8 Pandas 2: Plotting

Lab Objective: *Clear, insightful visualizations are a crucial part of data analysis. To facilitate quick data visualization, pandas includes several tools that wrap around matplotlib. These tools make it easy to compare different parts of a data set and explore the data as a whole.*

Overview of Plotting Tools

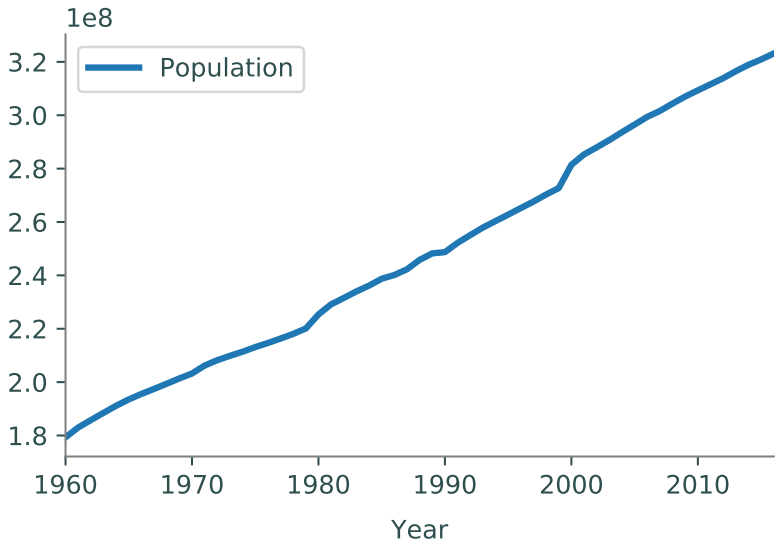
The main tool for visualization in pandas is the `plot()` method of the `Series` and `DataFrame`. The method has a keyword argument `kind` that specifies the type of plot to draw. The valid options for `kind` are detailed below.

Plot Type	`plot()` ID	Uses and Advantages
Line plot	`"line"`	Show trends ordered in data; easy to compare multiple data sets
Scatter plot	`"scatter"`	Compare exactly two data sets, independent of ordering
Bar plot	`"bar"`, `"barh"`	Compare categorical or sequential data
Histogram	`"hist"`	Show frequencies of one set of values, independent of ordering
Box plot	`"box"`	Display min, median, max, and quartiles; compare data distributions
Hexbin plot	`"hexbin"`	2D histogram; reveal density of cluttered scatter plots

Table 8.1: Uses for the `plot()` method of the pandas `Series` and `DataFrame`. The plot ID is the value of the keyword argument `kind`. That is, `df.plot(kind="scatter")` creates a scatter plot. The default `kind` is `"line"`.

The `plot()` method calls `plt.plot()`, `plt.hist()`, `plt.scatter()`, or another matplotlib plotting function, but it also assigns axis labels, tick marks, legends, and a few other things based on the index and the data. Most calls to `plot()` specify the `kind` of plot and which `Series` to use as the `x` and `y` axes. By default, the `index` of the `Series` or `DataFrame` is used for the `x` axis.

```
>>> import pandas as pd
>>> from matplotlib import pyplot as plt

>>> crime = pd.read_csv("crime_data.csv", index_col="Year")
>>> crime.plot(y="Population")  # Plot population against the index (years).
```



In this case, the call to the `plot()` method is essentially equivalent to the following code.

```
>>> plt.plot(crime.index, crime["Population"], label="Population")
>>> plt.xlabel(crime.index.name)
>>> plt.xlim(min(crime.index), max(crime.index))
>>> plt.legend(loc="best")
```

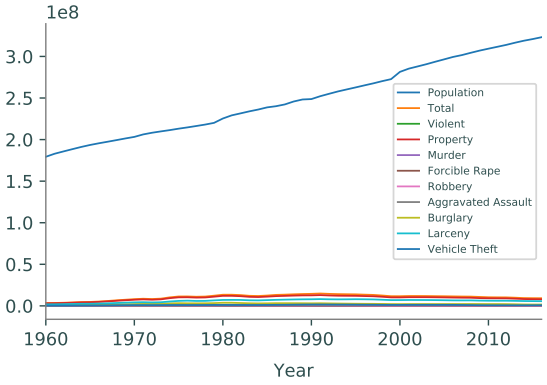
The `plot()` method also takes in many keyword arguments for matplotlib plotting and annotation functions. For example, setting `legend=False` disables the legend, providing a value for `title` sets the figure title, `grid=True` turns a grid on, and so on. For more customizations, see `https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html`.

Visualizing an Entire Data Set

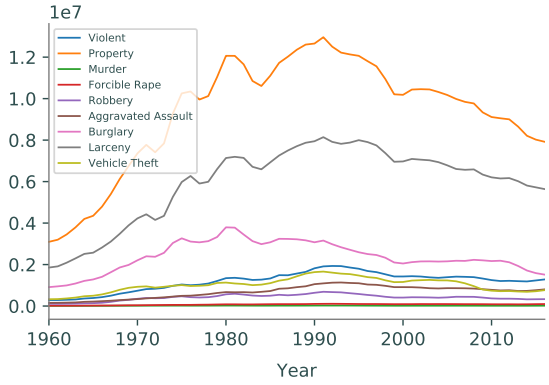
A good way to start analyzing an unfamiliar data set is to visualize as much of the data as possible to determine which parts are most important or interesting. For example, since the columns in a `DataFrame` share the same index, the columns can all be graphed together using the index as the x-axis. In fact, the `plot()` method attempts by default to plot **every** `Series` (column) in the `DataFrame`. This is especially useful with sequential data, like the crime data set.

The crime data set has 11 columns, so the resulting figure, Figure 8.1a, is fairly cluttered. However, it does shows that the `"Population"` column is on a completely different scale than the others. Dropping a few columns gives a better overview of the data, shown in Figure 8.1b.

```
# Plot all columns together against the index.
>>> crime.plot(linewidth=1)

# Plot all columns together except for 'Population' and 'Total'.
>>> crime.drop(["Population", "Total"], axis=1).plot(linewidth=1)
```



(a) All columns of the crime data set on the same figure, using the index as the x-axis.



(b) All columns of the crime data set except `"Population"` and `"Total"`.

Figure 8.1

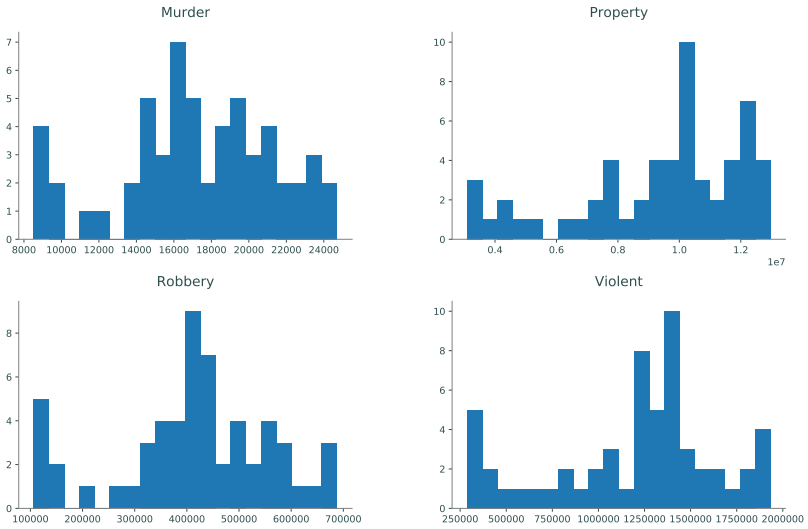
> **Achtung!**
>
> The `"Population"` column differs from the other columns because it is has **different units of measure**: population is measured by "number of people," but all other columns are measured in "number of crimes." Be careful not to plot parts of a data set together if those parts don't have the same units or are otherwise incomparable.

To quickly plot several columns in separate subplots, use `subplots=True` and specify a shape tuple as the `layout` for the plots. Subplots automatically share the same x-axis; set `sharey=True` to force them to share the same y-axis as well.

```
>>> crime.plot(y=["Property", "Larceny", "Burglary", "Violent"],
...            subplots=True, layout=(2,2), sharey=True,
...            style=['-', '--', '-.', ':'])
```

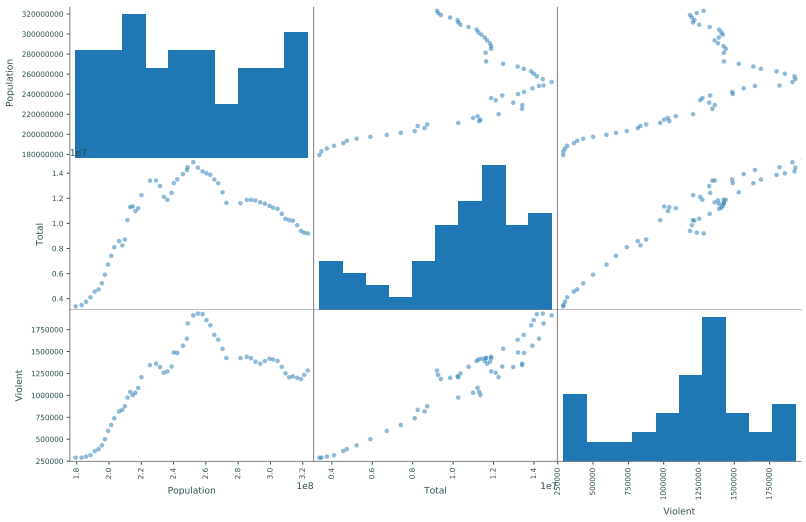
The `plot()` method can generate subplots of any kind of plot. However, since subplots share an x-axis by default, histograms turn out poorly whenever there are columns with very different data ranges. For histograms, use the `hist()` method of the `DataFrame` instead of the `plot()` method. Specify the number of bins with the `bins` parameter.

```
>>> crime[["Violent","Murder","Robbery","Property"]].hist(grid=False, bins=20)
```



Finally, the function `pd.plotting.scatter_matrix()` produces a table of plots where each column is plotted against each other column in separate scatter plots. The plots on the diagonal, instead of plotting a column against itself, displays a histogram of that column. This provides a way to very quickly do an initial analysis of the correlation between different columns.

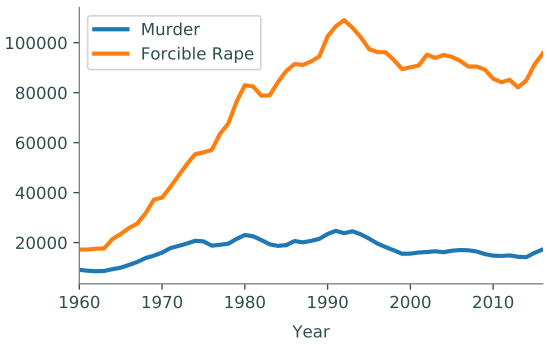
```
>>> pd.plotting.scatter_matrix(crime[["Population", "Total", "Violent"]])
```

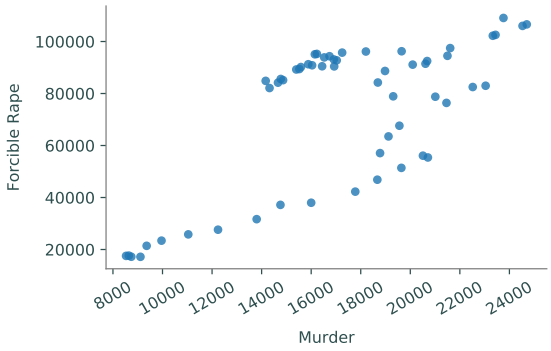


Patterns and Correlations

After visualizing the entire data set initially, the next step is usually to closely compare related parts of the data. For example, Figure 8.1b suggests that the `"Murder"` and `"Forcible Rape"` columns are roughly on the same scale. Since this data is sequential (indexed by time), start by plotting these two columns against the index. Next, create a scatter plot of one of the columns versus the other to investigate correlations that are independent of the index. Unlike other types of plots, using `kind="scatter"` requires both an `x` and a `y` column as arguments.

```
# Plot 'Murder' and 'Forcible Rape' as lines against the index.
>>> crime.plot(y=["Murder", "Forcible Rape"])

# Make a scatter plot of 'Murder' against 'Forcible Rape', ignoring the index.
>>> crime.plot(kind="scatter", x="Murder", y="Forcible Rape", alpha=.8, rot=30)
```





What does these graphs show about the data? First of all, rape is more common than murder. Second, rates of rape appear to be steadily increased from the mid 1960's to the mid 1990's before leveling out, while murder rates stay relatively constant. The disparity between rape and murder is confirmed in the scatter plot: the clump of data points at about 15,000 murders and 90,000 rapes shows that there have been many years where rape was relatively high while murder was somewhat low.

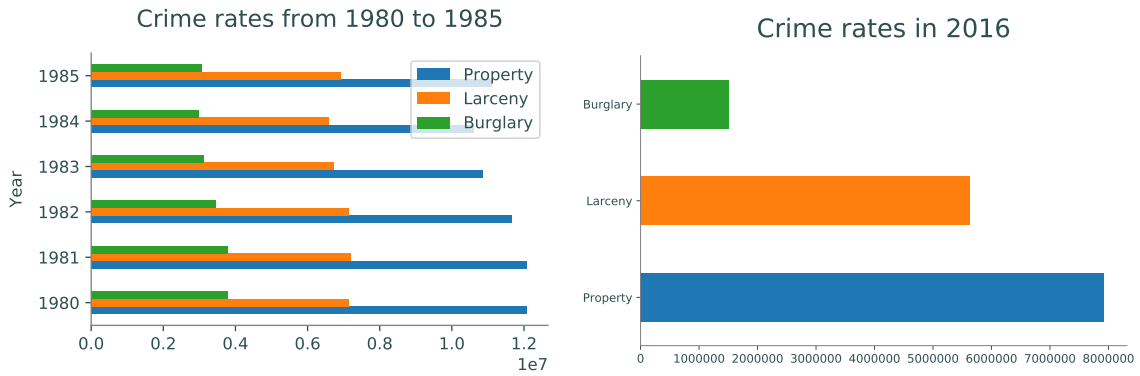
Achtung!

While analyzing data, especially while searching for patterns and correlations, **always** ask yourself if the data makes sense and is trustworthy. What lurking variables could have influenced the data measurements as they were being gathered?

The crime data set is somewhat suspect in this regard. The murder rate is likely accurate, since murder is conspicuous and highly reported, but what about the rape rate? Are the number of rapes increasing, or is the percentage of rapes being reported increasing? (It's probably both!) Be careful about drawing conclusions for sensitive or questionable data.

Figure 8.1b also reveals some general patterns relative to time. For instance, there seems to be a small bump in each type of crime in the early 1980's. Slicing the entries from 1980 to 1985 provides a closer look. Since there are only a few entries, we can treat the data as categorical and make a bar chart.

```
# Plot 'Property' and 'Larceny' rates from 1980 to 1985.
>>> crime.loc[1980:1985, ["Property", "Larceny", "Burglary"]].plot(kind="barh",
...                       title="Crime rates from 1980 to 1985")

# Plot the most recent year's crime rates for comparison.
>>> crime.iloc[-1][["Property", "Larceny", "Burglary"]].plot(kind="barh",
...                       title="Crime rates in 2016", color=["C0", "C1", "C2"])
>>> plt.tight_layout()
```

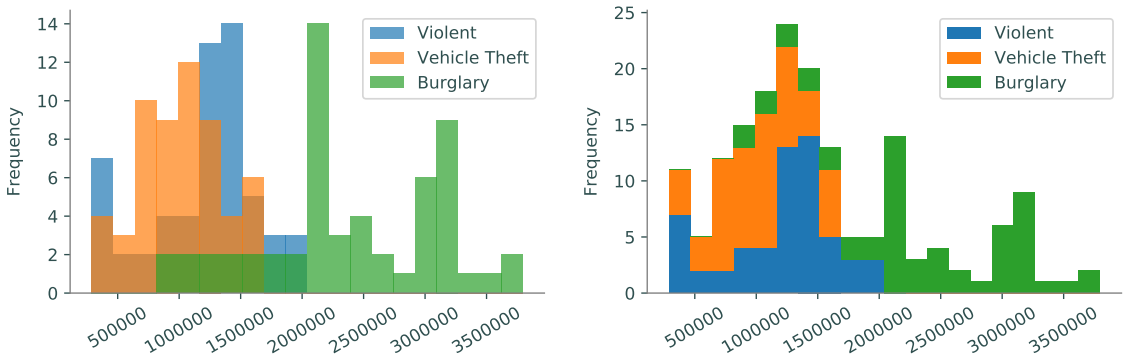


> **Note**
>
> As a general rule, horizontal bar charts (`kind="hbar"`) are better than the default vertical bar charts (`kind="bar"`) because most humans can detect horizontal differences more easily than vertical differences. If the labels are too long to fit on a normal figure, use `plt.tight_layout()` to adjust the plot boundaries to fit the labels in.

Distributional Visualizations

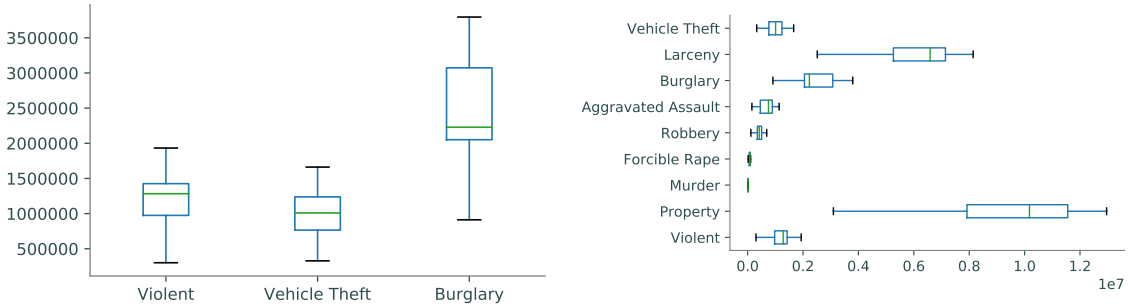
Histograms are good for examining the distribution of a **single** column in a data set. While pandas is capable of plotting several histograms on the same plot, the results are usually hard to read.

```
# Plot three histograms together.
>>> crime.plot(kind="hist", y=["Violent", "Vehicle Theft", "Burglary"],
...            bins=20, alpha=.7, rot=30)

# Plot three histograms, stacking one on top of the other.
>>> crime.plot(kind="hist", y=["Violent", "Vehicle Theft", "Burglary"],
...            bins=20, stacked=True, rot=30)
```



Instead of using histograms to compare distributions of data, use box plots. A *box plot* (sometimes called a "cat-and-whisker" plot) shows the five number summary: the minimum, first quartile, median, third quartile, and maximum of the data. While not quite the same as a histogram, box plots are much better suited to quickly compare relatable distributions.

```
# Compare the distributions of three columns.
>>> crime.plot(kind="box", y=["Violent", "Vehicle Theft", "Burglary"])

# Compare the distributions of all columns but 'Population' and 'Total'.
>>> crime.drop(["Population", "Total"], axis=1).plot(kind="box", vert=False)
```

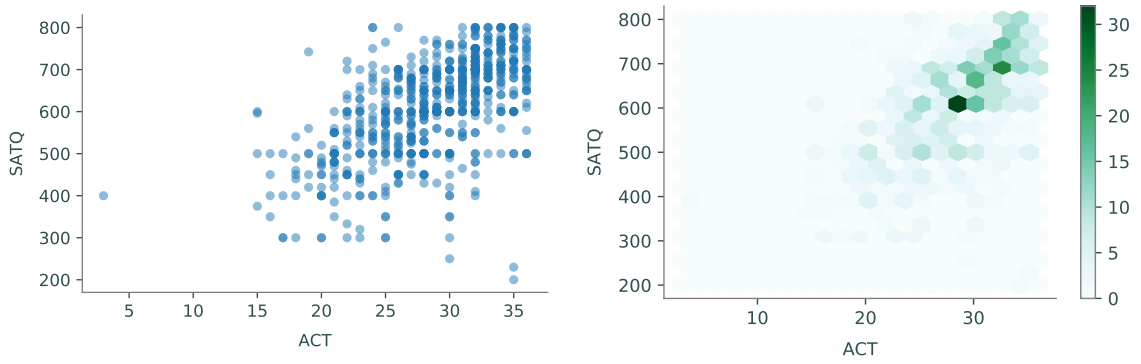


Hexbin Plots

A scatter plot is essentially a plot of samples from the joint distribution of two columns. However, scatter plots can be uninformative for large data sets when the points in a scatter plot are closely clustered. *Hexbin plots* solve this problem by plotting point density in hexagonal bins—essentially creating a 2-dimensional histogram.

The file `sat_act.csv` contains 700 self reported scores on the SAT Verbal, SAT Quantitative and ACT, collected as part of the Synthetic Aperture Personality Assessment (SAPA) web based personality assessment project. The obvious question with this data set is "how correlated are ACT and SAT scores?" The scatter plot of ACT scores versus SAT Quantitative scores, Figure 8.6a, is highly cluttered, even though the points have some transparency. A hexbin plot of the same data, Figure 8.6b, reveals the **frequency** of points in binned regions.

```
>>> satact = pd.read_csv("sat_act.csv", index_col="ID")
>>> list(satact.columns)
['gender', 'education', 'age', 'ACT', 'SATV', 'SATQ']

# Plot the ACT scores against the SAT Quant scores in a regular scatter plot.
>>> satact.plot(kind="scatter", x="ACT", y="SATQ", alpha=.8)

# Plot the densities of the ACT vs. SATQ scores with a hexbin plot.
>>> satact.plot(kind="Hexbin", x="ACT", y="SATQ", gridsize=20)
```



(a) ACT vs. SAT Quant scores.

(b) Frequency of ACT vs. SAT Quant scores.

Figure 8.6

Just as choosing a good number of `bins` is important for a good histogram, choosing a good `gridsize` is crucial for an informative hexbin plot. A large `gridsize` creates many small bins and a small `gridsize` creates fewer, larger bins.

See `http://pandas.pydata.org/pandas-docs/stable/visualization.html` for more types of plots available in Pandas and further examples.

Principles of Good Data Visualization

Visualization is much more than a set of pretty pictures scattered throughout a paper for the sole purpose of providing contrast to the text. When properly implemented, data visualization is a powerful tool for analysis and communication. When writing a paper or report, the author must make many decisions about how to use graphics effectively to convey useful information to the reader. Here we will go over a simple process for making deliberate, effective, and efficient design decisions.

Attention to Detail

Consider the plot in Figure 8.7. What does it depict? We can tell from a simple glance that it is a scatter plot of positively correlated data of some kind, with `temp`–likely temperature–on the x axis and `cons` on the y axis. However, the picture is not really communicating anything about the dataset. We have not specified the units for the x or the y axis, we have no idea what `cons` is, there is no title, and we don't even know where the data came from in the first place.

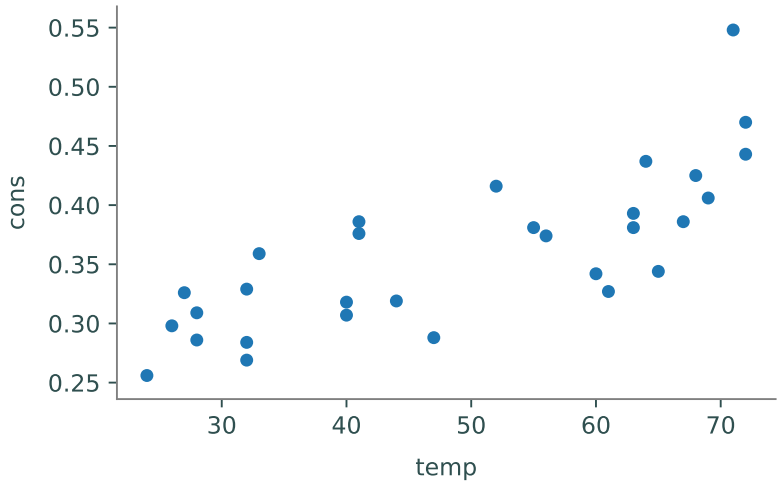


Figure 8.7: Non-specific data.

Labels and Citations

In a homework or lab setting, we sometimes (mistakenly) think that it is acceptable to leave off appropriate labels, legends, titles, and sourcing. In a published report or presentation, this kind of carelessness is confusing at best and, when the source is not included, even plagiaristic. Clearly, we need to explain our data in a useful manner that includes all of the vital information.

Consider again Figure 8.7. This figure comes from the `Icecream` dataset within the `pydataset` package, which we store here in a dataframe and then plot:

```
>>> from pydataset import data
>>> icecream = data("Icecream")
>>> icecream.plot(kind="scatter", x="temp", y="cons")
```

We have at this point reproduced the rather substandard plot in Figure 8.7. Using `data('Icecream', show_doc=True)` we find the following information:

1. The dataset details ice cream consumption via four-weekly observations from March 1951 to July 1953 in the United States.

2. `cons` corresponds to "consumption of ice cream per head" and is measured in pints.

3. `temp` corresponds to temperature, degrees Fahrenheit.

4. The listed source is: "Hildreth, C. and J. Lu (1960) *Demand relations with autocorrelated disturbances*, Technical Bulletin No 2765, Michigan State University."

We add these important details using the following code. As we have seen in previous examples, pandas automatically generates legends when appropriate. However, although pandas also automatically labels the x and y axes, our data frame column titles may be insufficient. Appropriate titles for the x and y axes must also list appropriate units. For example, the y axis should specify that the consumption is in units of *pints per head*, in place of the ambiguous label `cons`.

```
>>> icecream = data("Icecream")
# Set title via the title keyword argument
>>> icecream.plot(kind="scatter", x="temp", y="cons", title="Ice Cream ↩
    Consumption in the U.S., 1951-1953",)
# Override pandas automatic labelling using xlabel and ylabel
>>> plt.xlabel("Temp (Farenheit)")
>>> plt.ylabel("Consumption per head (pints)")
```

To arbitrarily add the necessary text to the figure, use either `plt.annotate()` or `plt.text()`. Alternatively, add text immediately below wherever the figure is displayed.

```
>>> plt.text(20, .1, r"Source: Hildreth, C. and J. Lu (1960) \emph{Demand"
...      "relations with autocorrelated disturbances}\nTechnical Bulletin No"
...      "2765, Michigan State University.", fontsize=7)
```

Both of these methods are imperfect, however, and can normally be just as easily replaced by a caption attached to the figure in your presentation or document setting. We again reiterate how important it is that you source any data you use. Failing to do so is plagiarism.

Finally, we have a clear and demonstrative graphic in Figure 8.8.

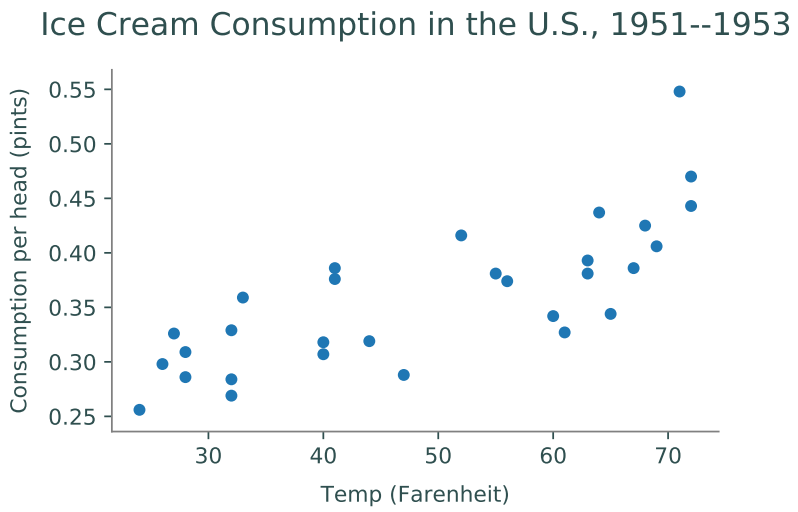


Figure 8.8: Source: Hildreth, C. and J. Lu (1960) *Demand relations with autocorrelated disturbances*, Technical Bulletin No 2765, Michigan State University.

Problem 1. The `pydataset` module[a] contains numerous data sets, each stored as a pandas `DataFrame`.

```
>>> from pydataset import data

# Call data() to see the entire list of data sets.
```

```
# To load a particular data set, enter its ID as an argument to data().
>>> titanic = data("Titanic")
# To see the information about a data set, call data() with show_doc=True.
>>> data("Titanic", show_doc=True)
Titanic

PyDataset Documentation (adopted from R Documentation. The displayed ↩
    examples
are in R)

## Survival of passengers on the Titanic
```

Visualize and describe at least 5 of the following data sets with 2 or 3 figures each. Comment on the implications and significance of each visualization and give a comprehensive summary of the data set.

- `"Arbuthnot"`: Ratios of male to female births in London from 1629-1710
- `"trees"`: Girth, height and volume for black cherry trees
- `"road"`: Road accident deaths in the United States
- `"birthdeathrates"`: Birth and death rates by country
- `"bfeed"`: Child breast feeding records
- `"heart"`: Survival of patients on the waiting list for the Stanford heart transplant program
- `"lung"`: Survival in patients with advanced lung cancer from the North Central Cancer Treatment group
- `"birthwt"`: Risk factors associated with low infant birth weight
- A data set of your choice

Include each of the following in each visualization.

- A clear title, with relevant information for the period or region the data was collected in.
- Axis labels that specify units.
- A legend (if appropriate).
- The source. You may include the source information in your plot or print it after the plot.

[a] Run `pip install pydataset` if needed.