Synopsis

With Python there are a number of options for plotting available, including:

- using pandas built-in functions
- using matplotlib functions directly
- using other libraries e.g. seaborn, plotly

Matplotlib

`%matplotlib inline` is a **magic** command.
It means when plotting matplotlib charts, embed them directly into the notebook

Load libraries and import data

In [1]:
```
# load pandas
import pandas as pd

# include %matplotlib inline
%matplotlib inline

# Create a DataFrame called df_premiums from Insurance Premiums.csv
import pandas as pd
df_premiums = pd.read_csv('https://s3.eu-west-1.amazonaws.com/neueda.conygre.com/pyda
```

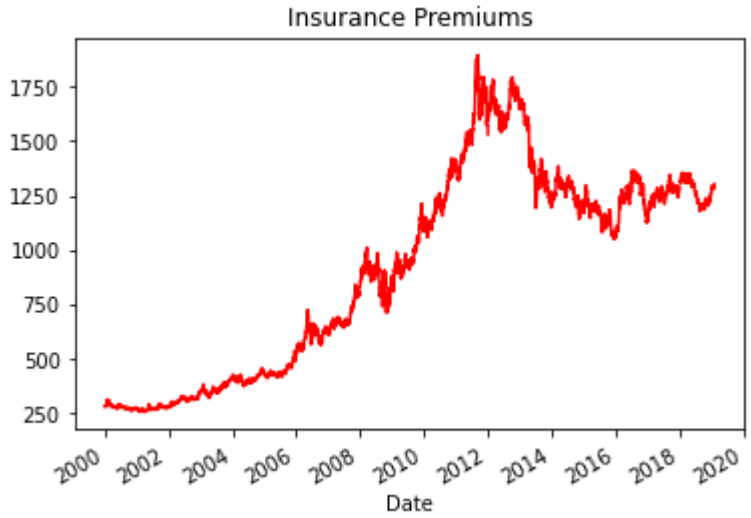
A quick plot of two columns, this uses matplotlib through pandas

In [2]:
```
# line plot of 'USD (AM)' and 'GBP (AM)' columns
ax = df_premiums[ ['USD (AM)', 'GBP (AM)'] ].plot()
```

Be more specific, plot a single column in red, add a title

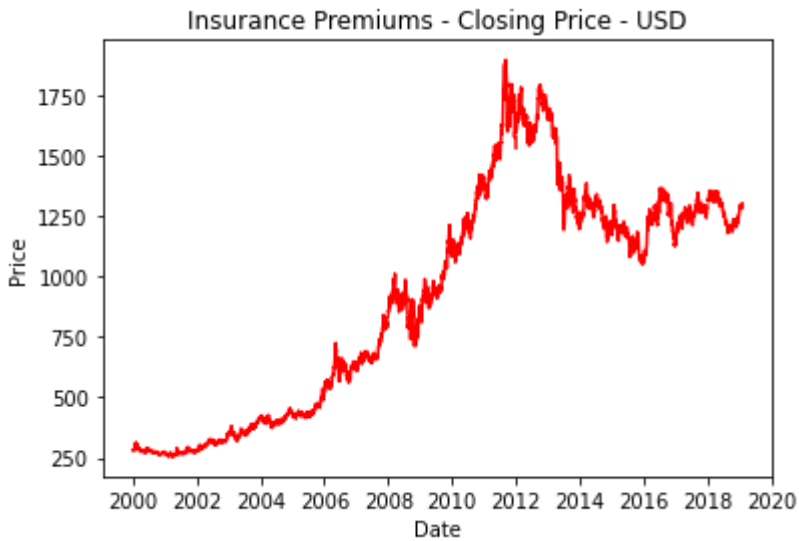
In [3]:
```
# some extra parameters - USD (AM) line, color and title
df_premiums['USD (PM)'].plot(kind='line', color='r', title='Insurance Premiums')
```

Out[3]: <AxesSubplot:title={'center':'Insurance Premiums'}, xlabel='Date'>

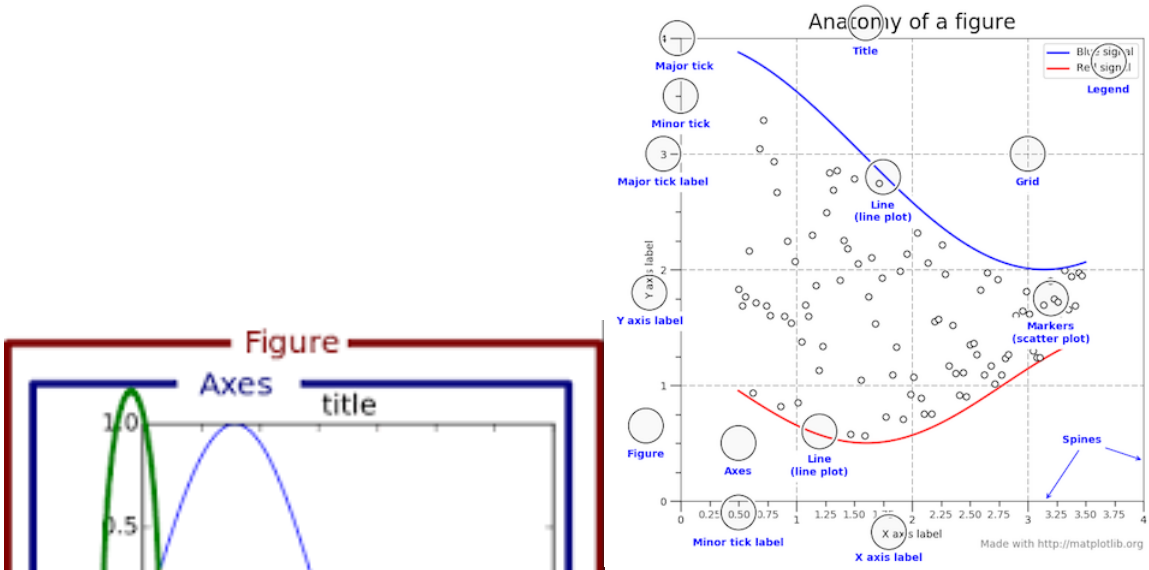


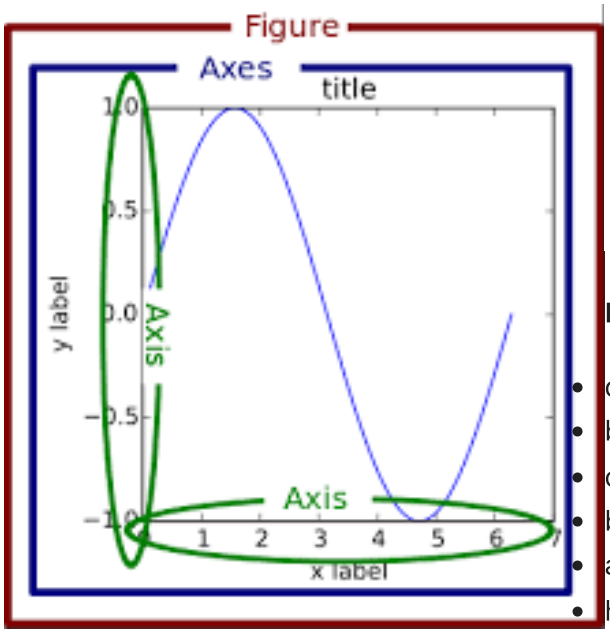
For more fine-grained control we can call matplotlib directly

In [4]:
```
import matplotlib.pyplot as plt

# create same plot as above, but this time directly through matplotlib
x = df_premiums.index
y = df_premiums['USD (PM)']

plt.plot(x, y, 'r') # 'r' is the color red
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Insurance Premiums - Closing Price - USD')

plt.show()
```



Anatomy of a figure





NOTE

- difference between `axes` and `axis`
- both pronounced the same
- one is singlular, one is plural
- both refer to different parts of a figure
- axes contains mulitple axis
- here axes contains and x-axis and a y-axis

Change style and plot multiple axes on a single figure

See styles at matplotlib styles

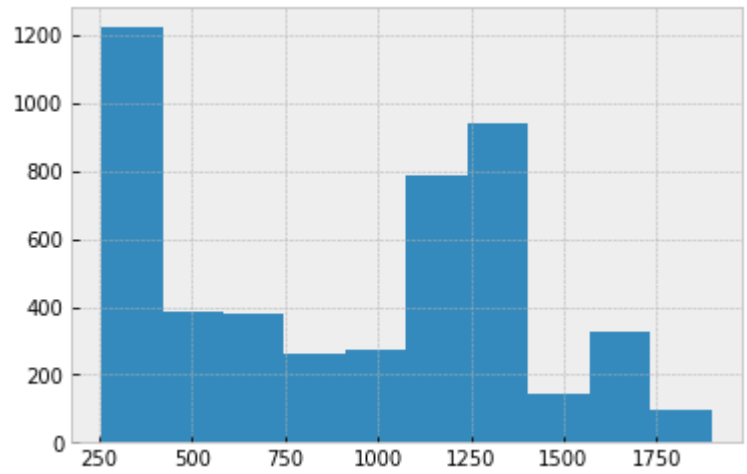
In [5]:
```
plt.style.use('bmh')

fig = plt.figure(figsize=(14, 6))

# left, bottom, width, height
axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])
axes2 = fig.add_axes([0.18, 0.6, 0.3, 0.3])

x = df_premiums.index.date
y1 = df_premiums['GBP (AM)']

axes1.plot(x, y1, 'g')
axes2.plot(df_premiums['GBP (AM)']['Aug 2008'], 'b')

plt.xticks(rotation=45)

plt.show()
```

In [6]:
```
# save this figure
fig.savefig('example.png')
```

Plot a histogram through pandas (uses matplotlib under the hood)

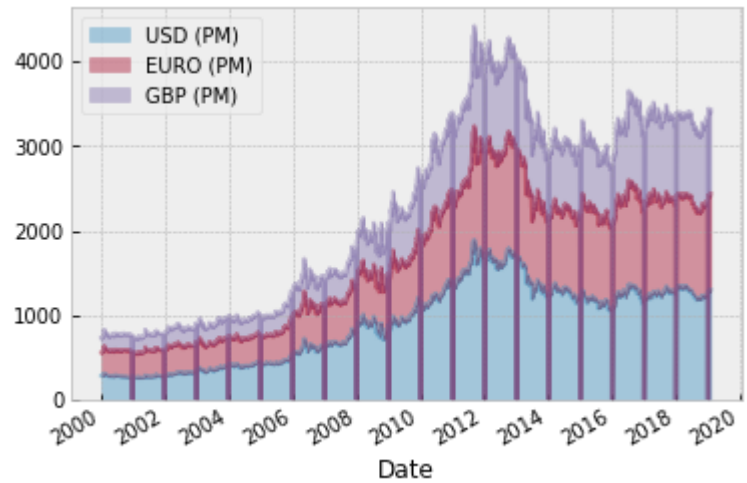
In [7]:
```
# 'USD (AM)' as a histogram?
df_premiums['USD (AM)'].hist()
```

Out[7]:
```
<AxesSubplot:>
```



An area chart for 3 columns

In [8]:
```
my_plot = df_premiums[ ['USD (PM)', 'EURO (PM)', 'GBP (PM)'] ].plot.area(alpha=0.4)
```



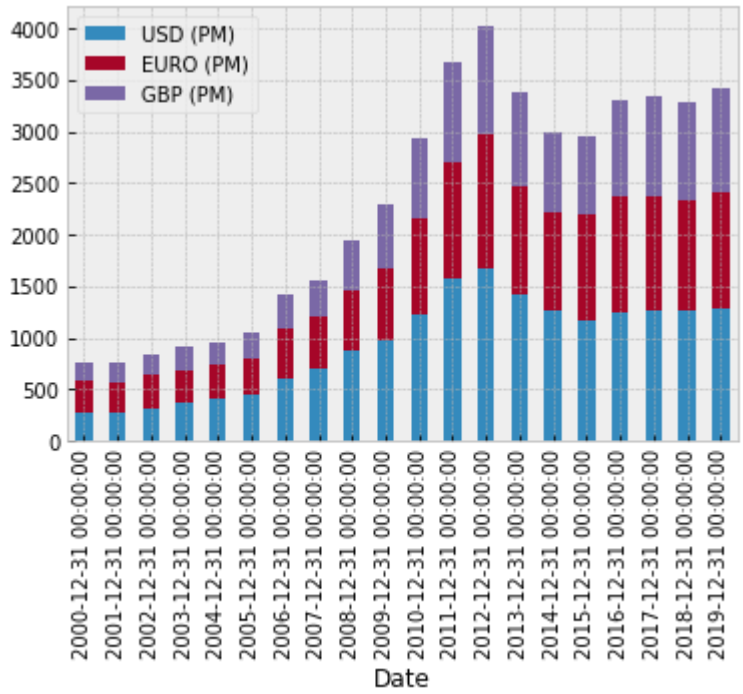
We can save plots to file

In [9]:
```
# save to file
my_plot.get_figure().savefig('out.jpg')
```

Stacked Bar Chart

In [10]:
```
df_premiums[ ['USD (PM)', 'EURO (PM)', 'GBP (PM)'] ].resample(rule='Y').mean().plot.b
```

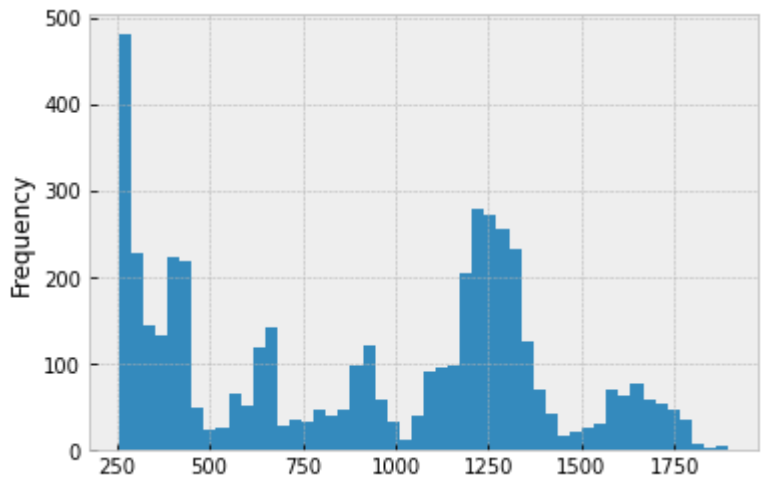
Out[10]:
```
<AxesSubplot:xlabel='Date'>
```



Histograms

In [11]:
```
plt.style.use('bmh')
df_premiums['USD (PM)'].plot.hist(bins=50)
```

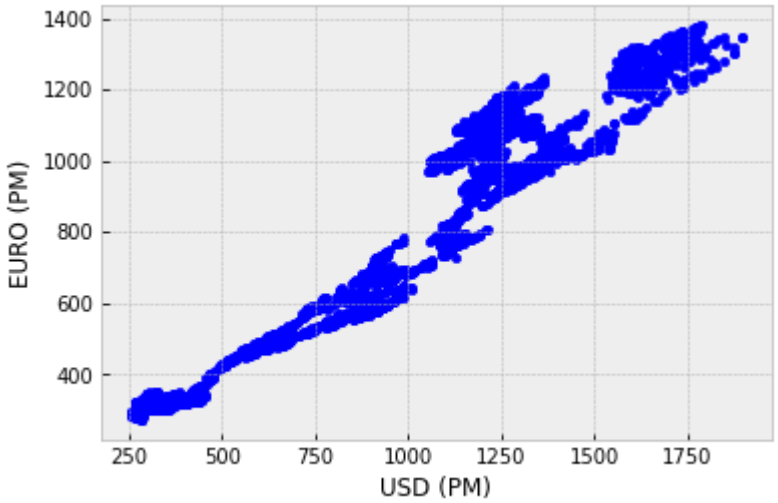
Out[11]:
```
<AxesSubplot:ylabel='Frequency'>
```



Scatter Plots

In [12]:
```
df_premiums.plot.scatter(x='USD (PM)', y='EURO (PM)')
```

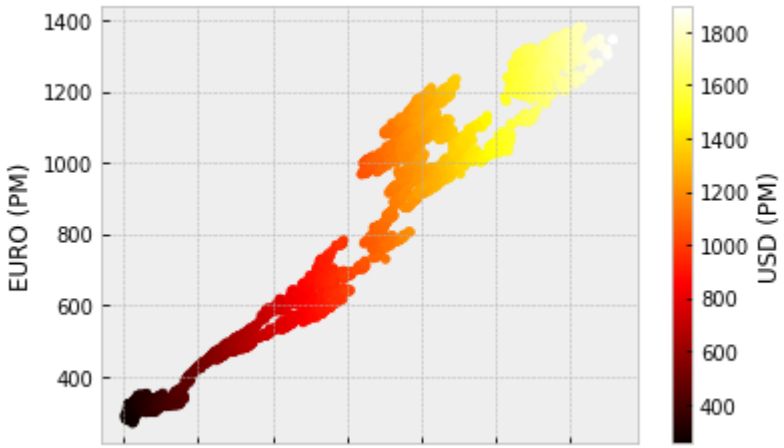
Out[12]:
```
<AxesSubplot:xlabel='USD (PM)', ylabel='EURO (PM)'>
```



You can use c to color based off another column value Use cmap to indicate colormap to use. For all the colormaps, check out: http://matplotlib.org/users/colormaps.html

In [13]:
```
df_premiums.plot.scatter(x='USD (PM)', y='EURO (PM)', c='USD (PM)', cmap='hot')
```

Out[13]:
```
<AxesSubplot:xlabel='USD (PM)', ylabel='EURO (PM)'>
```

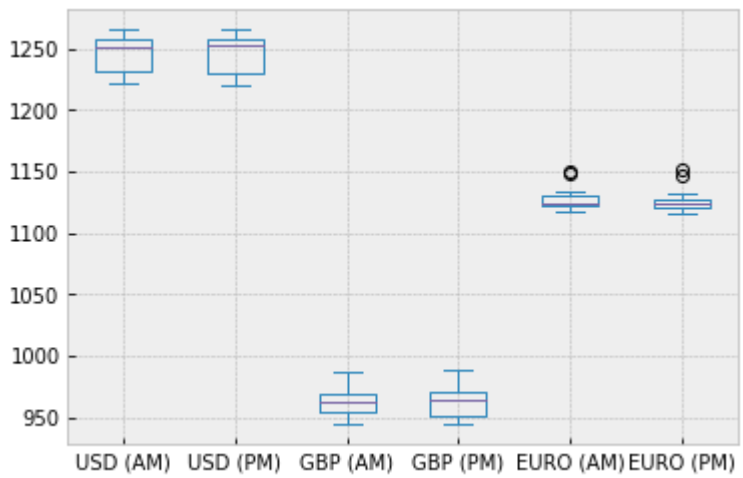


BoxPlots

In [14]:
```
df_premiums['2017-MAY'].plot.box() # Can also pass a "by=..." argument for groupby
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: In
dexing a DataFrame with a datetimelike index using a single string to slice the rows,
like `frame[string]`, is deprecated and will be removed in a future version. Use `fra
me.loc[string]` instead.
  """Entry point for launching an IPython kernel.
```

Out[14]:
```
<AxesSubplot:>
```

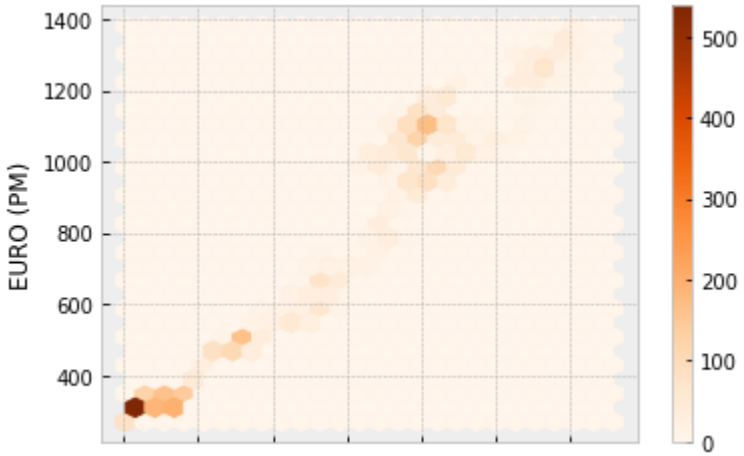


Hexagonal Bin Plot

Useful for Bivariate Data, alternative to scatterplot:

In [15]:
```
df_premiums.plot.hexbin(x='USD (PM)', y='EURO (PM)', gridsize=25, cmap='Oranges')
```

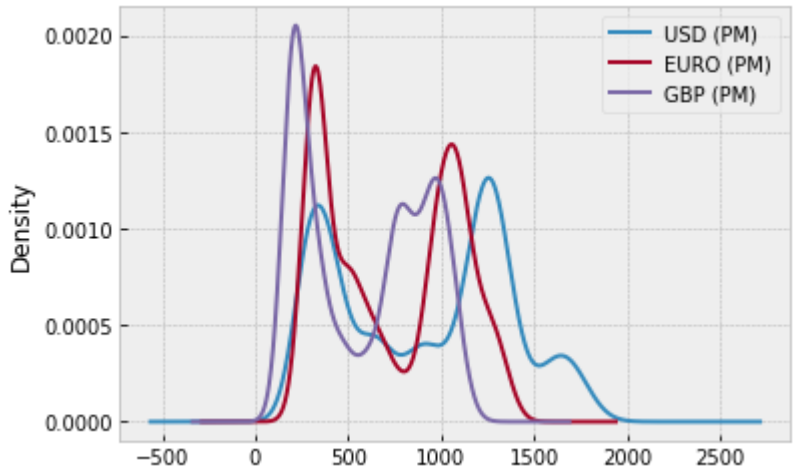
Out[15]:
```
<AxesSubplot:xlabel='USD (PM)', ylabel='EURO (PM)'>
```



Kernel Density Estimation plot (KDE)

In [16]:
```
df_premiums[ ['USD (PM)', 'EURO (PM)', 'GBP (PM)'] ].plot.density()
```

Out[16]:
```
<AxesSubplot:ylabel='Density'>
```



Styling the display of a DataFrame can also be a very useful visualisation tool

In [17]:
```
# highlight max and min in df_premiums.head()
df_premiums.head(10).style.highlight_max(color='lightgreen').highlight_min(color='red
```

Out[17]:

Date	USD (AM)	USD (PM)	GBP (AM)	GBP (PM)	EURO (AM)	EURO (PM)
2000-01-04 00:00:00	282.050000	281.500000	172.166000	171.929000	275.305000	272.402000
2000-01-05 00:00:00	282.100000	280.450000	171.729000	170.808000	272.035000	270.470000
2000-01-06 00:00:00	280.350000	279.400000	170.446000	169.518000	270.974000	269.152000
2000-01-07 00:00:00	282.000000	282.100000	171.324000	172.065000	273.840000	274.017000
2000-01-10 00:00:00	281.700000	281.600000	172.230000	171.959000	275.205000	274.812000
2000-01-11 00:00:00	281.700000	282.250000	171.590000	171.424000	273.469000	274.322000
2000-01-12 00:00:00	282.250000	282.250000	171.445000	171.372000	273.313000	274.269000
2000-01-13 00:00:00	282.200000	282.100000	171.467000	171.594000	273.874000	274.898000
2000-01-14 00:00:00	284.150000	283.300000	172.683000	172.439000	277.274000	277.881000
2000-01-17 00:00:00	284.900000	285.350000	174.112000	174.890000	281.577000	282.777000

In [18]:
```
# set a background gradient
df_premiums.head(10).style.background_gradient(cmap='Blues')
```

Out[18]:

Date	USD (AM)	USD (PM)	GBP (AM)	GBP (PM)	EURO (AM)	EURO (PM)
2000-01-04 00:00:00	282.050000	281.500000	172.166000	171.929000	275.305000	272.402000
2000-01-05 00:00:00	282.100000	280.450000	171.729000	170.808000	272.035000	270.470000
2000-01-06 00:00:00	280.350000	279.400000	170.446000	169.518000	270.974000	269.152000
2000-01-07 00:00:00	282.000000	282.100000	171.324000	172.065000	273.840000	274.017000
2000-01-10 00:00:00	281.700000	281.600000	172.230000	171.959000	275.205000	274.812000
2000-01-11 00:00:00	281.700000	282.250000	171.590000	171.424000	273.469000	274.322000
2000-01-12 00:00:00	282.250000	282.250000	171.445000	171.372000	273.313000	274.269000
2000-01-13 00:00:00	282.200000	282.100000	171.467000	171.594000	273.874000	274.898000
2000-01-14 00:00:00	284.150000	283.300000	172.683000	172.439000	277.274000	277.881000
2000-01-17 00:00:00	284.900000	285.350000	174.112000	174.890000	281.577000	282.777000

There are a number of other plotting libraries in python, e.g. seaborn and plotly

This is an example using the seaborn plotting library

In [19]:
```
# import seaborn
import seaborn as sns
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tools\_testing.py:19: FutureWa
rning: pandas.util.testing is deprecated. Use the functions in the public API at pand
as.testing instead.
  import pandas.util.testing as tm
```

In [20]:
```
# use a seaborn lineplot for comparison
sns.set(style="whitegrid")
ax = sns.lineplot(data=df_premiums['USD (AM)'])
```


1750
1500
1250
1000
750
500
250
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020