# Data Analysis with Python

## Pandas, Jupyter, and Friends

Andreas Herten, 4 May 2017

*»The data analyst's three foundations in Python«*

Matplotlib • Pandas • Jupyter Notebook

# Matplotlib

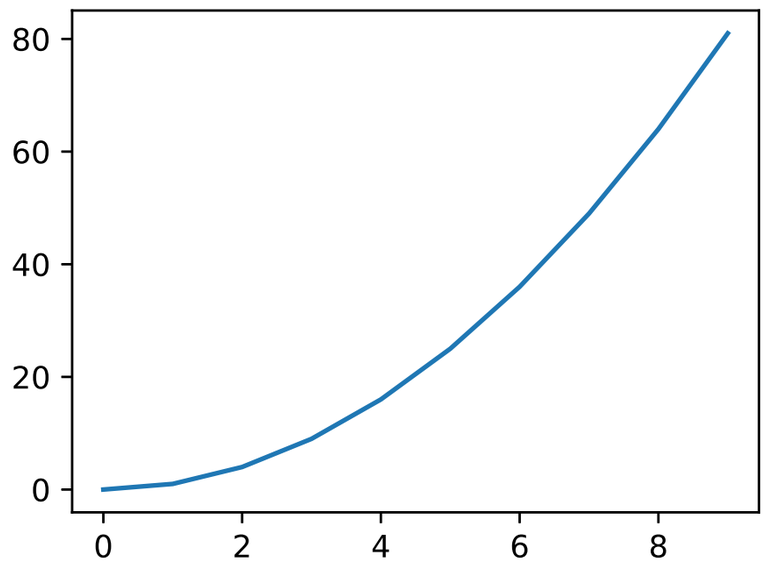
- Standard for plotting with Python
- Recently v. 2.0.0 released
- → https://matplotlib.org/index.html

## Using the *global API*

- Using the MATLAB-like interface
- Everything works through `plt...`

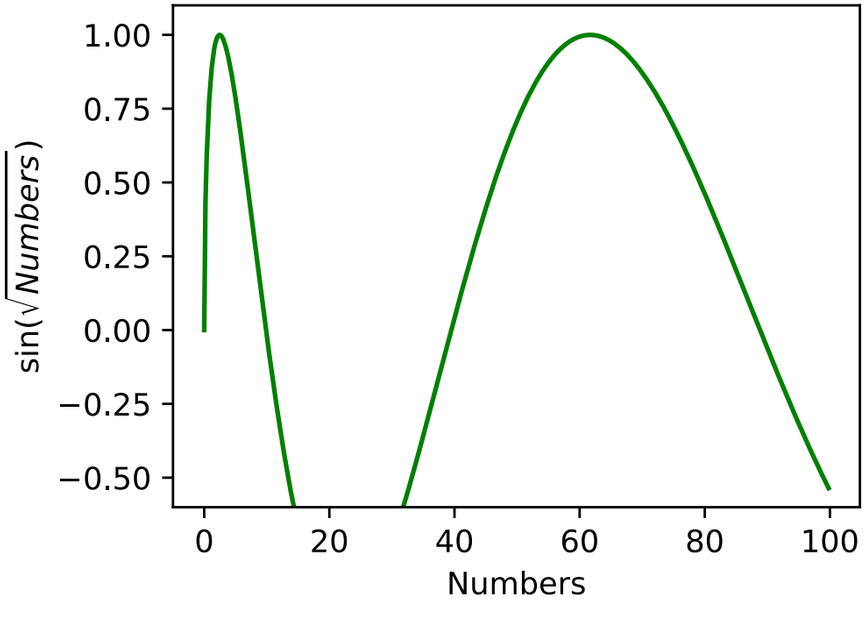
In [1]:
```
import matplotlib.pyplot as plt
x = range(10)
y = [i**2 for i in range(10)]
```

In [3]:
```
plt.plot(x, y)
plt.show()
```



## Option Showcase

In [4]:
```
import numpy as np
x = np.arange(0, 100, 0.2)
y = np.sin(np.sqrt(x))
plt.plot(x, y, color="green")
plt.ylim([-0.6,1.1])
plt.xlabel("Numbers")
plt.ylabel("$\sin(\sqrt{Numbers})$")
plt.show()
```



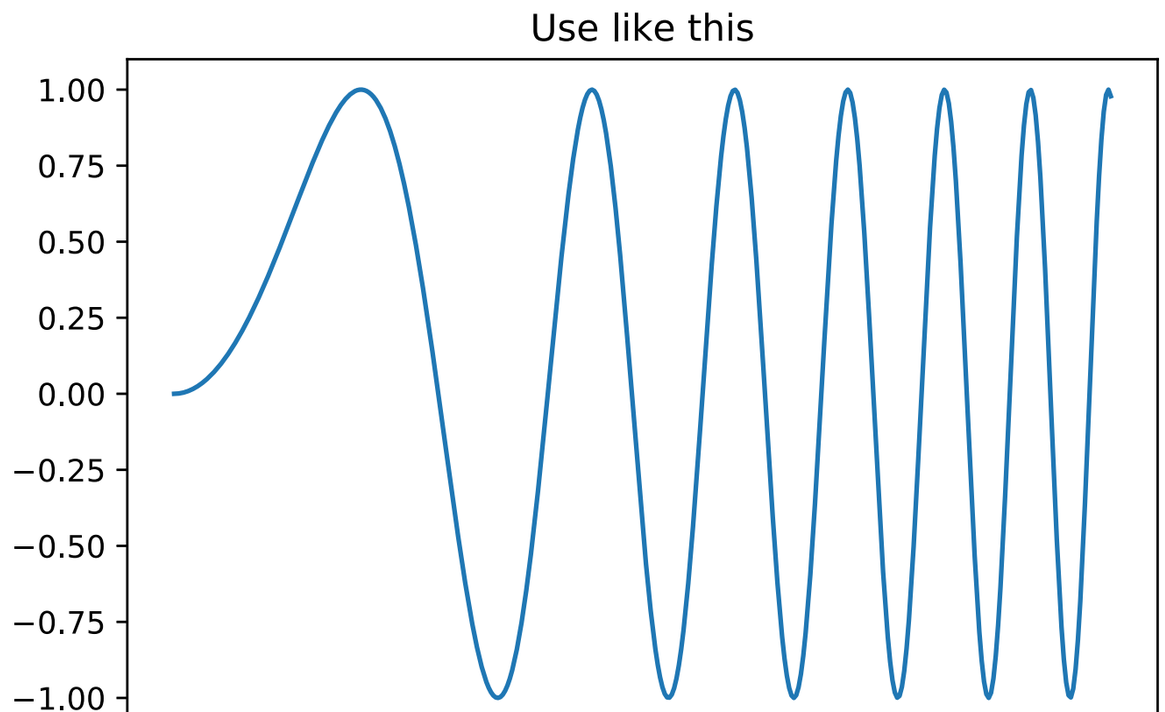
## Object API

- Instead of operation on *global objects* with `plt`, rather use `Figure` and `Axis` (axes ≈ plots)
- Cleaner approach (*IMHO*)
- Used under the hood of *global API* by leveraging `plt.gca()...` (*get current axis*)

In [5]:
```
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)
```

In [7]:
```
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_title('Use like this')
ax.set_xlabel("Numbers again")
```

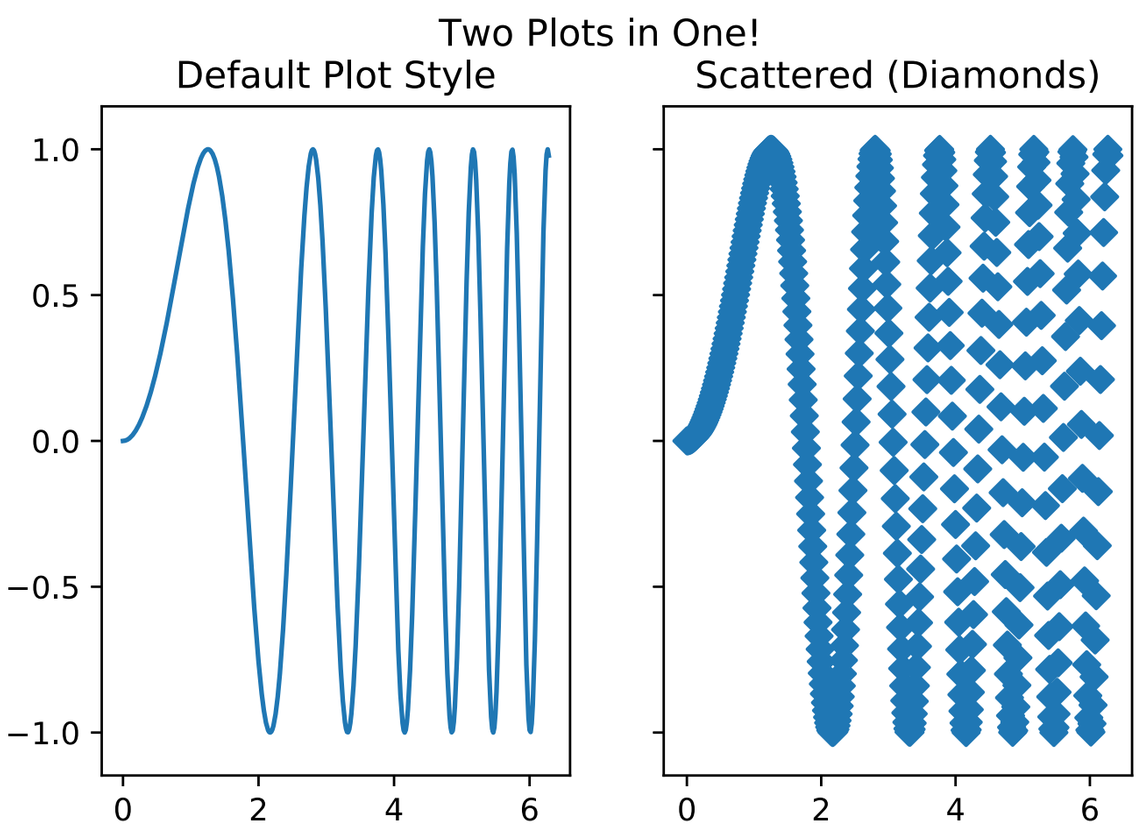
Out[7]: <matplotlib.text.Text at 0x112c8fb38>



## Multiple Plots

In [8]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, sharey=True)
ax1.plot(x, y)
ax1.set_title('Default Plot Style')
ax2.scatter(x, y, marker="D")
ax2.set_title('Scattered (Diamonds)')
fig.suptitle("Two Plots in One!")
```

Out[8]:
```
<matplotlib.text.Text at 0x112dddf60>
```



# Pandas

Introduction

## Introduction

> *pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.*

- Most important feature: `DataFrame`s and operations with them
- → http://pandas.pydata.org/

In [9]:
```
import pandas as pd
```

## Creating a DataFrame

Using a dictionary as an input

In [10]:
```
frame = pd.DataFrame({
    "A": 1.2,
    "B": pd.Timestamp('20170503'),
    "C": [(-1)**i * np.sqrt(i) + np.e * (-1)**(i-1) for i in range(5)],
    "D": pd.Categorical(["This", "column", "has", "entries", "entries"]),
    "E": "Same"
})
frame
```

Out[10]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **0** | 1.2 | 2017-05-03 | -2.718282 | This | Same |
| **1** | 1.2 | 2017-05-03 | 1.718282 | column | Same |
| **2** | 1.2 | 2017-05-03 | -1.304068 | has | Same |
| **3** | 1.2 | 2017-05-03 | 0.986231 | entries | Same |
| **4** | 1.2 | 2017-05-03 | -0.718282 | entries | Same |

Also available: `.read_csv` and `.read_excel`

## Popular Functions on Frames

In [11]:
```
frame.describe()
```

Out[11]:

| | A | C |
|---|---|---|
| **count** | 5.0 | 5.000000 |
| **mean** | 1.2 | -0.407224 |
| **std** | 0.0 | 1.781963 |
| **min** | 1.2 | -2.718282 |
| **25%** | 1.2 | -1.304068 |
| **50%** | 1.2 | -0.718282 |
| **75%** | 1.2 | 0.986231 |
| **max** | 1.2 | 1.718282 |

In [12]:
```
frame.head(2)
```

Out[12]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **0** | 1.2 | 2017-05-03 | -2.718282 | This | Same |
| **1** | 1.2 | 2017-05-03 | 1.718282 | column | Same |

## Popular Functions on Frames II

In [13]:
```
frame.transpose()
```

Out[13]:

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **A** | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 |
| **B** | 2017-05-03 00:00:00 | 2017-05-03 00:00:00 | 2017-05-03 00:00:00 | 2017-05-03 00:00:00 | 2017-05-03 00:00:00 |
| **C** | -2.71828 | 1.71828 | -1.30407 | 0.986231 | -0.718282 |
| **D** | This | column | has | entries | entries |
| **E** | Same | Same | Same | Same | Same |

In [14]:
```
frame.sort_values("C")
```

Out[14]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **0** | 1.2 | 2017-05-03 | -2.718282 | This | Same |
| **2** | 1.2 | 2017-05-03 | -1.304068 | has | Same |
| **4** | 1.2 | 2017-05-03 | -0.718282 | entries | Same |
| **3** | 1.2 | 2017-05-03 | 0.986231 | entries | Same |
| **1** | 1.2 | 2017-05-03 | 1.718282 | column | Same |

## Popular Functions on Frames III

In [15]:
```
round(frame,2)
frame.round(2)
```

Out[15]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **0** | 1.2 | 2017-05-03 | -2.72 | This | Same |
| **1** | 1.2 | 2017-05-03 | 1.72 | column | Same |
| **2** | 1.2 | 2017-05-03 | -1.30 | has | Same |
| **3** | 1.2 | 2017-05-03 | 0.99 | entries | Same |
| **4** | 1.2 | 2017-05-03 | -0.72 | entries | Same |

In [16]:
```
frame.sum()
```

Out[16]:
```
A    6.000000
C   -2.036119
dtype: float64
```

In [17]:
```
frame.round(2).sum()
```

Out[17]:
```
A    6.00
C   -2.03
dtype: float64
```

## Popular Functions on Frames IV

In [18]:
```
print(frame.round(2).to_latex())
```

```
\begin{tabular}{lrlrll}
\toprule
{} &    A &          B &     C &        D &     E \\
\midrule
0 &  1.2 & 2017-05-03 & -2.72 &     This &  Same \\
1 &  1.2 & 2017-05-03 &  1.72 &   column &  Same \\
2 &  1.2 & 2017-05-03 & -1.30 &      has &  Same \\
3 &  1.2 & 2017-05-03 &  0.99 &  entries &  Same \\
4 &  1.2 & 2017-05-03 & -0.72 &  entries &  Same \\
\bottomrule
\end{tabular}
```

## Index, Columns

In [19]:
```
frame["NewIdx"] = pd.date_range('20170504', periods=5)
frame.head(3)
```

Out[19]:

| | A | B | C | D | E | NewIdx |
|---|---|---|---|---|---|---|
| **0** | 1.2 | 2017-05-03 | -2.718282 | This | Same | 2017-05-04 |
| **1** | 1.2 | 2017-05-03 | 1.718282 | column | Same | 2017-05-05 |
| **2** | 1.2 | 2017-05-03 | -1.304068 | has | Same | 2017-05-06 |

## Index, Columns II

In [20]:
```
frame = frame.set_index("NewIdx")  # Also: inplace=True
frame.head(3)
```

Out[20]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **NewIdx** | | | | | |
| **2017-05-04** | 1.2 | 2017-05-03 | -2.718282 | This | Same |
| **2017-05-05** | 1.2 | 2017-05-03 | 1.718282 | column | Same |
| **2017-05-06** | 1.2 | 2017-05-03 | -1.304068 | has | Same |

In [21]:
```
frame.index
```

Out[21]:
```
DatetimeIndex(['2017-05-04', '2017-05-05', '2017-05-06', '2017-05-07',
               '2017-05-08'],
              dtype='datetime64[ns]', name='NewIdx', freq=None)
```

In [22]:
```
frame.columns
```

Out[22]:
```
Index(['A', 'B', 'C', 'D', 'E'], dtype='object')
```

## Slicing

Select only column "A"

In [23]:
```
frame["A"]
```

Out[23]:
```
NewIdx
2017-05-04    1.2
2017-05-05    1.2
2017-05-06    1.2
2017-05-07    1.2
2017-05-08    1.2
Name: A, dtype: float64
```

Select columns "A" and "C"

In [24]:
```
frame[["A", "C"]].sort_values("C")
```

Out[24]:

| | A | C |
|---|---|---|
| **NewIdx** | | |
| **2017-05-04** | 1.2 | -2.718282 |
| **2017-05-06** | 1.2 | -1.304068 |
| **2017-05-08** | 1.2 | -0.718282 |
| **2017-05-07** | 1.2 | 0.986231 |

## Slicing II

In [25]:
```
frame[1:3]
```

Out[25]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **NewIdx** | | | | | |
| **2017-05-05** | 1.2 | 2017-05-03 | 1.718282 | column | Same |
| **2017-05-06** | 1.2 | 2017-05-03 | -1.304068 | has | Same |

In [26]:
```
frame.loc["2017-05-06"]
```

Out[26]:
```
A                    1.2
B    2017-05-03 00:00:00
C               -1.30407
D                    has
E                   Same
Name: 2017-05-06 00:00:00, dtype: object
```

In [27]:
```
frame.iloc[2]
```

Out[27]:
```
A                    1.2
B    2017-05-03 00:00:00
C               -1.30407
D                    has
E                   Same
Name: 2017-05-06 00:00:00, dtype: object
```

## Slicing III

In [28]:
```
frame[frame["C"] > 0]
```

Out[28]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **NewIdx** | | | | | |
| **2017-05-05** | 1.2 | 2017-05-03 | 1.718282 | column | Same |
| **2017-05-07** | 1.2 | 2017-05-03 | 0.986231 | entries | Same |

In [29]:
```
frame[(frame["C"] > 0) & (frame["D"] == "has")]
```

Out[29]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **NewIdx** | | | | | |

## Plotting

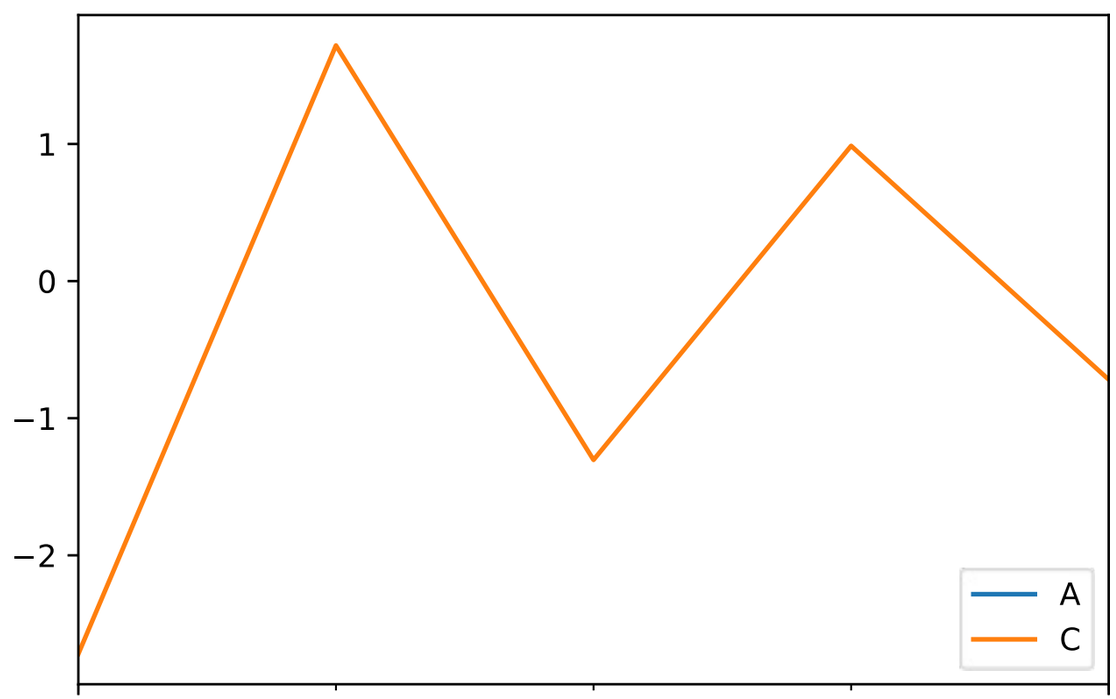
In [30]:
```
frame[["A", "C"]].head(3)
```

Out[30]:

| | A | C |
|---|---|---|
| **NewIdx** | | |
| **2017-05-04** | 1.2 | -2.718282 |
| **2017-05-05** | 1.2 | 1.718282 |
| **2017-05-06** | 1.2 | -1.304068 |

In [31]:
```
frame[["A", "C"]].plot()
```

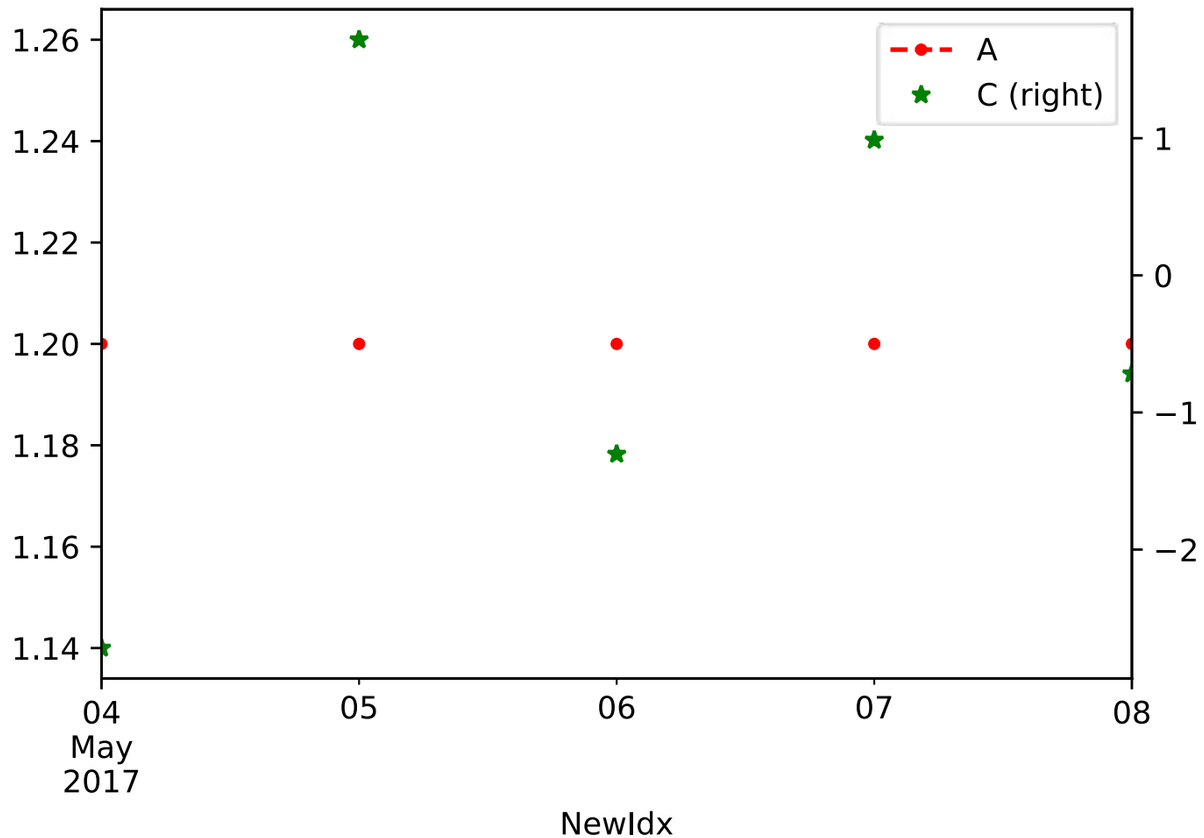
Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x114187160>
```



# Plotting II

In [32]:
```
frame[["A", "C"]].plot(
        color=["red", "green"],
        style=[".--","*"],
        grid=True,
        secondary_y=["C"]
    )
```

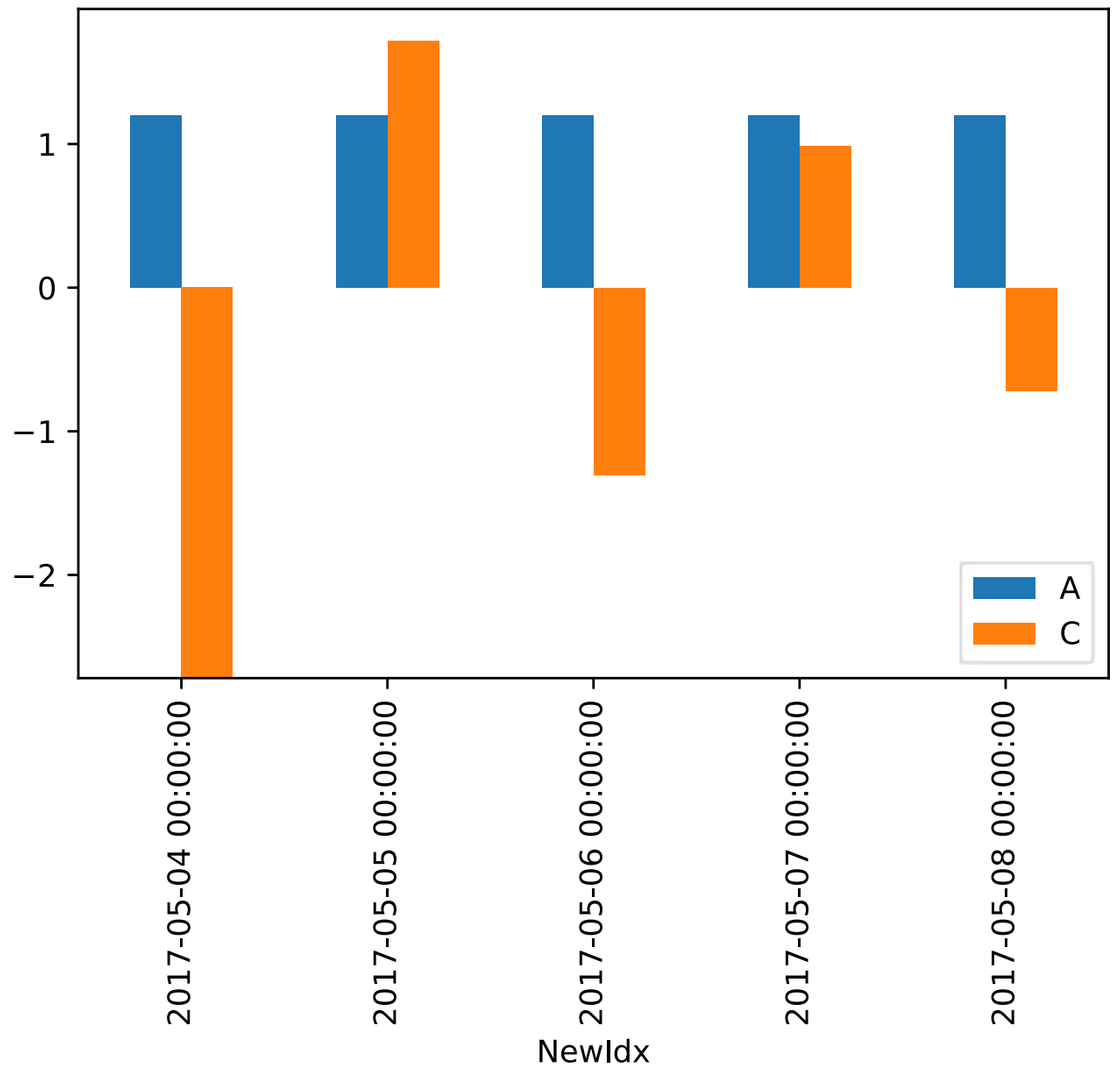
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x1141c75c0>



# Plotting III

In [33]:
```
frame[["A", "C"]].plot(kind="bar")
```

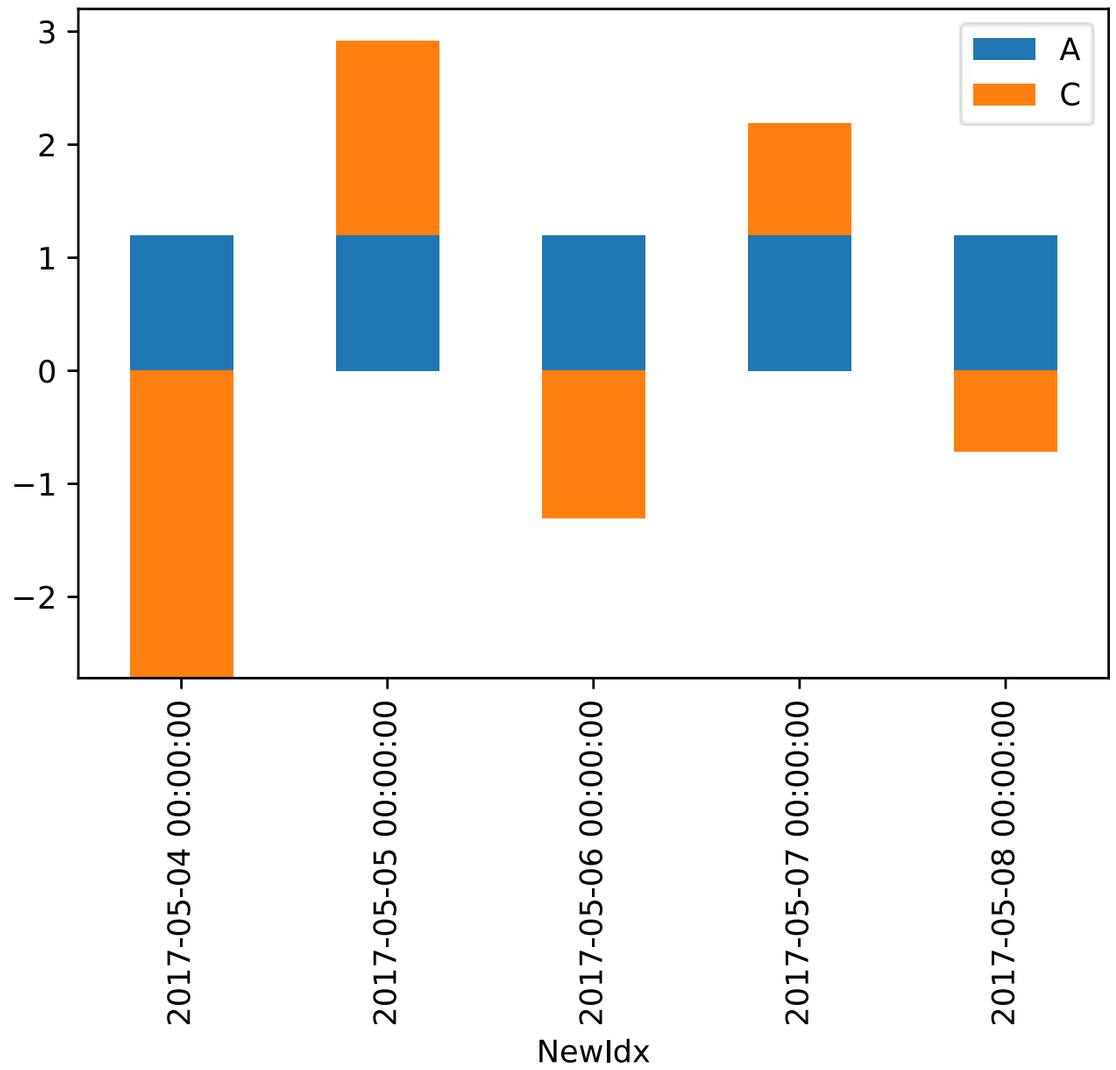
Out[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11433d5f8>
```



# Plotting III (2)

In [34]:
```
frame[["A", "C"]].plot(kind="bar", stacked=True)
```

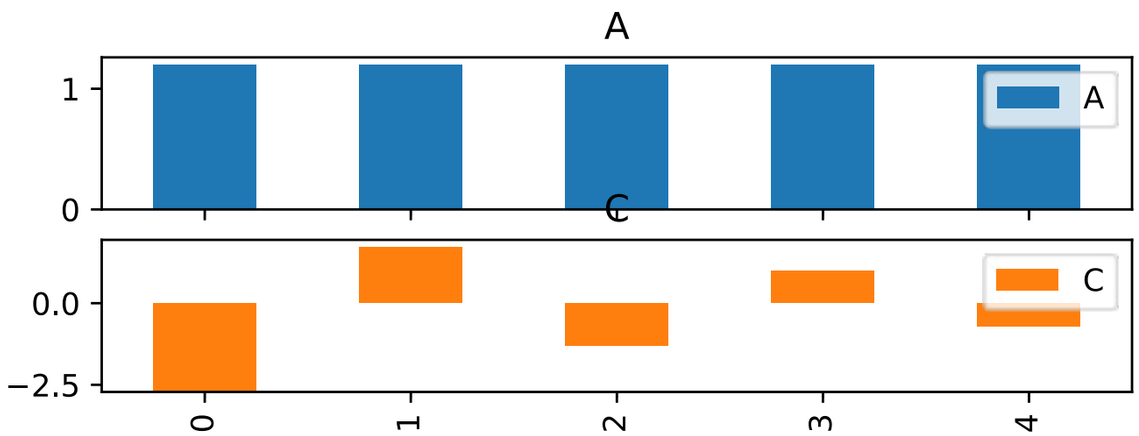
Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1143d89b0>
```



# Plotting III (3)

In [35]:
```
frame[["A", "C"]].reset_index().plot(kind="bar", subplots=True, figsize=(6,2))
```

Out[35]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x1144b3438>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x114593668>], dtype=object)
```



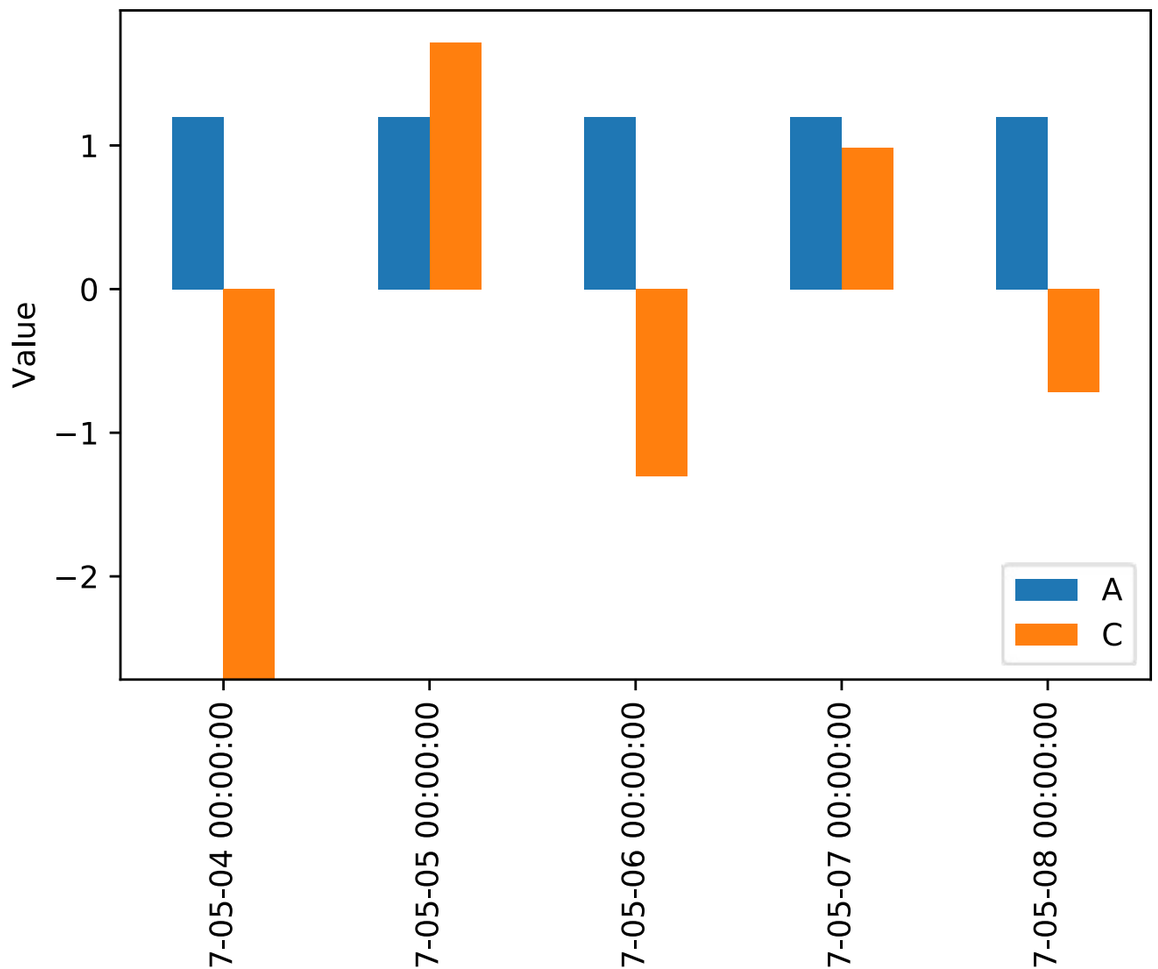

- Further `kinds`: `barh`, `box`, `hist`, `kde` (a better histogram!), `scatter`; more: http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html
- Instead of `.plot(kind="bar")`, also possible: `.plot.bar()`

# Advanced Plotting

## Combine Pandas & Matplotlib

Combine Pandas and Matplotlib by letting Pandas draw to an axis with `ax`

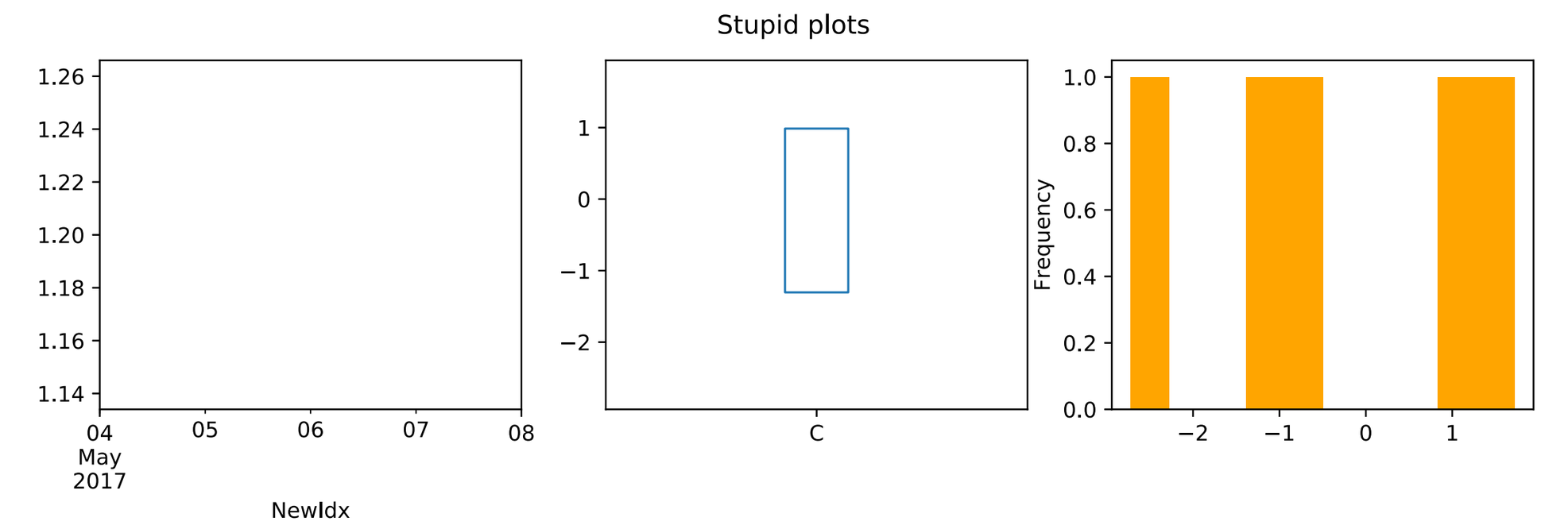
In [36]:
```
fig, ax = plt.subplots()
frame[["A", "C"]].plot(kind="bar", ax=ax)
ax.set_xlabel("Datetime")
ax.set_ylabel("Value")
fig.savefig("barplot.pdf")
```



## Combination II

In [38]:
```
fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, nrows=1, figsize=(12,3))
ax1 = frame["A"].plot.line(ax=ax1)
ax2 = frame["C"].plot.box(ax=ax2)
ax3 = frame["C"].plot.hist(ax=ax3, color="orange")
fig.suptitle("Stupid plots")
```

Out[38]:
```
<matplotlib.text.Text at 0x1148029b0>
```



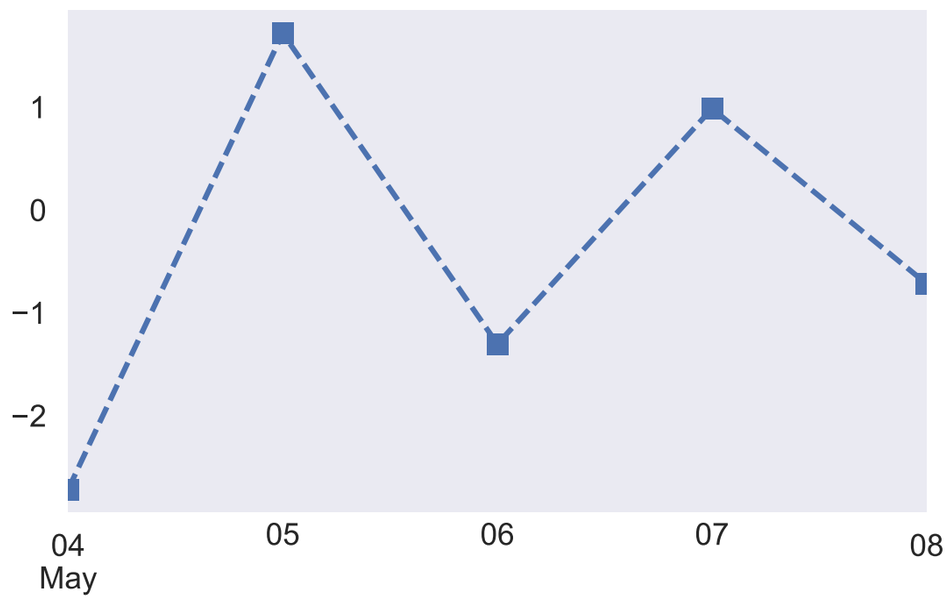
## Seaborn

> *Seaborn is a library for making attractive and informative statistical graphics in Python*

- Provides plotting interfaces
- And sets nice defaults
- Also: Colormaps
- → http://seaborn.pydata.org/

In [70]:
```
import seaborn as sns
sns.set(rc={"figure.figsize": (5, 3)})
frame["C"].plot(marker="s", linestyle="--")
```

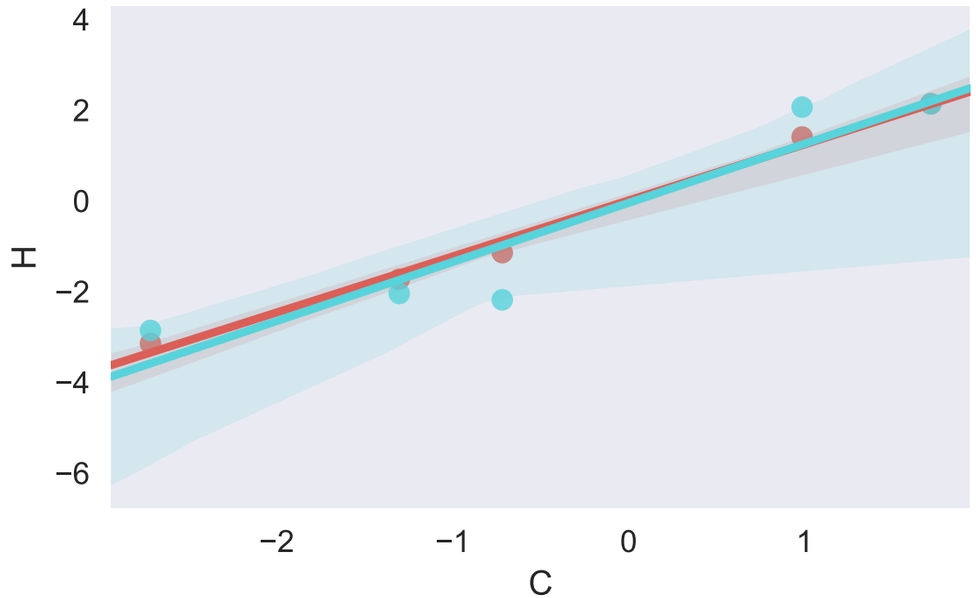
Out[70]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x117fae240>
```



## Seaborn Color Palette

In [71]:
```
frame["G"] = [(-1)**i * np.sqrt(i) + np.pi * (-1)**(i-1) for i in range(len(frame.index))]
frame["H"] = [(-1)**i * np.sqrt(i) + np.pi * (-1.1)**(i-1) for i in range(len(frame.index))]
```

In [72]:
```
with sns.color_palette("hls", 2):
    fig, ax = plt.subplots()
    sns.regplot(x="C", y="G", data=frame, ax=ax)
    sns.regplot(x="C", y="H", data=frame, ax=ax)
```



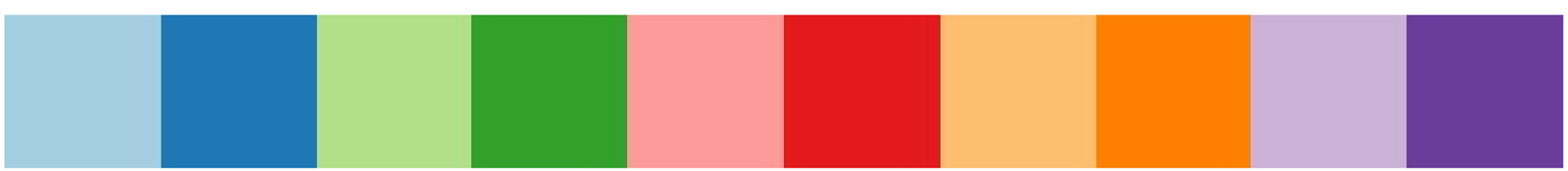
## Seaborn Color Palette II

In [73]:
```
sns.palplot(sns.color_palette())
```

In [74]:
```
sns.palplot(sns.color_palette("hls", 10))
```

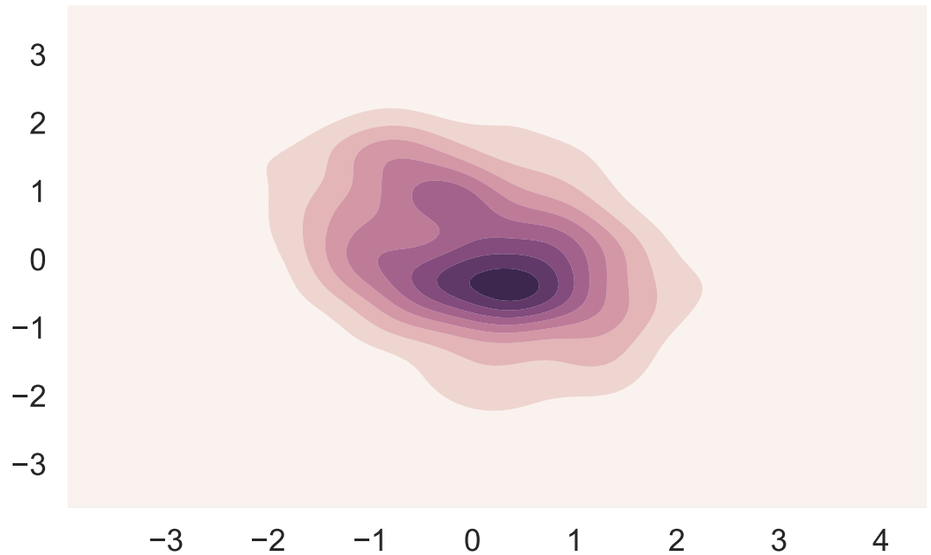
In [75]:
```
sns.palplot(sns.color_palette("hls", 20))
```

In [76]:
```
sns.palplot(sns.color_palette("Paired", 10))
```



## Seaborn Color Palette III / KDE Plot

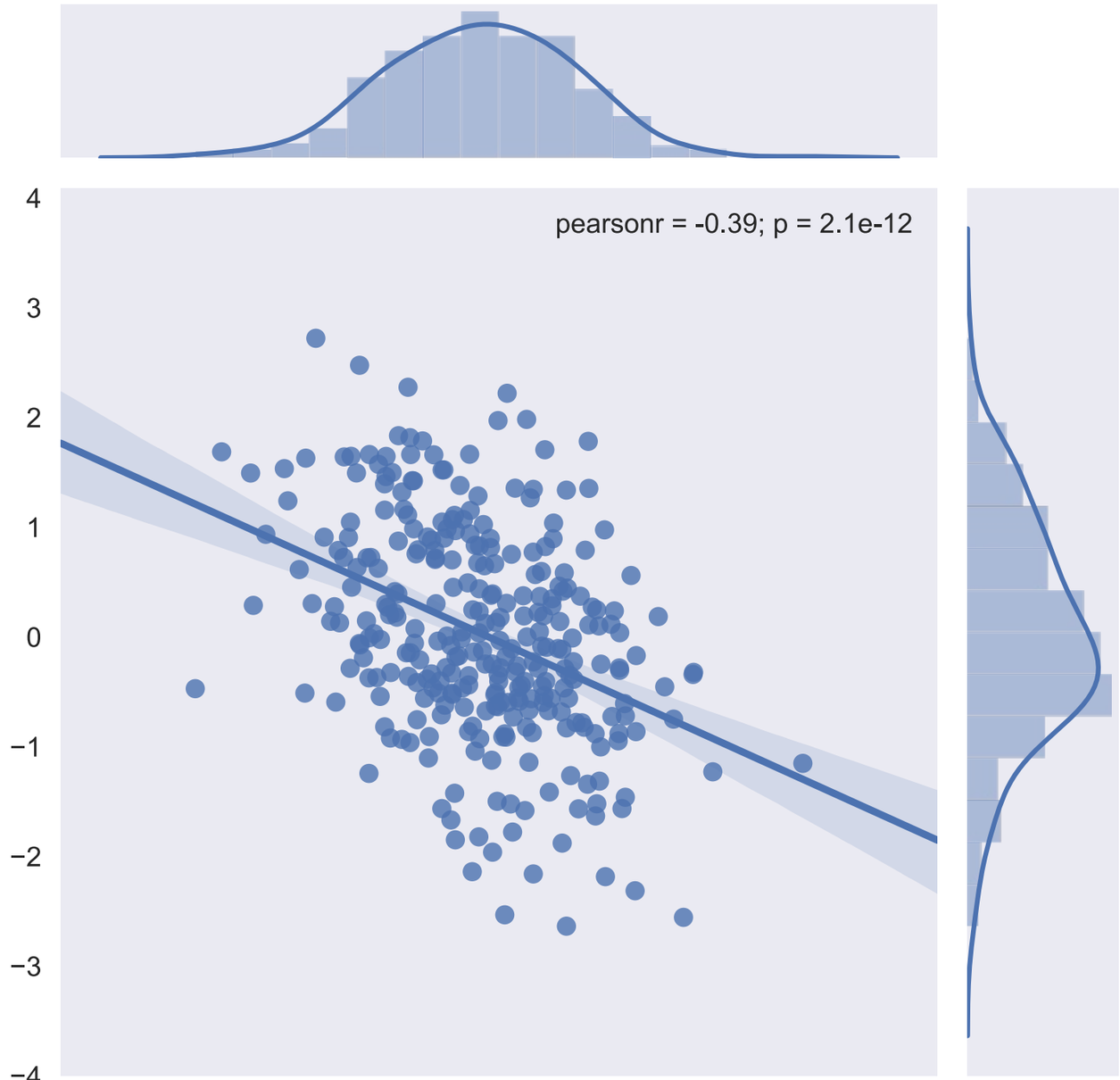
In [77]:
```
x, y = np.random.multivariate_normal([0, 0], [[1, -.5], [-.5, 1]], size=300).T
cmap = sns.cubehelix_palette(light=1, as_cmap=True)
sns.kdeplot(x, y, cmap=cmap, shade=True);
```



# Seaborn Color Palette IV / Jointplot

In [78]:
```
sns.jointplot(x=x, y=y, kind="reg")
```

Out[78]:
```
<seaborn.axisgrid.JointGrid at 0x1188c15f8>
```



# Complex Data

## Some real data...

Some PAPI counters for different number of particles (=program run lengths), compiled with different compilers

In [79]:
```
dfCounters = pd.read_csv("juron-jube-add_one_to_list.csv")
dfCounters.head(2)
```

Out[79]:

| | modules | compiler | n_particles | hwc | HWC |
|---|---|---|---|---|---|
| **0** | gcc/5.4.0 openmpi/2.0.2-gcc_5.4.0 PAPI/5.5.0-cuda | gfortran | 100000 | PAPI_TOT_INS | 32809671 |
| **1** | gcc/5.4.0 openmpi/2.0.2-gcc_5.4.0 PAPI/5.5.0-cuda | gfortran | 100000 | PAPI_TOT_CYC | 21246423 |

In [80]:
```
dfCounters = dfCounters.rename(columns={
    "modules": "Modules",
    "compiler": "Compiler",
    "n_particles": "Number of Particles",
    "hwc": "Counter Name",
    "HWC": "Counter Value"
})
dfCounters.head(2)
```

Out[80]:

| | Modules | Compiler | Number of Particles | Counter Name | Counter Value |
|---|---|---|---|---|---|
| **0** | gcc/5.4.0 openmpi/2.0.2-gcc_5.4.0 PAPI/5.5.0-cuda | gfortran | 100000 | PAPI_TOT_INS | 32809671 |

## Massaging

I want some relative values…

In [81]:
```
dfCounters["Counter Value (rel.)"] = dfCounters["Counter Value"] / dfCounters["Number of Particles"]
dfCounters.head(2)
```

Out[81]:

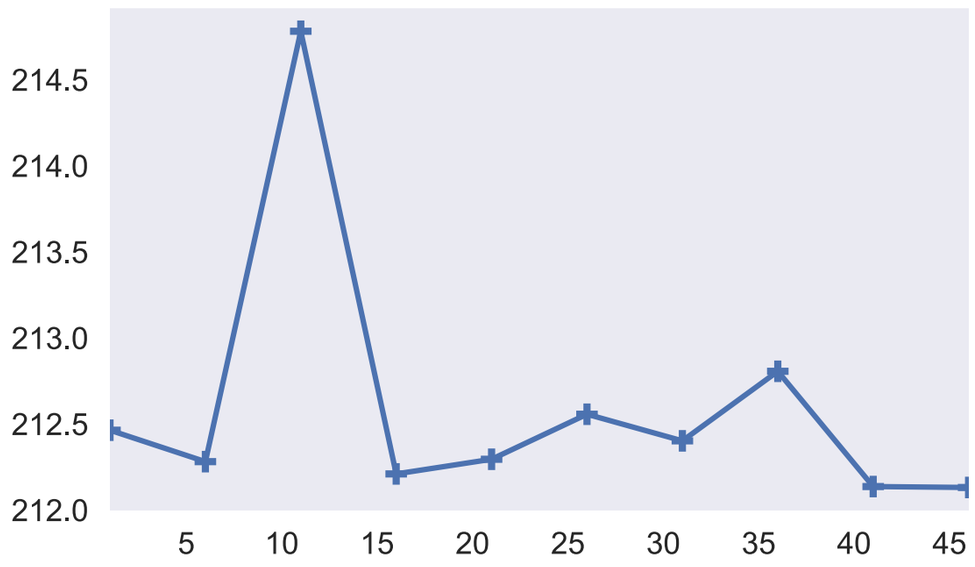
| | Modules | Compiler | Number of Particles | Counter Name | Counter Value | Counter Value (rel.) |
|---|---|---|---|---|---|---|
| **0** | gcc/5.4.0 openmpi/2.0.2-gcc_5.4.0 PAPI/5.5.0-cuda | gfortran | 100000 | PAPI_TOT_INS | 32809671 | 328.09671 |
| **1** | gcc/5.4.0 openmpi/2.0.2-gcc_5.4.0 PAPI/5.5.0-cuda | gfortran | 100000 | PAPI_TOT_CYC | 21246423 | 212.46423 |

## Some Values

Plot relative values of `PAPI_TOT_CYC` for `gfortoran`

In [82]:
```
dfCounters[
    (dfCounters["Compiler"] == "gfortran")
    &
    (dfCounters["Counter Name"] == "PAPI_TOT_CYC")
]["Counter Value (rel.)"]\
.plot(marker="P")
```

Out[82]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1185e66d8>
```

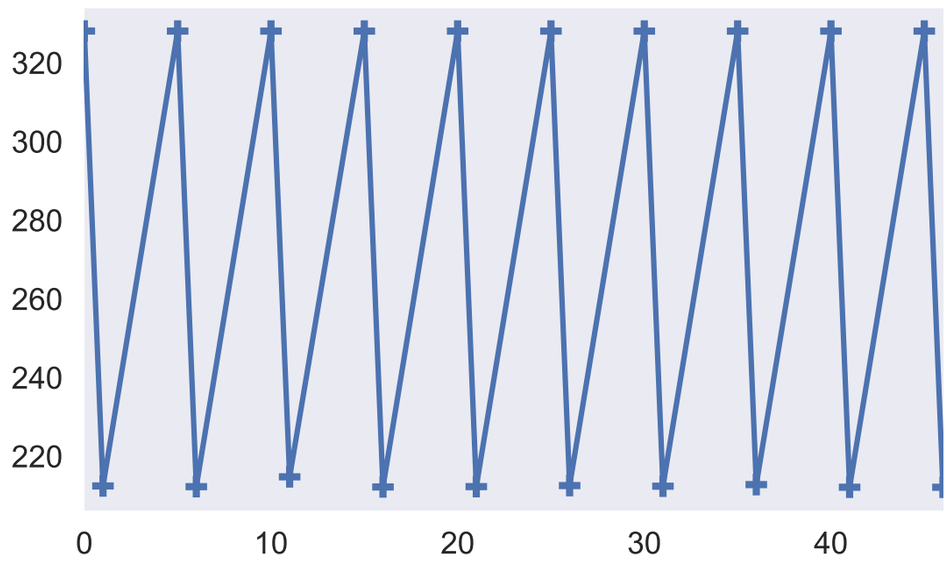


## More Values

Plot same relativ values, but also those of counter `PAPI_TOT_INS`

In [83]:
```
dfCounters[
    (dfCounters["Compiler"] == "gfortran")
    &
    ((dfCounters["Counter Name"] == "PAPI_TOT_CYC") | (dfCounters["Counter Name"] == "PAPI_T
OT_INS"))
]["Counter Value (rel.)"]\
.plot(marker="P")
```

Out[83]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1186db4a8>
```



## More Values

Plot same relativ values, but also those of counter `PAPI_TOT_INS`

Nope! Because

In [84]:
```
dfCounters[
    (dfCounters["Compiler"] == "gfortran")
    &
    ((dfCounters["Counter Name"] == "PAPI_TOT_CYC") | (dfCounters["Counter Name"] == "PAPI_T
OT_INS"))
].head(3)
```

Out[84]:

| | Modules | Compiler | Number of Particles | Counter Name | Counter Value | Counter Value (rel.) |
|---|---|---|---|---|---|---|
| **0** | gcc/5.4.0 openmpi/2.0.2-gcc_5.4.0 PAPI/5.5.0-cuda | gfortran | 100000 | PAPI_TOT_INS | 32809671 | 328.096710 |
| **1** | gcc/5.4.0 openmpi/2.0.2-gcc_5.4.0 PAPI/5.5.0-cuda | gfortran | 100000 | PAPI_TOT_CYC | 21246423 | 212.464230 |
| **5** | gcc/5.4.0 openmpi/2.0.2-gcc_5.4.0 PAPI/5.5.0-cuda | gfortran | 1000000 | PAPI_TOT_INS | 328081236 | 328.081236 |

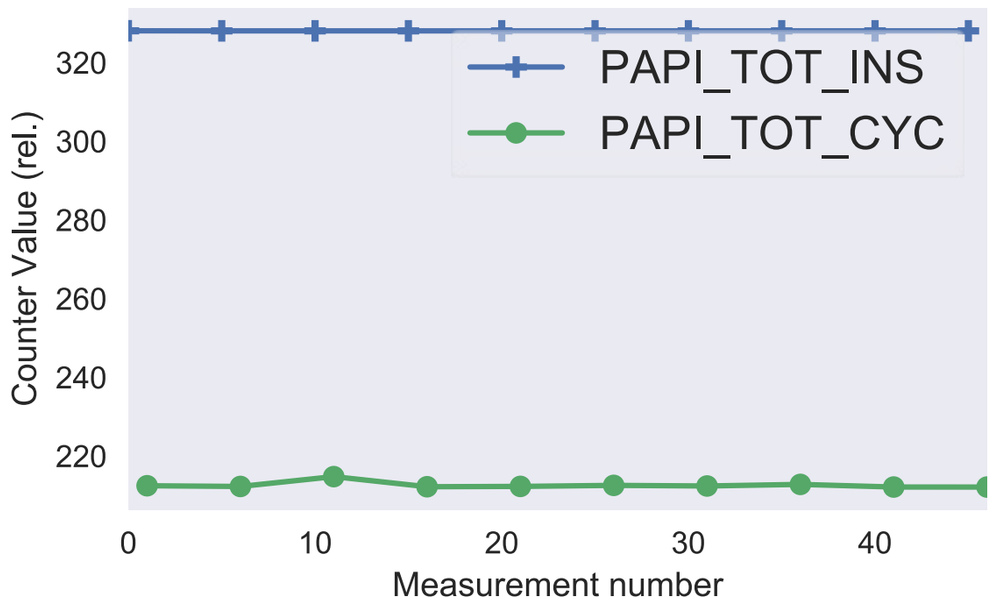
## Workaround

Create a canvas with matplotlib and explicitly draw to it

Care about legend etc.

In [85]:
```
fig, ax = plt.subplots()
ax = dfCounters[(dfCounters["Compiler"] == "gfortran") & (dfCounters["Counter Name"] == "PAP
I_TOT_INS")]["Counter Value (rel.)"].plot(marker="P", ax=ax, label="PAPI_TOT_INS")
ax = dfCounters[(dfCounters["Compiler"] == "gfortran") &  (dfCounters["Counter Name"] == "PA
PI_TOT_CYC")]["Counter Value (rel.)"].plot(marker="o", ax=ax, label="PAPI_TOT_CYC")
ax.legend(loc="best", frameon=True, fontsize=15, framealpha=0.5)
ax.set_xlabel("Measurement number")
ax.set_ylabel("Counter Value (rel.)")
```

Out[85]:
```
<matplotlib.text.Text at 0x118817c50>
```



## Pivoting!

Basically: Combine similar categorial data in a DataFrame

In [86]:
```
dfCounters.head(2)
```

Out[86]:

| | Modules | Compiler | Number of Particles | Counter Name | Counter Value | Counter Value (rel.) |
|---|---|---|---|---|---|---|
| **0** | gcc/5.4.0 openmpi/2.0.2-gcc_5.4.0 PAPI/5.5.0-cuda | gfortran | 100000 | PAPI_TOT_INS | 32809671 | 328.09671 |
| **1** | gcc/5.4.0 openmpi/2.0.2-gcc_5.4.0 PAPI/5.5.0-cuda | gfortran | 100000 | PAPI_TOT_CYC | 21246423 | 212.46423 |

Some data massaging: I want to remove `Modules` column; but to prevent double-entries, I want to rename all `mpifort` Compiler entries run with module `openmpi/1.10.2-pgi_16.10` loaded to `PGI+MPI`

In [87]:
```
dfCounters.loc[
    dfCounters["Modules"].str.contains("openmpi/1.10.2-pgi_16.10")
    &
    (dfCounters["Compiler"] == "mpifort"),
    "Compiler"
]  = "PGI+MPI"
```

In [88]:
```
dfCounters = dfCounters.drop("Modules", axis=1)
```

In [89]:
```
dfCounters.head(2)
```

Out[89]:

| | Compiler | Number of Particles | Counter Name | Counter Value | Counter Value (rel.) |
|---|---|---|---|---|---|
| **0** | gfortran | 100000 | PAPI_TOT_INS | 32809671 | 328.09671 |
| **1** | gfortran | 100000 | PAPI_TOT_CYC | 21246423 | 212.46423 |

## Pivoting, Actually

- `index`: What should be my new index? If array → hierarchical multi-index
- `values`: What value should be printed in the cells
- `columns`: What should be the new columns? If array → hierarchical

In [90]:
```
dfPivot = dfCounters.pivot_table(
    index="Number of Particles",
    values="Counter Value (rel.)",
    columns=["Compiler", "Counter Name"]
)
dfPivot.head(3)
```

Out[90]:

| Compiler | PGI+MPI | | | | | gfor |
|---|---|---|---|---|---|---|
| Counter Name | PAPI_L1_DCM | PAPI_L2_DCM | PAPI_STL_ICY | PAPI_TOT_CYC | PAPI_TOT_INS | PAPI |
| Number of Particles | | | | | | |
| **100000** | 3.032350 | 0.010760 | 479.309470 | 747.119030 | 780.156460 | 5.305 |
| **1000000** | 3.039885 | 0.008920 | 479.860810 | 747.309137 | 780.122863 | 2.744 |
| **2500000** | 3.419826 | 0.008527 | 479.873831 | 746.905123 | 780.120623 | 6.243 |

## Pivot and Stack

Maybe getting the counters to the index side is more useful?

In [91]:
```
dfPivot.stack().head(6)
```

Out[91]:

| | Compiler | PGI+MPI | gfortran | pgfortran |
|---|---|---|---|---|
| **Number of Particles** | **Counter Name** | | | |
| **100000** | **PAPI_L1_DCM** | 3.032350 | 5.30549 | 1.088945 |
| | **PAPI_L2_DCM** | 0.010760 | 0.00215 | 0.006175 |
| | **PAPI_STL_ICY** | 479.309470 | 137.86412 | 232.514715 |
| | **PAPI_TOT_CYC** | 747.119030 | 212.46423 | 436.386840 |
| | **PAPI_TOT_INS** | 780.156460 | 328.09671 | 672.144140 |
| **1000000** | **PAPI_L1_DCM** | 3.039885 | 2.74486 | 5.081417 |

… which is the same as

In [92]:
```
dfCounters.pivot_table(
    index=["Number of Particles", "Counter Name"],
    values="Counter Value (rel.)",
    columns="Compiler"
).head(6)
```

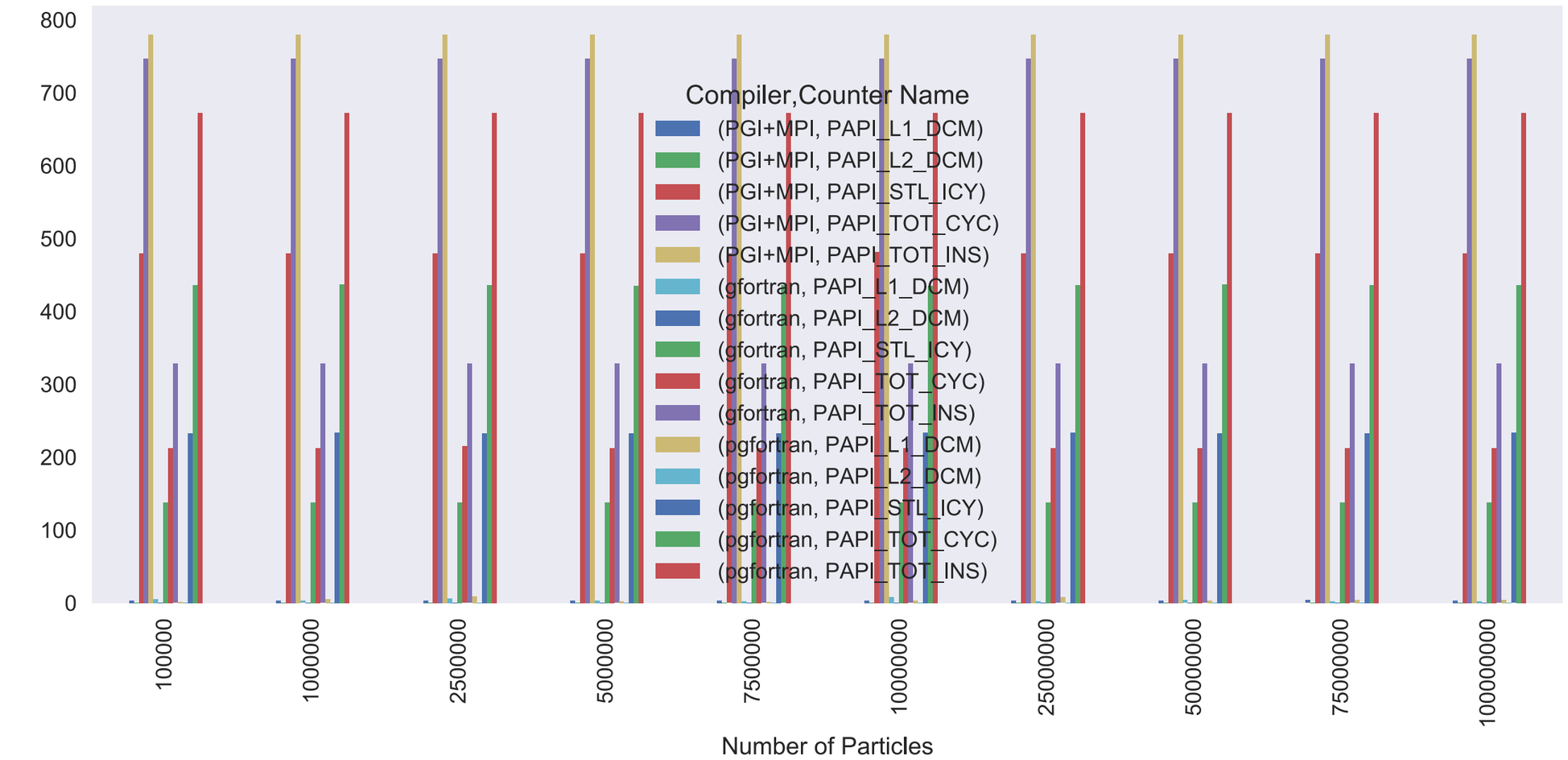
Out[92]:

| | Compiler | PGI+MPI | gfortran | pgfortran |
|---|---|---|---|---|

# Plotting Pivoted DataFrames

In [93]:
```
dfPivot.plot(kind="bar", figsize=(12,5))
```

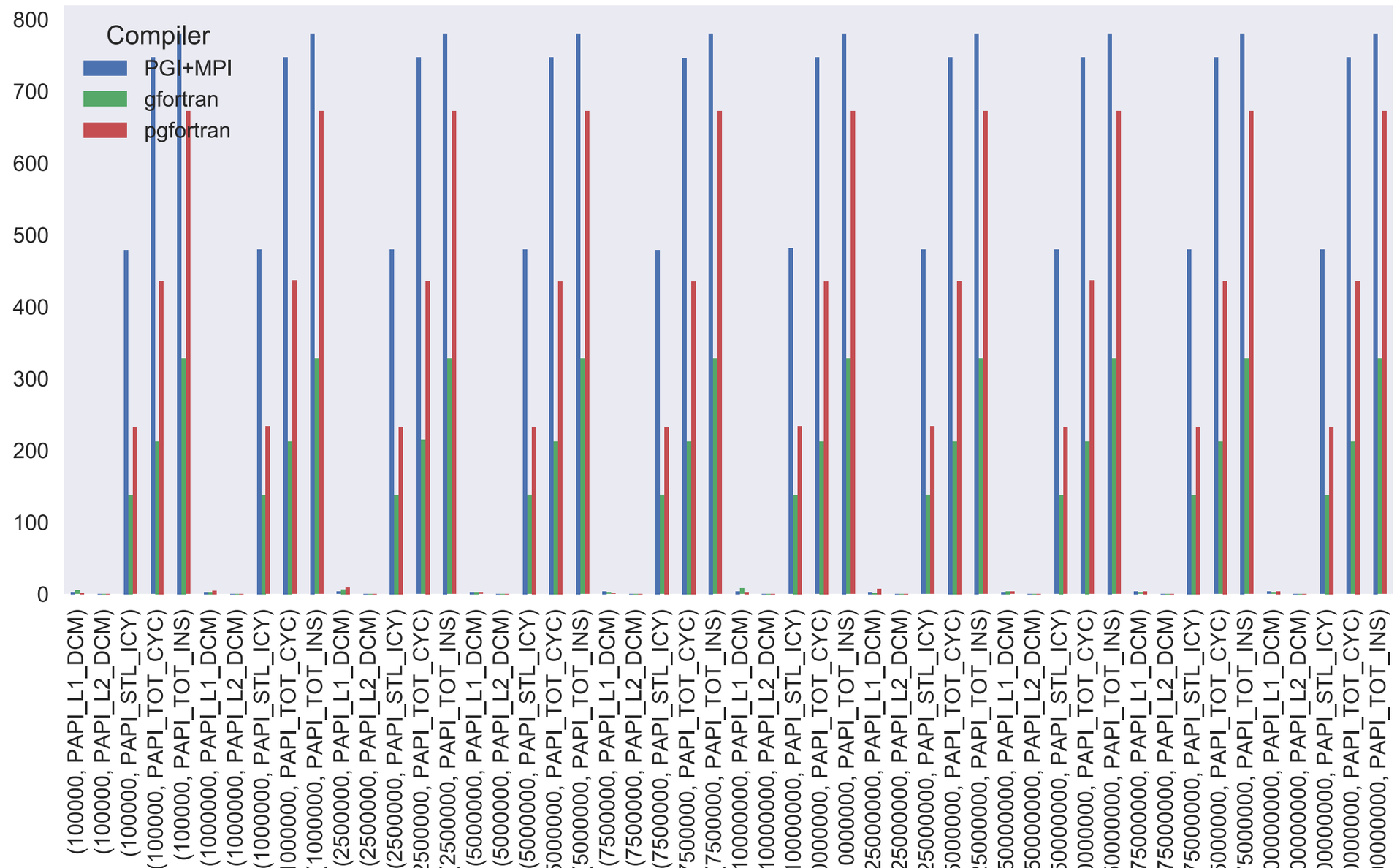
Out[93]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1188ff2b0>
```



# Plotting Pivoted DataFrames II

In [94]:
```
dfPivot.stack().plot(kind="bar", figsize=(11,5))
```

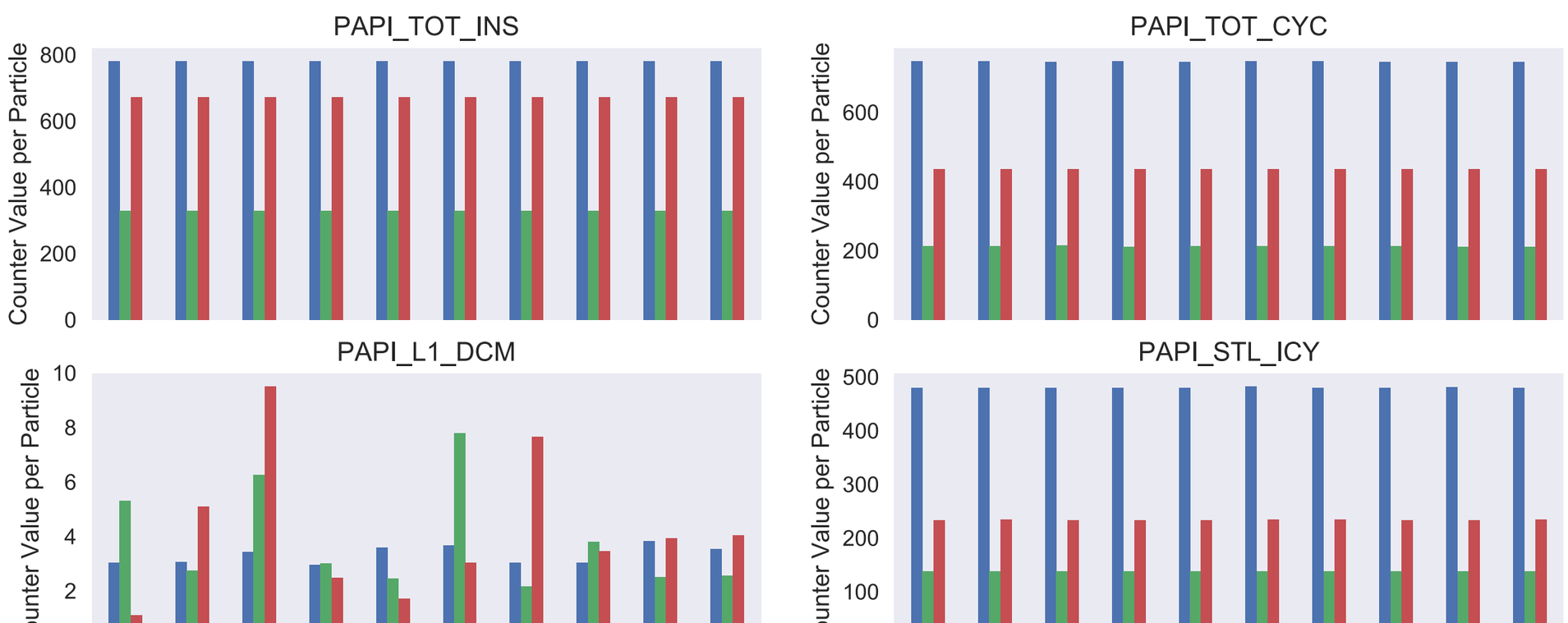
Out[94]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x118bedd68>
```



## Plotting Pivoted DataFrames III

Focus on four counters, plot them next to each other

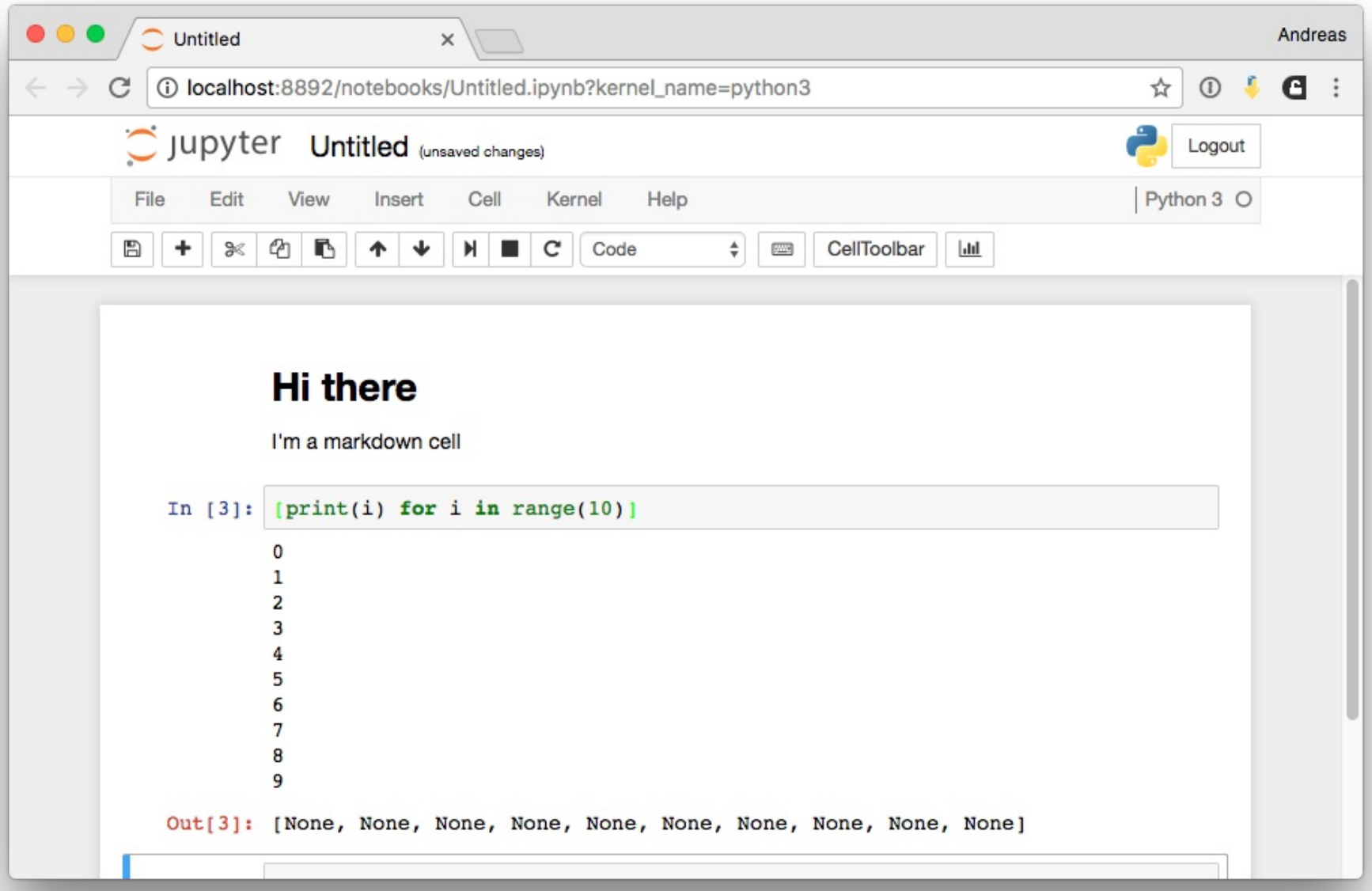
In [95]:
```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(ncols=2, nrows=2, sharex=True, figsize=(12,5))
for (ax, counter) in zip([ax1, ax2, ax3, ax4], ["PAPI_TOT_INS", "PAPI_TOT_CYC", "PAPI_L1_DCM
", "PAPI_STL_ICY"]):
    ax = dfPivot.stack().loc[(slice(None), counter),:].plot(kind="bar", ax=ax, legend=False)
    labels = [int(label.get_text().split(",")[0][1:-1]) for label in ax.get_xticklabels()]
    ax.set_title(counter)
    ax.set_xlabel("Number of Particles")
    ax.set_ylabel("Counter Value per Particle")
    ax.set_xticklabels(labels)
```



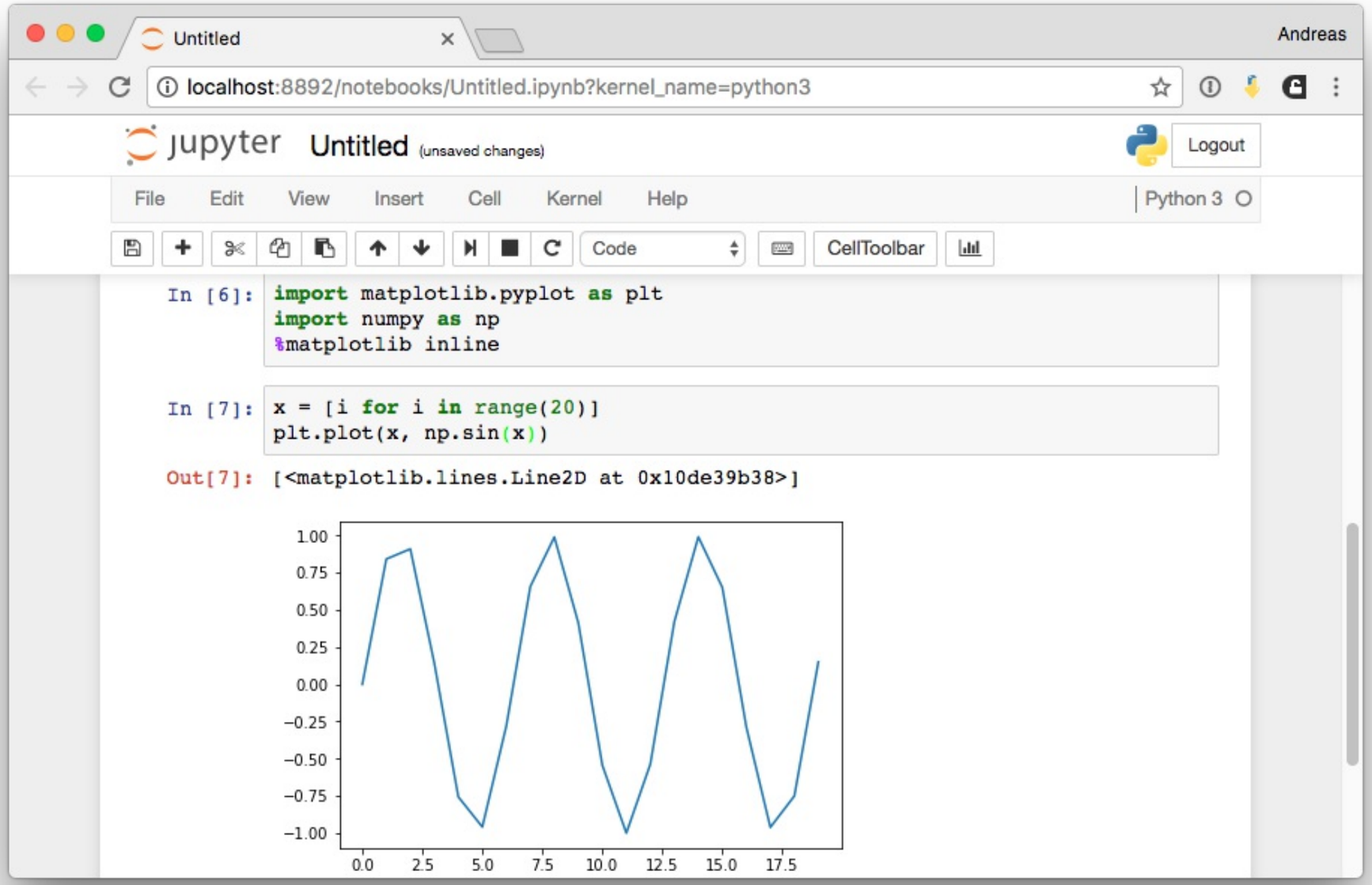
# Jupyter Notebooks

# Introduction

Use Python in your browser, interactively



# Inline Magic I



## Inline Magic II

In [96]:
```
%timeit np.sin(range(1000))
```

```
1000 loops, best of 3: 698 µs per loop
```

In [97]:
```
%ls .
```

```
Pandas-Analysis.ipynb                juron-jube-add_one_to_list.csv
Pandas-Analysis.slides.html          notebook-screenshot--inline1.png
convertNotebookToHtmlSlides.sh*      notebook-screenshot.png
convertNotebookToPdfDocument.sh*     reveal.js/
convertSlidesToPdf.sh*               serveSlidesForPresentation.sh*
custom.css
```

In [98]:
```
!pip install something
```

```
Collecting something
  Could not find a version that satisfies the requirement something (from versions: )
No matching distribution found for something
```

In [99]:
```
%lsmagic
```

Out[99]:
```
Available line magics:
%alias  %alias_magic  %autocall  %automagic  %autosave  %bookmark  %cat  %cd  %clear  %color
s  %config  %connect_info  %cp  %debug  %dhist  %dirs  %doctest_mode  %ed  %edit  %env  %gui
  %hist  %history  %killbgscripts  %ldir  %less  %lf  %lk  %ll  %load  %load_ext  %loadpy  %
logoff  %logon  %logstart  %logstate  %logstop  %ls  %lsmagic  %lx  %macro  %magic  %man  %m
atplotlib  %mkdir  %more  %mv  %notebook  %page  %pastebin  %pdb  %pdef  %pdoc  %pfile  %pin
fo  %pinfo2  %popd  %pprint  %precision  %profile  %prun  %psearch  %psource  %pushd  %pwd
%pycat  %pylab  %qtconsole  %quickref  %recall  %rehashx  %reload_ext  %rep  %rerun  %reset
 %reset_selective  %rm  %rmdir  %run  %save  %sc  %set_env  %store  %sx  %system  %tb  %time
  %timeit  %unalias  %unload_ext  %who  %who_ls  %whos  %xdel  %xmode
```

## Converting

- Notebooks are rendered at Github and our Gitlab server
- Can be converted to static HTML
- Can be converted to PDF, Markdown, reST, **Slides**

**This presentation is one large Jupyter Notebook**

# The End