- expected=m*x+b from linregess
- so your resdiual beomces y-(mx+b)
- plot the residual versus x
- add the horizontal line by using ax.axhline(y=0)
- think about the data...

In [39]:
```
# Your code here!
```

Out[39]:
```
<matplotlib.lines.Line2D at 0x1082b0f0>
```



So linear ain't working!

We can use poly1d to fit a polynominal!

So lets figure out to poly1d and see how a second order equation fits the data! poly1d has lots of cool tricks!

Here is the description https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.poly1d.html

Remember for a polynominal 1st order y=ax+b

2nd order $y=ax^2+bx+c$

3rd order $y=ax^3+bx^2+cx+d$

etc...

How it works. Like linregress. You pass your x,y and now the order to poly1d

set fit_order to 2 for a second order polynominal fit_order=2

Now use polyfit

a=np.polyfit(x,y,fit_order)

Now it gets cool.

a is now a python class. You can pass that to np.poly1d(a)

polynominal=np.poly1d(a)

It returns the equation. I called it polynomial. But that equation will also do math for you. You give it an X and it gives you Y! So used linspace to make a bunch of x values! You want more than 2 now becuase your line will be curved.

x_fit=np.linspace(x.min(),x.max())

Now send your x_fit into polynominal and you have all your y's!

y_fit=polynominal(x_fit)

Now plot them!

I know that was all just weird. But that is python and object oriented code. It can do work for you.

If you are going to use textstr to make a string with the results of poly1d to show the equation it takes a trick or two. I did it two ways. First you could cast something to a string. So you could do

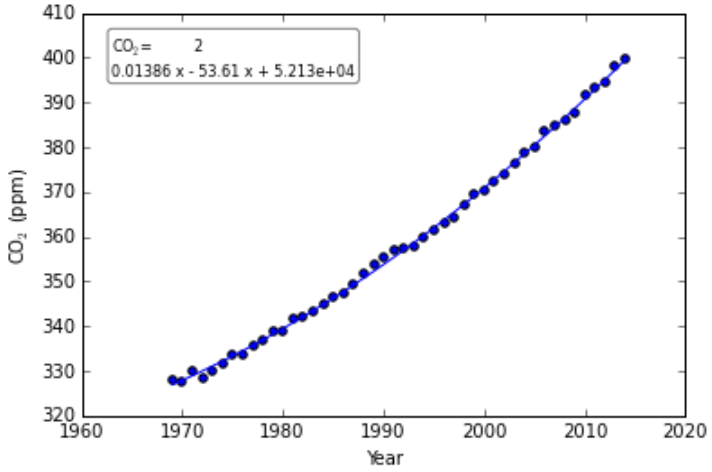
textstr='CO2={}'.format(poly1d(a))

Now for your y-axis If you want to subscript the 2 you can do the dollar sign trick for an equation with an underscore. I can't close the dollar signs as it will show the eqauation CO_2 so you would need to write CO$_2$

Double click on this box to see how I did the CO2

In [40]:
```
# Your code here!
```

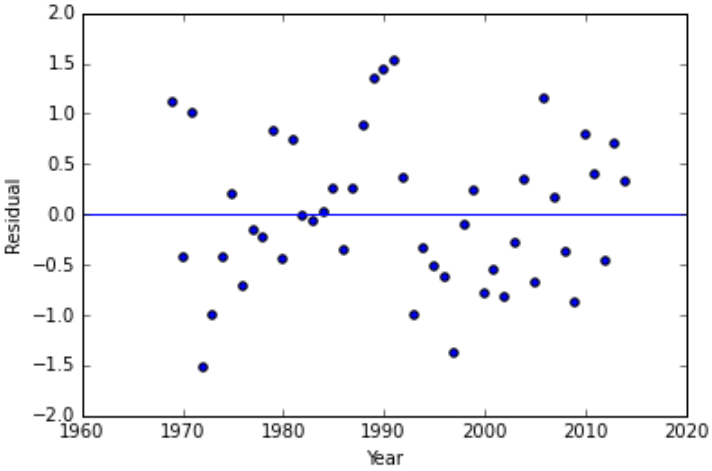
Out[40]:
```
<matplotlib.text.Text at 0x10870320>
```



Now that is a great fit!!!!!! We can also look at the residuals for it. Can you plot the residuals???

In [41]:
```
# Your code here!
```

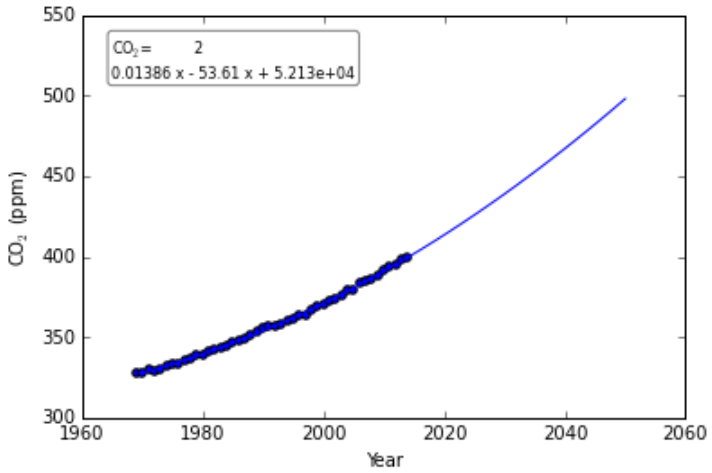
Out[41]: <matplotlib.lines.Line2D at 0x1085f4a8>



Now make a prediction for CO2 for 2016-2050! Can you extend the line to 2050?

In [47]:
```
# Your code here
```

Out[47]: <matplotlib.text.Text at 0x118087b8>

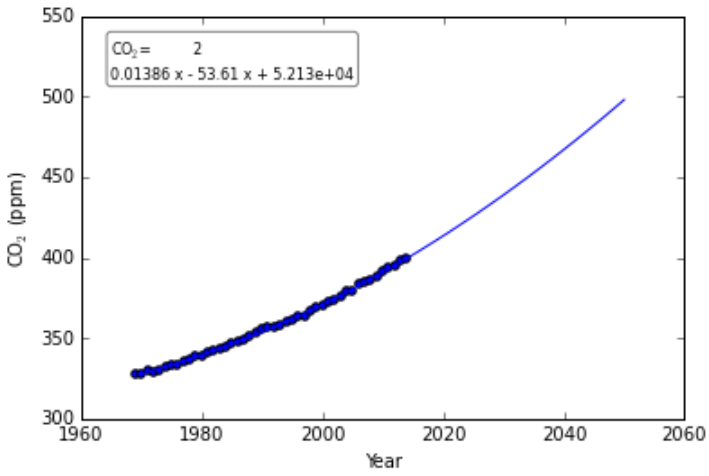


Comments!!!

I have been really bad about adding comments to the code I write. I try to write in markdown but you also want to add comments so people can follow the code. So lets take our last code and add comments to help us using the hastag! you will thank yourself in a few months from now. So comment your last code and I know you can do better than me!

In [54]:
```
#your code and comments here!
```

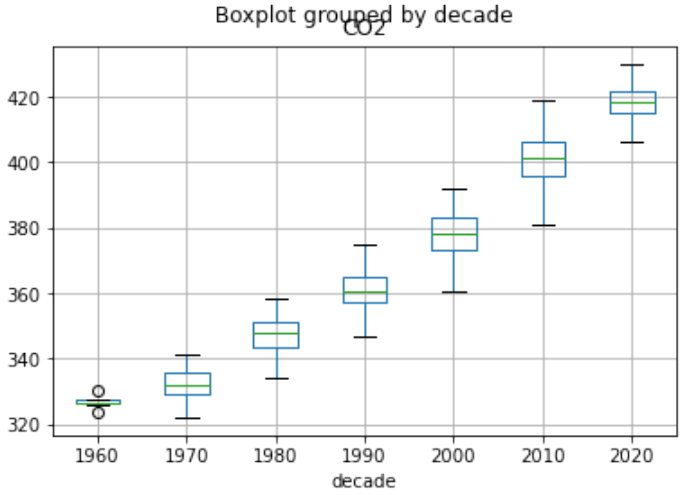
Out[54]:
```
<matplotlib.text.Text at 0x12282588>
```



Answers

In [47]:
```
fig,ax=plt.subplots()
df_scripps.boxplot(column='CO2',by='decade',ax=ax)
```

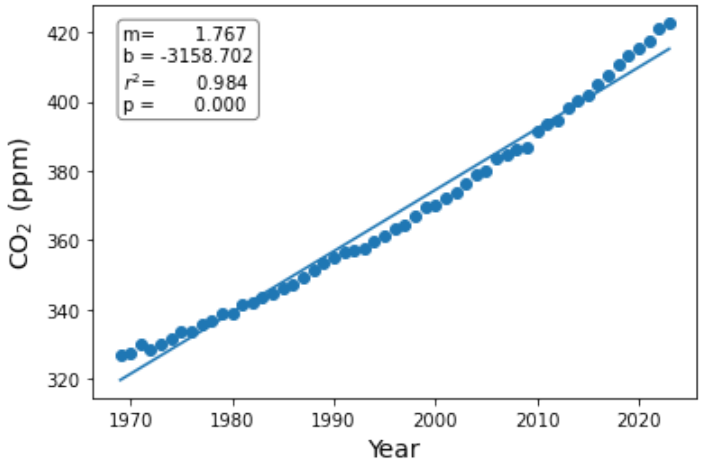
Out[47]:
```
<AxesSubplot:title={'center':'CO2'}, xlabel='decade'>
```



In [54]:
```
fig,ax=plt.subplots()
props=dict(boxstyle='round',facecolor='white',alpha=0.5)

x=df_scripps.CO2.resample('A').mean().index.year
y=df_scripps.CO2.resample('A').mean()
ax.scatter(x,y)

results=stats.linregress(x,y)
ax.plot(x,results[0]*x+results[1])
textstr='m={:>12.3f}\nb ={:>10.3f}\n$r^2$={:>12.3f}\np ={:>12.3f}'\
        .format(results[0],results[1],results[2]**2,results[3])
ax.text(0.05,0.95,textstr,transform=ax.transAxes,fontsize=10
        ,verticalalignment='top',bbox=props)
```

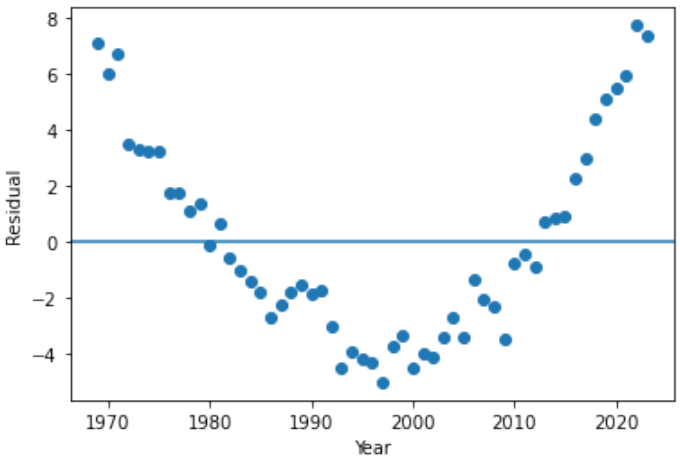
```
ax.set_ylabel('CO$_2$ (ppm)',fontsize=14)
ax.set_xlabel('Year',fontsize=14)
```

Out[54]: Text(0.5, 0, 'Year')



In [59]:
```
fig,ax=plt.subplots()

x=df_scripps.CO2.resample('A').mean().index.year
y=df_scripps.CO2.resample('A').mean()

results=stats.linregress(x,y)
ax.scatter(x,y-(results[0]*x+results[1]))
ax.set_ylabel('Residual')
ax.set_xlabel('Year')
ax.axhline(y=0)
```

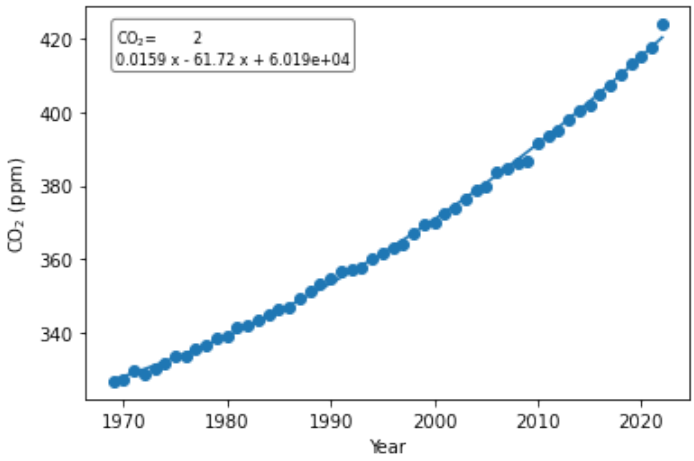
Out[59]: <matplotlib.lines.Line2D at 0x7fc660f59f10>



In [58]:
```
fit_order=2

fig,ax=plt.subplots()
props=dict(boxstyle='round',facecolor='white',alpha=0.5)

x=df_scripps.CO2.resample('A').mean().index.year
y=df_scripps.CO2.resample('A').mean()

x_fit=np.linspace(x.min(),x.max())  #I set the range using the data
a=np.polyfit(x,y,fit_order)  #I just copied from above to have in the same cell
polynominal=np.poly1d(a)
y_fit=polynominal(x_fit)

textstr='CO$_2$={}'.format(polynominal)
ax.text(0.05,0.95,textstr,transform=ax.transAxes,fontsize=8
```

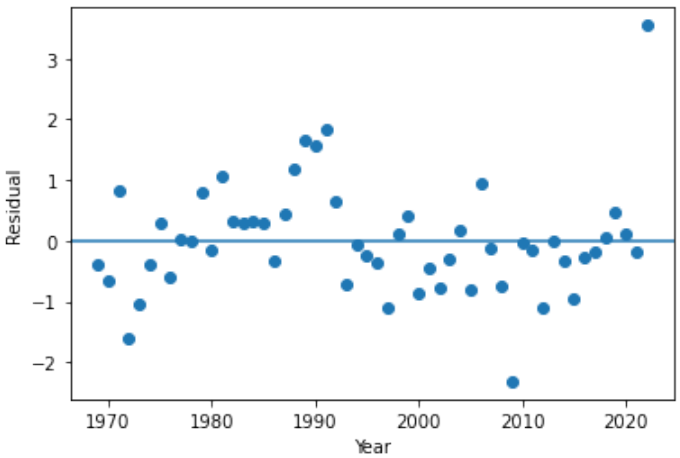
```
        ,verticalalignment='top',bbox=props)

ax.plot(x_fit,y_fit)
ax.scatter(x,y)
ax.set_ylabel('CO$_2$ (ppm)')
ax.set_xlabel('Year')
```

Out[58]: Text(0.5, 0, 'Year')



In [59]:
```
fit_order=2

fig,ax=plt.subplots()

x=df_scripps.CO2.resample('A').mean().index.year
y=df_scripps.CO2.resample('A').mean()

x_fit=np.linspace(x.min(),x.max())
a=np.polyfit(x,y,fit_order)
polynominal=np.poly1d(a)

ax.scatter(x,y-polynominal(x))

ax.set_ylabel('Residual')
ax.set_xlabel('Year')
ax.axhline(y=0)
```

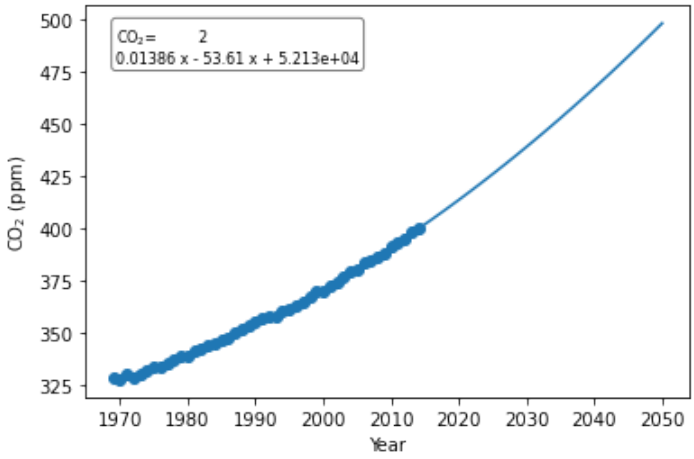
Out[59]: <matplotlib.lines.Line2D at 0x7fc42f9f7f70>



In [106…
```
fit_order=2

fig,ax=plt.subplots()
props=dict(boxstyle='round',facecolor='white',alpha=0.5)
```

```
x=df_scripps.CO2.resample('A').mean().index.year
y=df_scripps.CO2.resample('A').mean()

x_fit=np.linspace(1970,2050)
a=np.polyfit(x,y,fit_order)
polynominal=np.poly1d(a)
y_fit=polynominal(x_fit)

ax.plot(x_fit,y_fit)
ax.scatter(x,y)
ax.set_ylabel('CO$_2$ (ppm)')
ax.set_xlabel('Year')

textstr='CO$_2$={}'.format(np.poly1d(a))
ax.text(0.05,0.95,textstr,transform=ax.transAxes,fontsize=8
        ,verticalalignment='top',bbox=props)
```

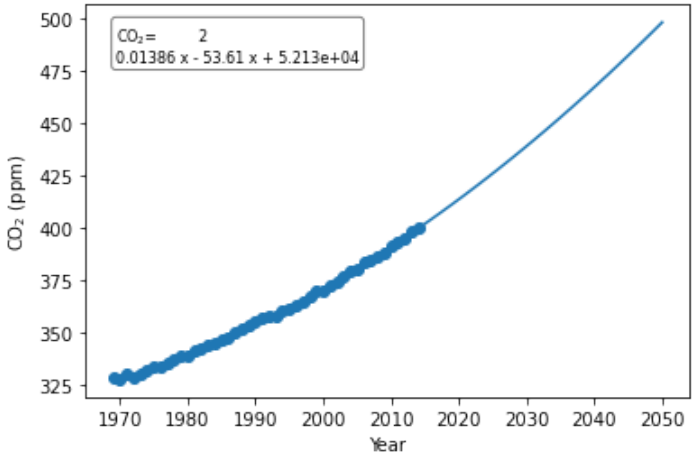
Out[106… `Text(0.05, 0.95, 'CO$_2$=         2\n0.01386 x - 53.61 x + 5.213e+04')`



In [105…

```
#get the x and y data from the larger data set for plotting and statistics.
x=df_scripps.CO2.resample('A').mean().index.year
y=df_scripps.CO2.resample('A').mean()

#i am going to fit the data using poly1d and use linspace to give us a bunch of points.
fit_order=2   #this is the order of the line.
x_fit=np.linspace(1970,2050)
a=np.polyfit(x,y,fit_order)
polynominal=np.poly1d(a)
y_fit=polynominal(x_fit)

#plot the data, the fit, label the axes and add the equation using a text box.
fig,ax=plt.subplots()
ax.plot(x_fit,y_fit)
ax.scatter(x,y)
ax.set_ylabel('CO$_2$ (ppm)')
ax.set_xlabel('Year')

props=dict(boxstyle='round',facecolor='white',alpha=0.5)
textstr='CO$_2$={}'.format(np.poly1d(a))
ax.text(0.05,0.95,textstr,transform=ax.transAxes,fontsize=8
        ,verticalalignment='top',bbox=props)
```

Out[105… `Text(0.05, 0.95, 'CO$_2$=         2\n0.01386 x - 53.61 x + 5.213e+04')`


CO_2 = 2
0.01386 x - 53.61 x + 5.213e+04
500
475
450
425
400
375
350
325
CO_2 (ppm)
1970
1980
1990
2000
2010
2020
2030
2040
2050
Year

In [ ]:

In [ ]:

In [ ]: