Bar Plots

The `plot.bar()` and `plot.barh()` make vertical and horizontal bar plots, respectively. In this case, the Series or DataFrame index will be used as the x (`bar`) or y (`barh`) ticks (see Figure 9-15):

```
In [64]: fig, axes = plt.subplots(2, 1)

In [65]: data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))

In [66]: data.plot.bar(ax=axes[0], color='k', alpha=0.7)
Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb62493d470>

In [67]: data.plot.barh(ax=axes[1], color='k', alpha=0.7)
```

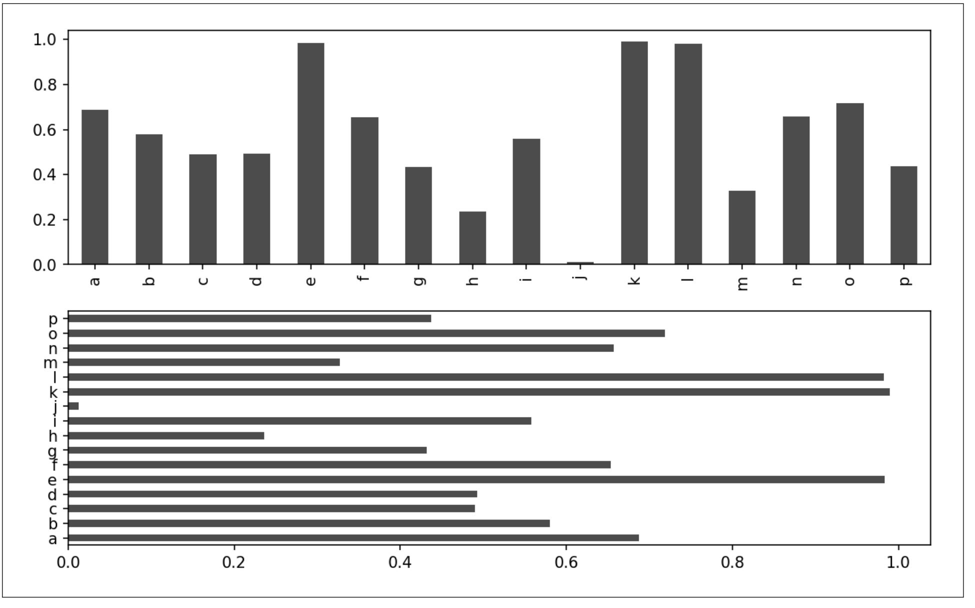

Figure 9-15. Horizonal and vertical bar plot

The options `color='k'` and `alpha=0.7` set the color of the plots to black and use partial transparency on the filling.

With a DataFrame, bar plots group the values in each row together in a group in bars, side by side, for each value. See Figure 9-16:

```
In [69]: df = pd.DataFrame(np.random.rand(6, 4),
   ....:                   index=['one', 'two', 'three', 'four', 'five', 'six'],
   ....:                   columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))

In [70]: df
Out[70]:
Genus         A         B         C         D
one    0.370670  0.602792  0.229159  0.486744
two    0.420082  0.571653  0.049024  0.880592
three  0.814568  0.277160  0.880316  0.431326
four   0.374020  0.899420  0.460304  0.100843
five   0.433270  0.125107  0.494675  0.961825
six    0.601648  0.478576  0.205690  0.560547

In [71]: df.plot.bar()
```

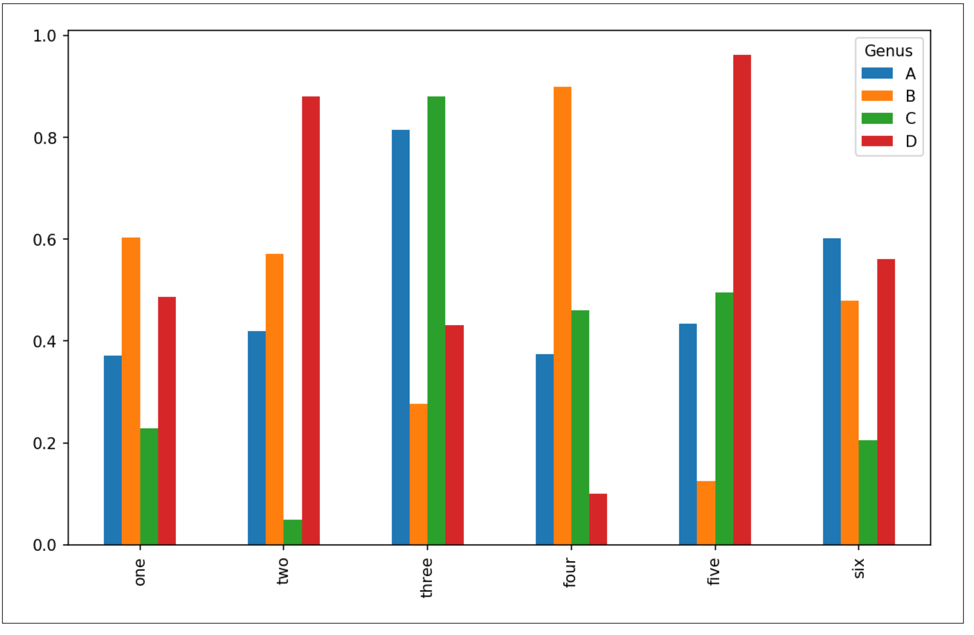

Figure 9-16. DataFrame bar plot

Note that the name "Genus" on the DataFrame's columns is used to title the legend.

We create stacked bar plots from a DataFrame by passing `stacked=True`, resulting in the value in each row being stacked together (see Figure 9-17):

```
In [73]: df.plot.barh(stacked=True, alpha=0.5)
```

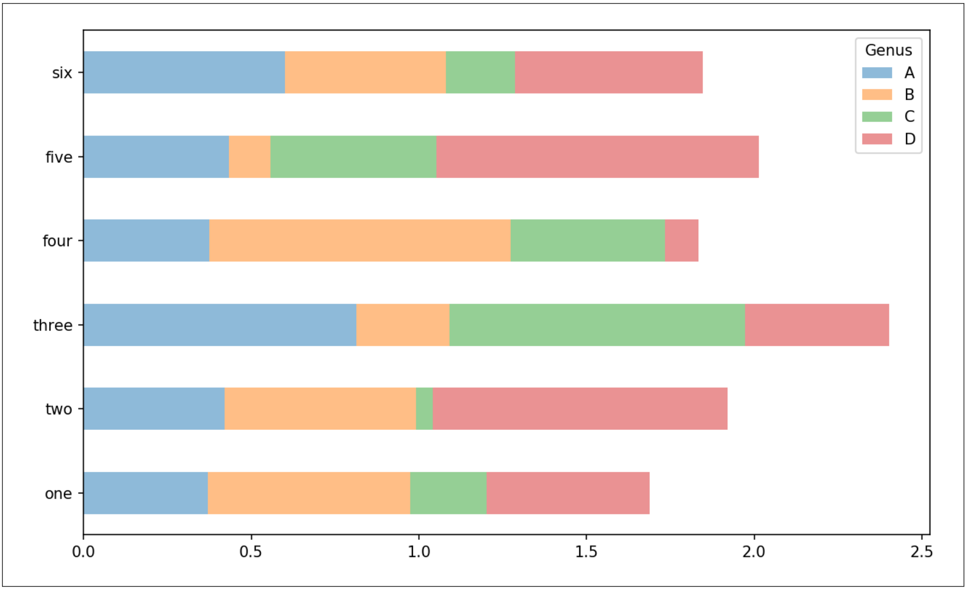

Figure 9-17. DataFrame stacked bar plot

A useful recipe for bar plots is to visualize a Series's value frequency using `value_counts`: `s.value_counts().plot.bar()`.

Returning to the tipping dataset used earlier in the book, suppose we wanted to make a stacked bar plot showing the percentage of data points for each party size on each day. I load the data using `read_csv` and make a cross-tabulation by day and party size:

```
In [75]: tips = pd.read_csv('examples/tips.csv')

In [76]: party_counts = pd.crosstab(tips['day'], tips['size'])

In [77]: party_counts
Out[77]:
size  1   2   3   4  5  6
day
Fri   1  16   1   1  0  0
Sat   2  53  18  13  1  0
Sun   0  39  15  18  3  1
Thur  1  48   4   5  1  3
```

```
# Not many 1- and 6-person parties
In [78]: party_counts = party_counts.loc[:, 2:5]
```

Then, normalize so that each row sums to 1 and make the plot (see Figure 9-18):

```
# Normalize to sum to 1
In [79]: party_pcts = party_counts.div(party_counts.sum(1), axis=0)

In [80]: party_pcts
Out[80]:
size         2         3         4         5
day
Fri   0.888889  0.055556  0.055556  0.000000
Sat   0.623529  0.211765  0.152941  0.011765
Sun   0.520000  0.200000  0.240000  0.040000
Thur  0.827586  0.068966  0.086207  0.017241

In [81]: party_pcts.plot.bar()
```

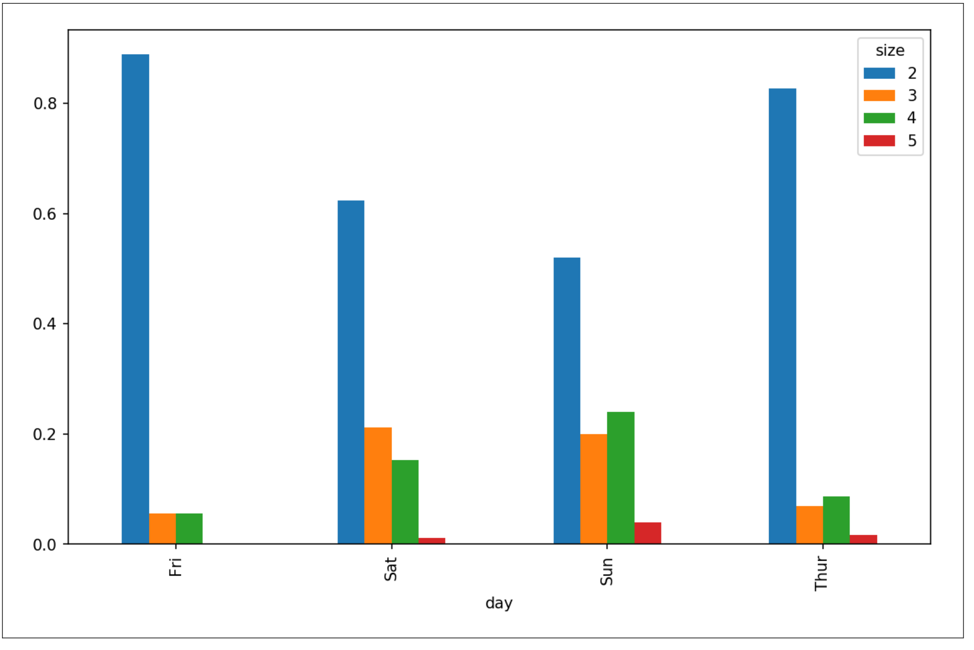

Figure 9-18. Fraction of parties by size on each day

So you can see that party sizes appear to increase on the weekend in this dataset.

With data that requires aggregation or summarization before making a plot, using the `seaborn` package can make things much simpler. Let's look now at the tipping percentage by day with seaborn (see Figure 9-19 for the resulting plot):

```
In [83]: import seaborn as sns

In [84]: tips['tip_pct'] = tips['tip'] / (tips['total_bill'] - tips['tip'])

In [85]: tips.head()
Out[85]:
   total_bill   tip smoker  day    time  size   tip_pct
0       16.99  1.01     No  Sun  Dinner     2  0.063204
1       10.34  1.66     No  Sun  Dinner     3  0.191244
2       21.01  3.50     No  Sun  Dinner     3  0.199886
3       23.68  3.31     No  Sun  Dinner     2  0.162494
4       24.59  3.61     No  Sun  Dinner     4  0.172069

In [86]: sns.barplot(x='tip_pct', y='day', data=tips, orient='h')
```

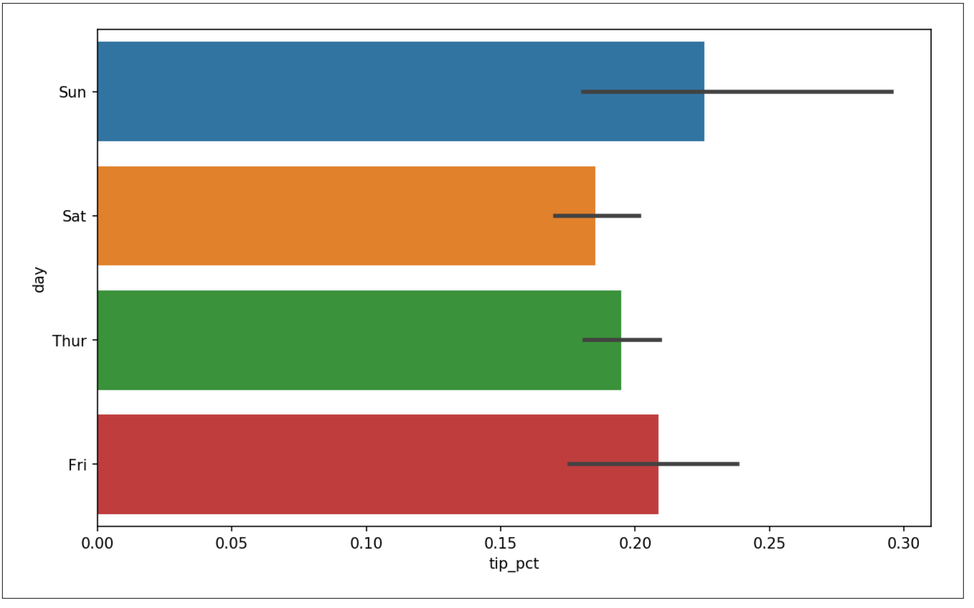

Figure 9-19. Tipping percentage by day with error bars

Plotting functions in seaborn take a `data` argument, which can be a pandas DataFrame. The other arguments refer to column names. Because there are multiple observations for each value in the `day`, the bars are the average value of `tip_pct`. The black lines drawn on the bars represent the 95% confidence interval (this can be configured through optional arguments).

`seaborn.barplot` has a `hue` option that enables us to split by an additional categorical value (Figure 9-20):

```
In [88]: sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h')
```

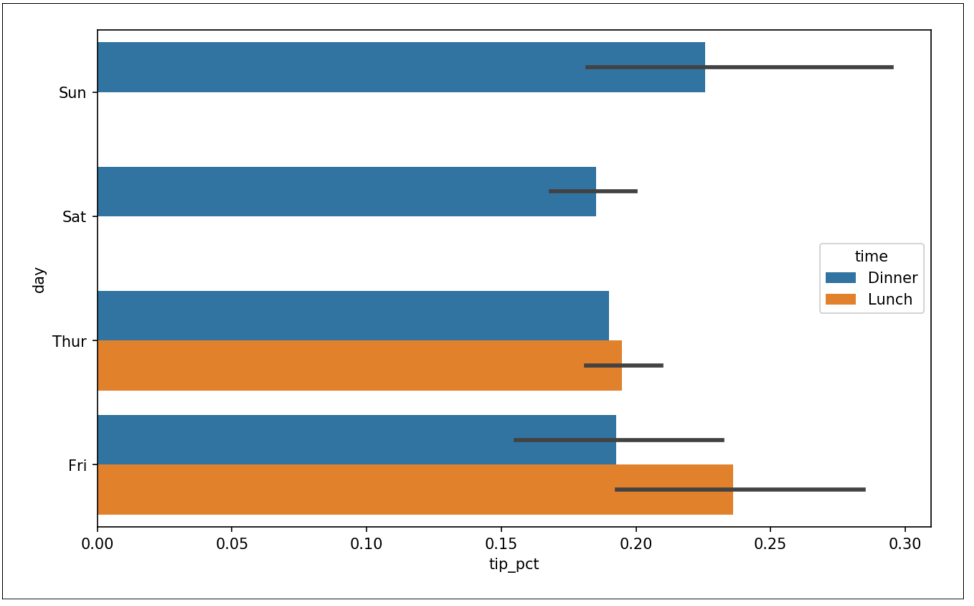

Figure 9-20. Tipping percentage by day and time

Notice that seaborn has automatically changed the aesthetics of plots: the default color palette, plot background, and grid line colors. You can switch between different plot appearances using `seaborn.set`:

```
In [90]: sns.set(style="whitegrid")
```

Histograms and Density Plots

A histogram is a kind of bar plot that gives a discretized display of value frequency. The data points are split into discrete, evenly spaced bins, and the number of data points in each bin is plotted. Using the tipping data from before, we can make a histogram of tip percentages of the total bill using the `plot.hist` method on the Series (see Figure 9-21):

```
In [92]: tips['tip_pct'].plot.hist(bins=50)
```

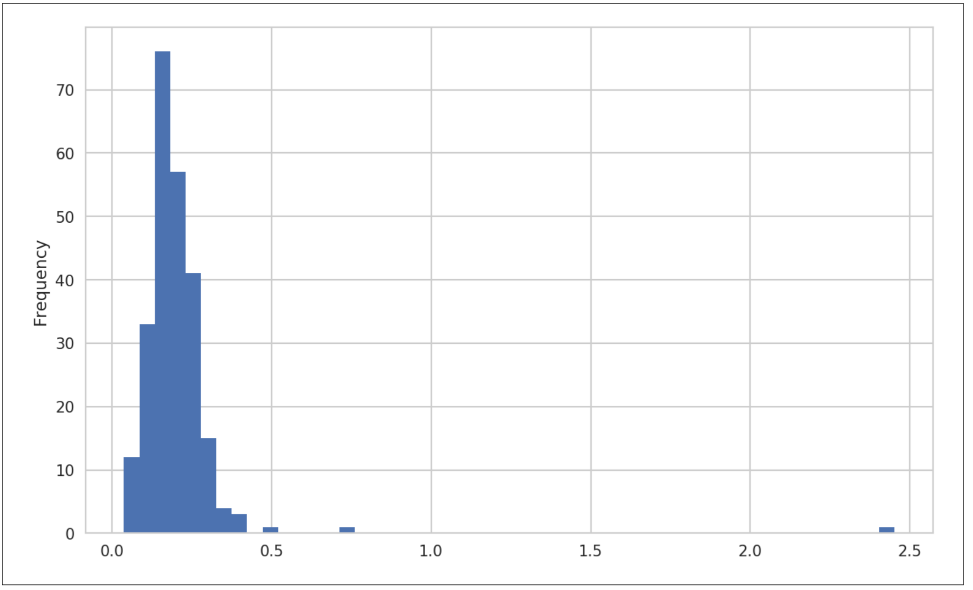

Figure 9-21. Histogram of tip percentages

A related plot type is a *density plot*, which is formed by computing an estimate of a continuous probability distribution that might have generated the observed data. The usual procedure is to approximate this distribution as a mixture of "kernels"—that is, simpler distributions like the normal distribution. Thus, density plots are also known as kernel density estimate (KDE) plots. Using `plot.kde` makes a density plot using the conventional mixture-of-normals estimate (see Figure 9-22):

```
In [94]: tips['tip_pct'].plot.density()
```

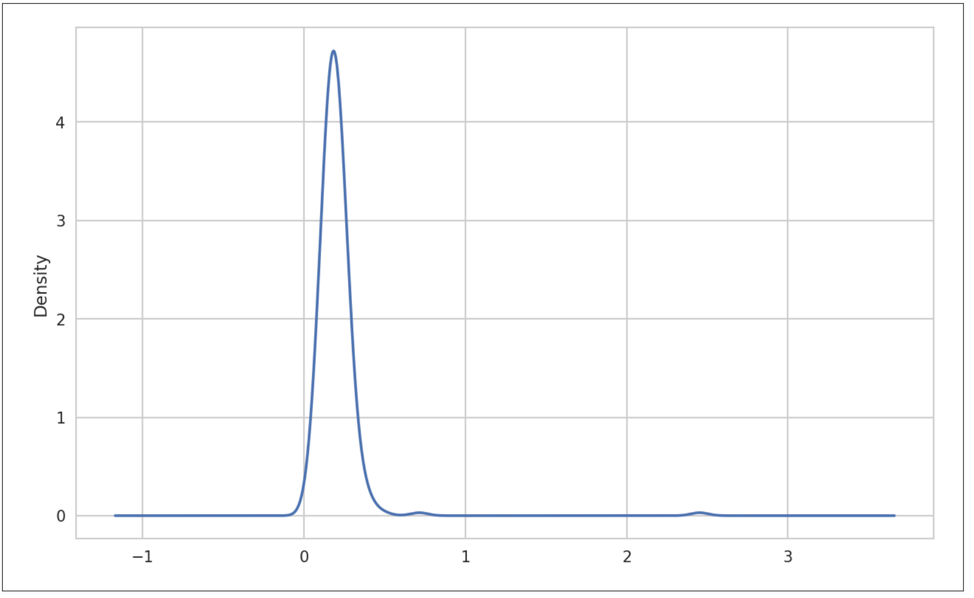

Figure 9-22. Density plot of tip percentages

Seaborn makes histograms and density plots even easier through its `distplot` method, which can plot both a histogram and a continuous density estimate simultaneously. As an example, consider a bimodal distribution consisting of draws from two different standard normal distributions (see Figure 9-23):

```
In [96]: comp1 = np.random.normal(0, 1, size=200)

In [97]: comp2 = np.random.normal(10, 2, size=200)

In [98]: values = pd.Series(np.concatenate([comp1, comp2]))

In [99]: sns.distplot(values, bins=100, color='k')
```

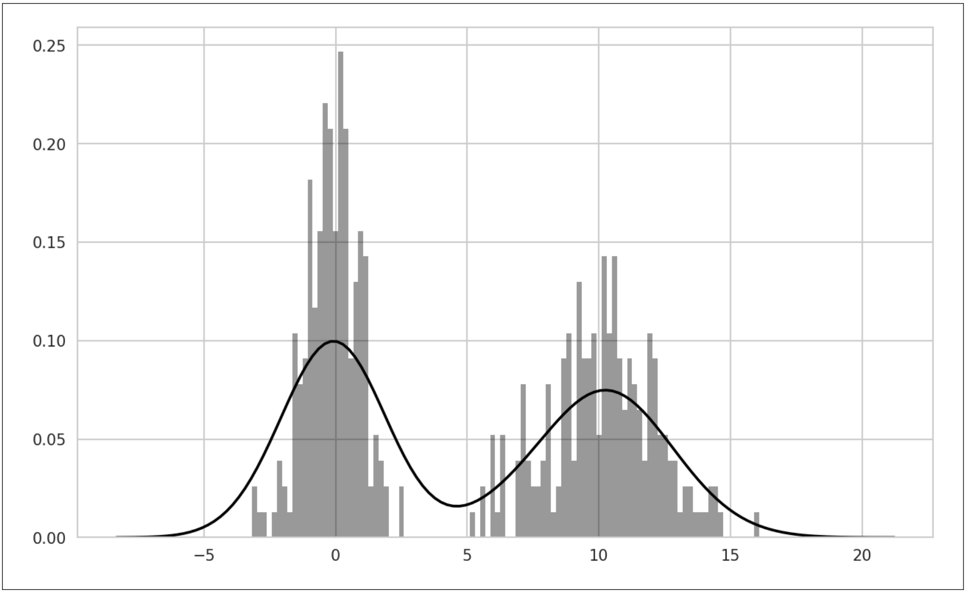

Figure 9-23. Normalized histogram of normal mixture with density estimate

Scatter or Point Plots

Point plots or scatter plots can be a useful way of examining the relationship between two one-dimensional data series. For example, here we load the `macrodata` dataset from the statsmodels project, select a few variables, then compute log differences:

```
In [100]: macro = pd.read_csv('examples/macrodata.csv')

In [101]: data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]

In [102]: trans_data = np.log(data).diff().dropna()

In [103]: trans_data[-5:]
Out[103]:
          cpi        m1  tbilrate     unemp
198 -0.007904  0.045361 -0.396881  0.105361
199 -0.021979  0.066753 -2.277267  0.139762
200  0.002340  0.010286  0.606136  0.160343
201  0.008419  0.037461 -0.200671  0.127339
202  0.008894  0.012202 -0.405465  0.042560
```

We can then use seaborn's `regplot` method, which makes a scatter plot and fits a linear regression line (see Figure 9-24):

```
In [105]: sns.regplot('m1', 'unemp', data=trans_data)
Out[105]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb613720be0>

In [106]: plt.title('Changes in log %s versus log %s' % ('m1', 'unemp'))
```

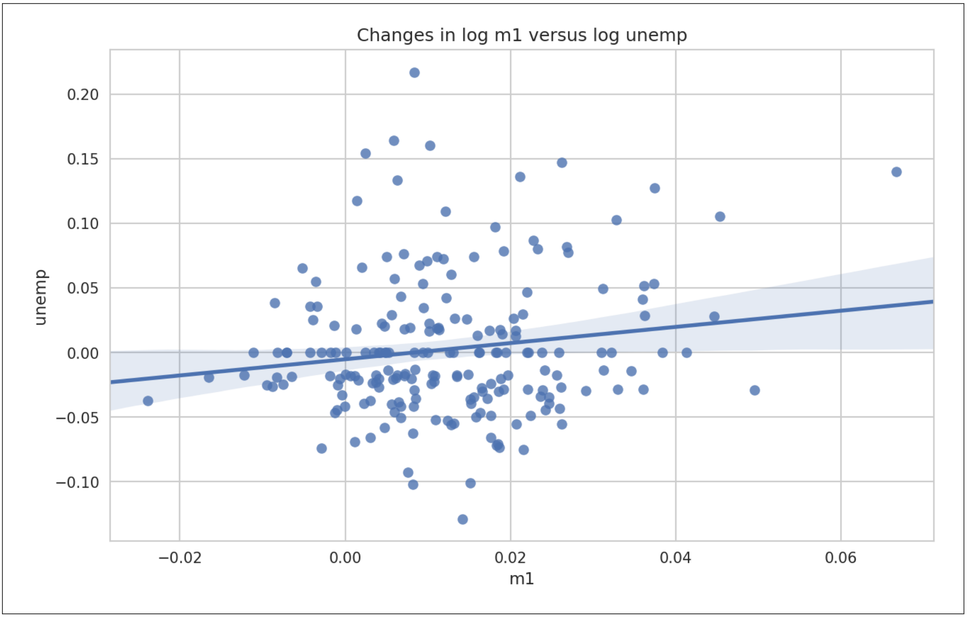

Figure 9-24. A seaborn regression/scatter plot

In exploratory data analysis it's helpful to be able to look at all the scatter plots among a group of variables; this is known as a *pairs* plot or *scatter plot matrix*. Making such a plot from scratch is a bit of work, so seaborn has a convenient `pairplot` function, which supports placing histograms or density estimates of each variable along the diagonal (see Figure 9-25 for the resulting plot):

```
In [107]: sns.pairplot(trans_data, diag_kind='kde', plot_kws={'alpha': 0.2})
```