In [1]:
```
import matplotlib
from datascience import *
%matplotlib inline
import matplotlib.pyplot as plots
import numpy as np
plots.style.use('fivethirtyeight')
```

Lecture 26

Chebyshev's Bounds

In [2]:
```
births = Table.read_table('baby.csv')
```

In [3]:
```
births.labels
```

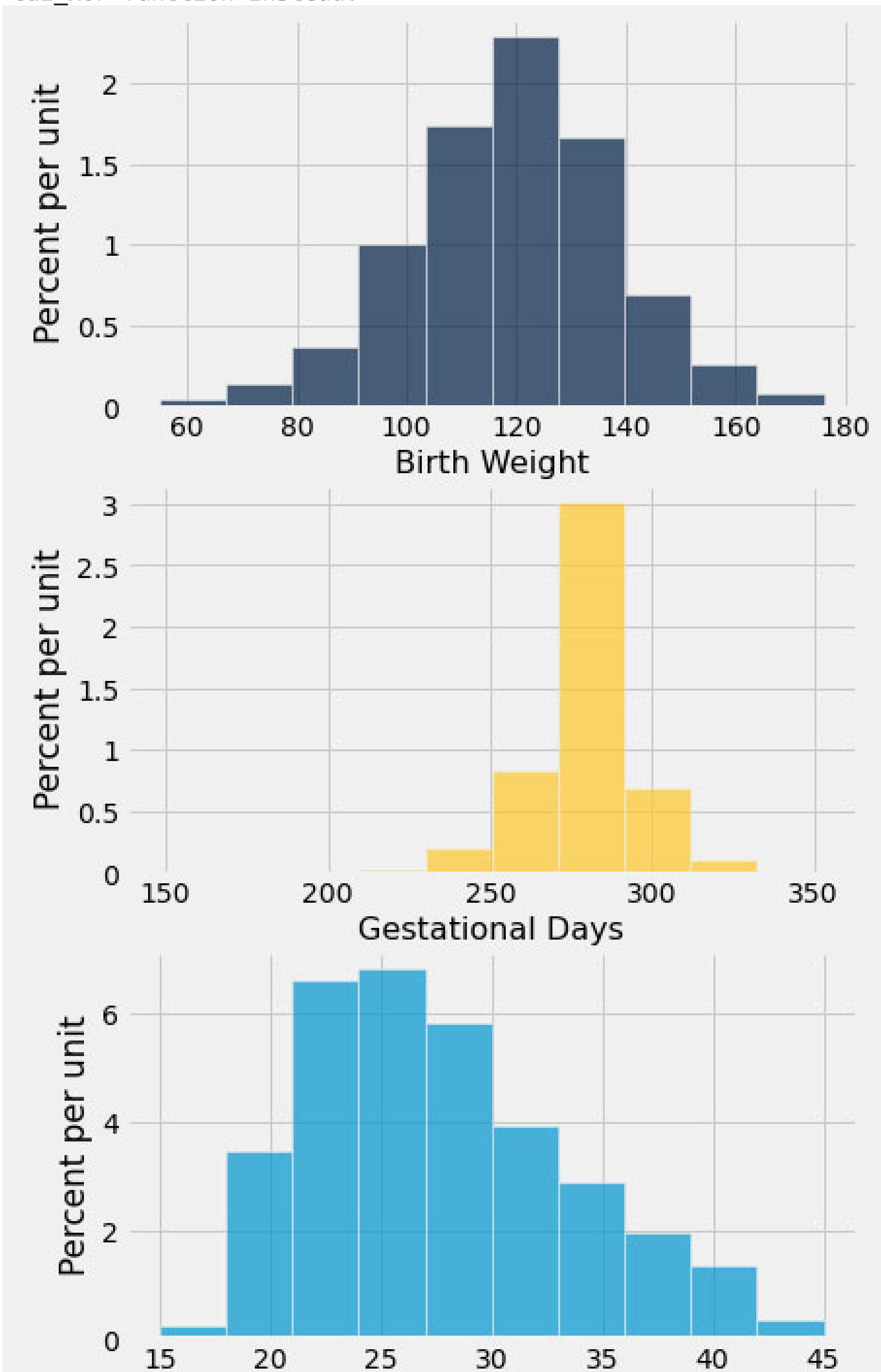
Out[3]:
```
('Birth Weight',
 'Gestational Days',
 'Maternal Age',
 'Maternal Height',
 'Maternal Pregnancy Weight',
 'Maternal Smoker')
```

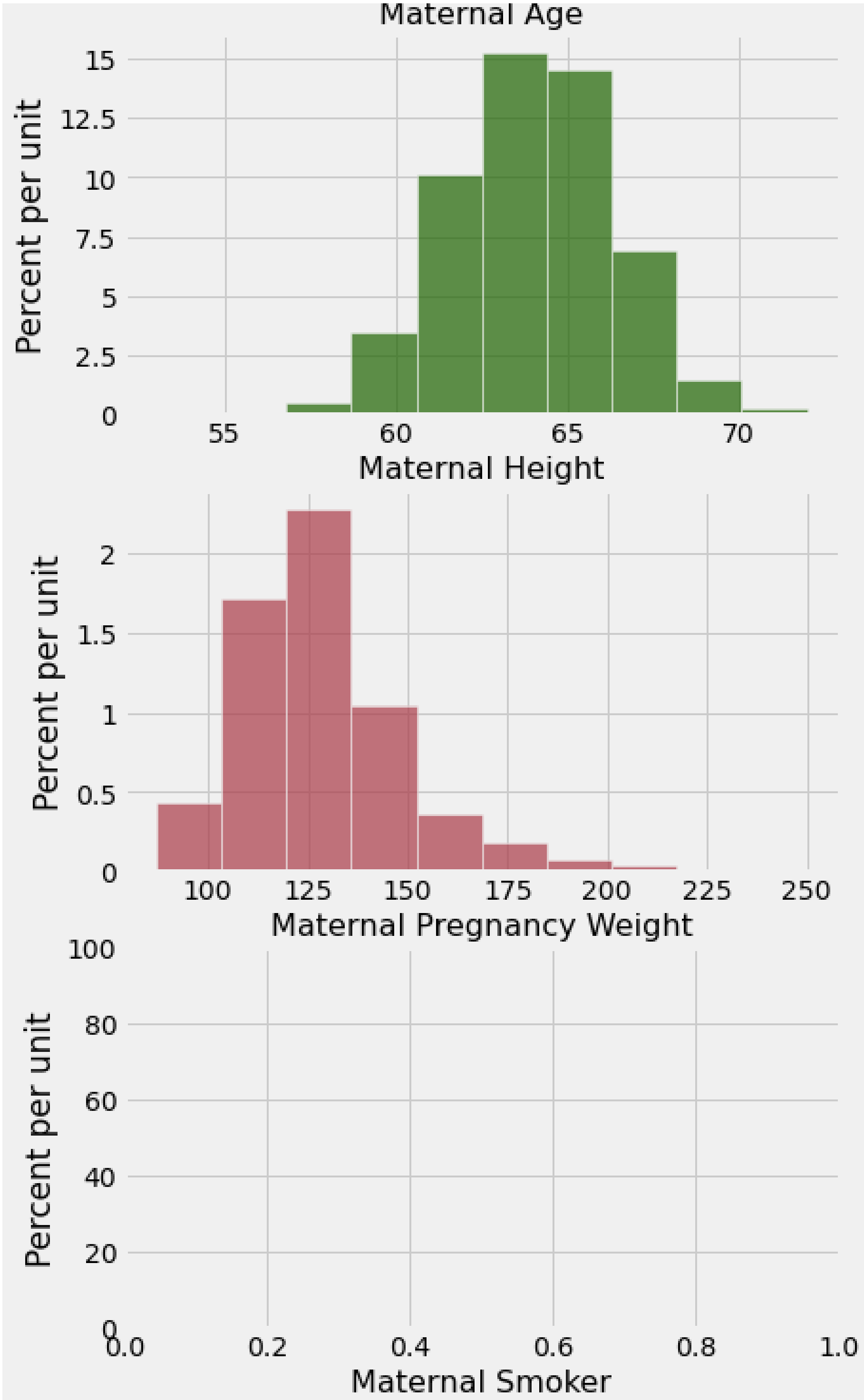
In [4]:
```
births.hist(overlay = False)
```

```
<__array_function__ internals>:5: RuntimeWarning: Converting input from bool to <class 'numpy.uint8'> for compatibility.
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\Anaconda3\lib\site-packages\numpy\lib\histograms.py in _unsigned_subtract(a, b)
    350     try:
--> 351         dt = signed_to_unsigned[dt.type]
    352     except KeyError:

KeyError: <class 'numpy.bool_'>

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
<ipython-input-4-bd2fcacff5c8> in <module>
----> 1 births.hist(overlay = False)
```

```
~\Anaconda3\lib\site-packages\datascience\tables.py in hist(self, overlay, bins, bin_column, unit, counts, group, rug, si
de_by_side, left_end, right_end, width, height, *columns, **vargs)
   4874                     type(self).plots.append(axis)
   4875
-> 4876         draw_hist(values_dict)
   4877
   4878     def hist_of_counts(self, *columns, overlay=True, bins=None, bin_column=None,

~\Anaconda3\lib\site-packages\datascience\tables.py in draw_hist(values_dict)
   4863                         vargs['weights'] = weights[i]
   4864                     axis.set_xlabel(hist_name + x_unit, fontsize=16)
-> 4865                     heights, bins, patches = axis.hist(values_for_hist, color=color, **vargs)
   4866                     if left_end is not None and right_end is not None:
   4867                         x_shade, height_shade, width_shade = _compute_shading(heights, bins.copy(), left_end,
right_end)

~\Anaconda3\lib\site-packages\matplotlib\__init__.py in inner(ax, data, *args, **kwargs)
   1445     def inner(ax, *args, data=None, **kwargs):
   1446         if data is None:
-> 1447             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1448
   1449         bound = new_sig.bind(ax, *args, **kwargs)

~\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py in hist(self, x, bins, range, density, weights, cumulative, botto
m, histtype, align, orientation, rwidth, log, color, label, stacked, **kwargs)
   6649             # this will automatically overwrite bins,
   6650             # so that each histogram uses the same bins
-> 6651             m, bins = np.histogram(x[i], bins, weights=w[i], **hist_kwargs)
   6652             tops.append(m)
   6653         tops = np.array(tops, float)  # causes problems later if it's an int

<__array_function__ internals> in histogram(*args, **kwargs)

~\Anaconda3\lib\site-packages\numpy\lib\histograms.py in histogram(a, bins, range, normed, weights, density)
    820
    821         # Pre-compute histogram scaling factor
--> 822         norm = n_equal_bins / _unsigned_subtract(last_edge, first_edge)
    823
    824         # We iterate over blocks here for two reasons: the first is that for

~\Anaconda3\lib\site-packages\numpy\lib\histograms.py in _unsigned_subtract(a, b)
    351         dt = signed_to_unsigned[dt.type]
    352     except KeyError:
--> 353         return np.subtract(a, b, dtype=dt)
    354     else:
    355         # we know the inputs are integers, and we are deliberately casting
```

```
TypeError: numpy boolean subtract, the `-` operator, is not supported, use the bitwise_xor, the `^` operator, or the logi
cal_xor function instead.
```




Percent per unit
15
12.5
10
7.5
5
2.5
0
55
60
65
70
Maternal Height
Percent per unit
2
1.5
1
0.5
0
100
125
150
175
200
225
250
Maternal Pregnancy Weight
100
80
60
40
20
0
Percent per unit
0.0
0.2
0.4
0.6
0.8
1.0
Maternal Smoker

In [5]:
```
mpw = births.column('Maternal Pregnancy Weight')
mean = np.mean(mpw)
sd = np.std(mpw)
mean, sd
```

Out[5]:
```
(128.4787052810903, 20.72544970428041)
```

In [6]:
```
within_3_SDs = births.where('Maternal Pregnancy Weight', are.between(mean - 3*sd, mean + 3*sd))
```

In [7]:
```
within_3_SDs.num_rows/births.num_rows
```

Out[7]:
```
0.9863713798977853
```

In [8]:
```
1 - 1/9
```

Out[8]:
```
0.8888888888888888
```

In [9]:
```
# See if Chebyshev's bounds work for different distributions

for k in births.labels:
    values = births.column(k)
    mean = np.mean(values)
    sd = np.std(values)
    print()
    print(k)
    for z in np.arange(2, 6):
        chosen = births.where(k, are.between(mean - z*sd, mean + z*sd))
        proportion = chosen.num_rows/births.num_rows
        percent = round(proportion * 100, 2)
        print('Mean plus or minus', z, 'SDs:', percent, '%')
```

```
Birth Weight
Mean plus or minus 2 SDs: 94.89 %
Mean plus or minus 3 SDs: 99.57 %
Mean plus or minus 4 SDs: 100.0 %
Mean plus or minus 5 SDs: 100.0 %

Gestational Days
```

```
Mean plus or minus 2 SDs: 93.78 %
Mean plus or minus 3 SDs: 98.64 %
Mean plus or minus 4 SDs: 99.57 %
Mean plus or minus 5 SDs: 99.83 %

Maternal Age
Mean plus or minus 2 SDs: 94.89 %
Mean plus or minus 3 SDs: 99.91 %
Mean plus or minus 4 SDs: 100.0 %
Mean plus or minus 5 SDs: 100.0 %

Maternal Height
Mean plus or minus 2 SDs: 97.19 %
Mean plus or minus 3 SDs: 99.66 %
Mean plus or minus 4 SDs: 99.91 %
Mean plus or minus 5 SDs: 100.0 %

Maternal Pregnancy Weight
Mean plus or minus 2 SDs: 95.06 %
Mean plus or minus 3 SDs: 98.64 %
Mean plus or minus 4 SDs: 99.49 %
Mean plus or minus 5 SDs: 99.91 %

Maternal Smoker
Mean plus or minus 2 SDs: 100.0 %
Mean plus or minus 3 SDs: 100.0 %
Mean plus or minus 4 SDs: 100.0 %
Mean plus or minus 5 SDs: 100.0 %
```

Standard Units

In [10]:
```
def standard_units(x):
    """Convert array x to standard units."""
    return (x - np.mean(x))/np.std(x)
```

In [11]:
```
ages = births.column('Maternal Age')
```

In [12]:
```
ages_standard_units = standard_units(ages)
```

In [13]:
```
np.mean(ages_standard_units), np.std(ages_standard_units)
```

Out[13]: (-7.868020072300939e-17, 1.0)

In [14]:
```
both = Table().with_columns(
    'Age in Years', ages,
    'Age in Standard Units', ages_standard_units
)
both
```

Out[14]:

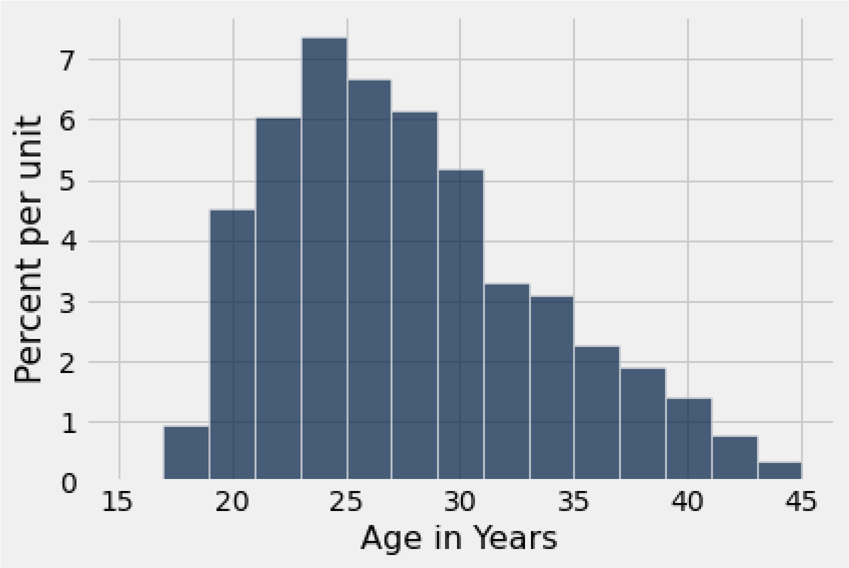
Age in Years	Age in Standard Units
27	-0.0392546
33	0.992496
28	0.132704
23	-0.727088
25	-0.383171
33	0.992496
23	-0.727088
25	-0.383171
30	0.476621
27	-0.0392546

... (1164 rows omitted)

In [15]:
```
np.mean(ages), np.std(ages)
```

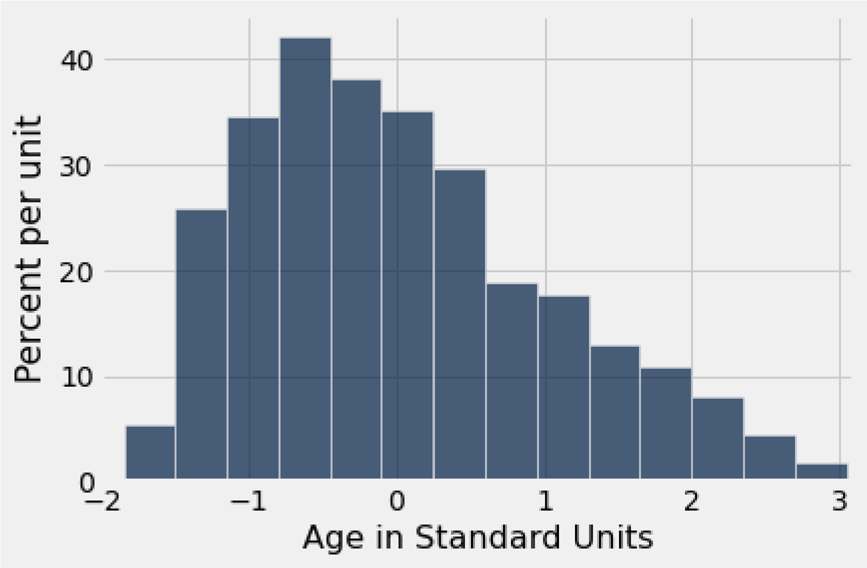
Out[15]: (27.228279386712096, 5.815360404190897)

In [16]:
```
both.hist('Age in Years', bins = np.arange(15, 46, 2))
```



In [17]:

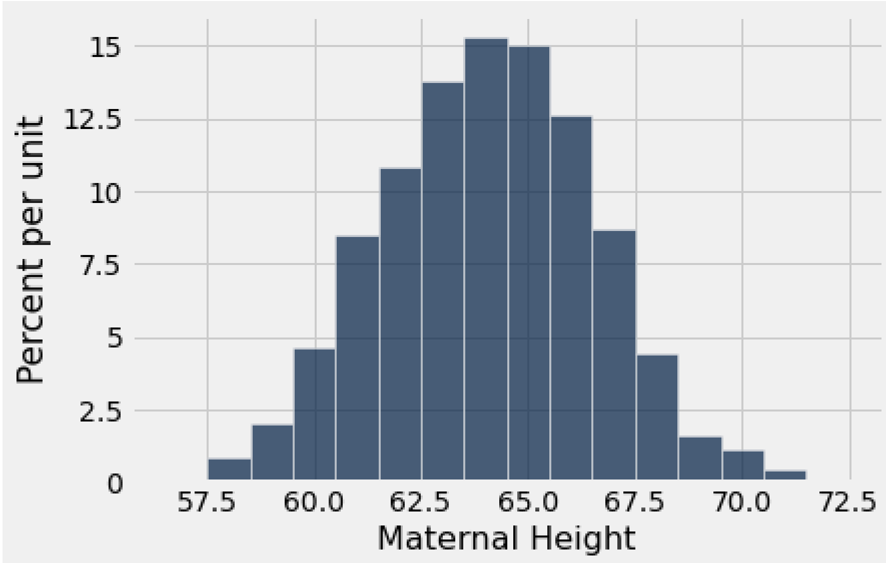
```
both.hist('Age in Standard Units', bins = np.arange(-2.2, 3.4, 0.35))
plots.xlim(-2, 3.1);
```



The SD and Bell Shaped Curves

In [18]:

```
births.hist('Maternal Height', bins = np.arange(56.5, 72.6, 1))
```



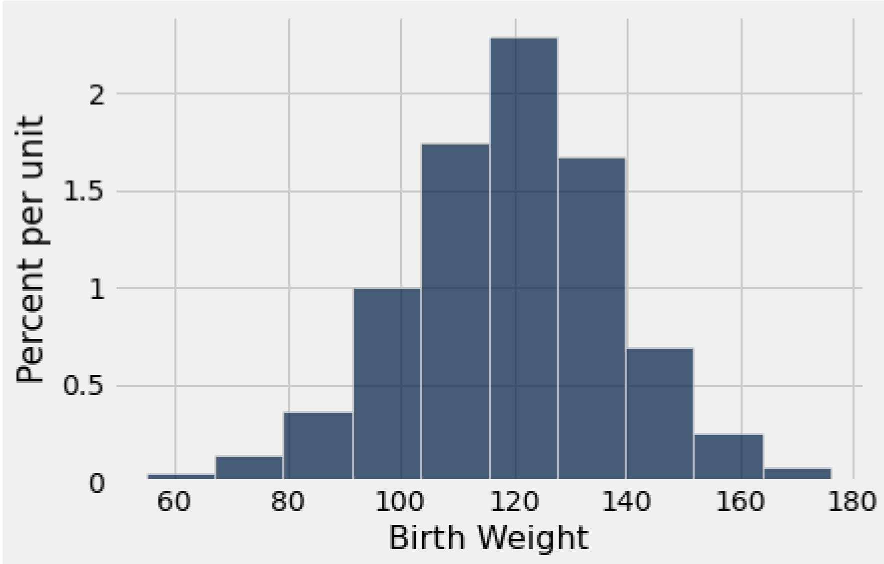
In [19]:

```
heights = births.column('Maternal Height')
np.mean(heights), np.std(heights)
```

Out[19]: (64.04940374787053, 2.5250254409674375)

In [20]:

```
births.hist('Birth Weight')
```



In [21]:

```
bw = births.column('Birth Weight')
mean_w = np.mean(bw)
sd_w = np.std(bw)
mean_w, sd_w
```

Out[21]: (119.46252129471891, 18.32086370220278)

The Normal curve

In [22]:

```
red_winnings = np.append(1*np.ones(18), -1*np.ones(20))
red = Table().with_columns('Winnings on Red', red_winnings)
```

In [23]:

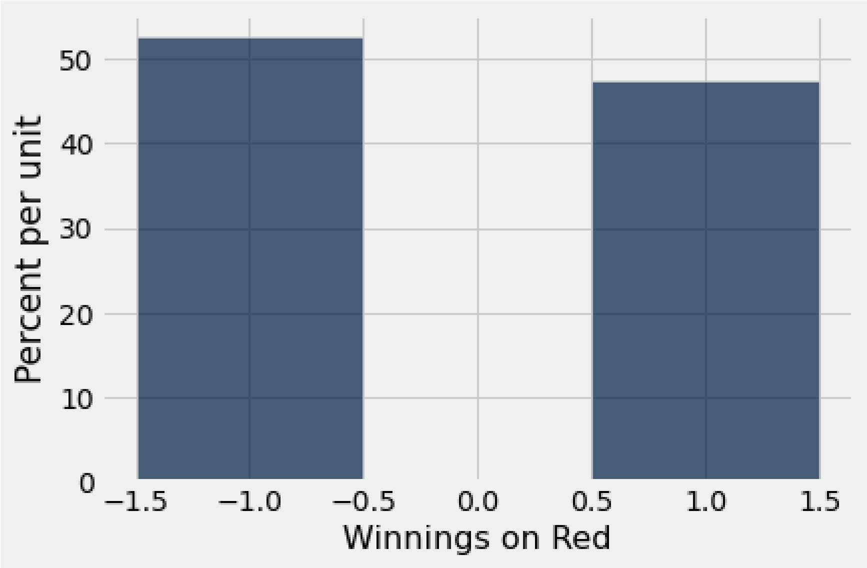
```
red.show()
```

Winnings on Red
1
1
1

Winnings on Red
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1

Winnings on Red
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1

In [24]:
```
red.hist(bins = np.arange(-1.5, 1.6, 1))
```

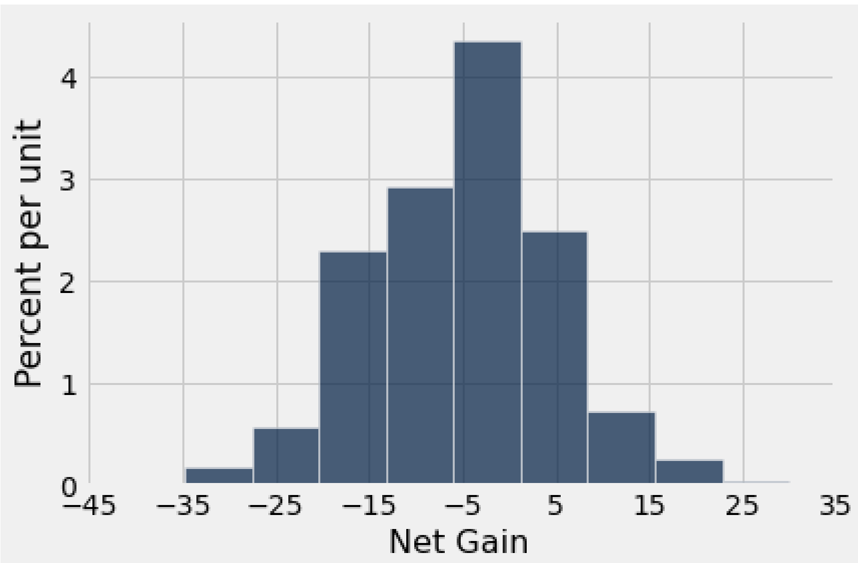


In [25]:
```
18/38
```

Out[25]: 0.47368421052631576

In [26]:
```
num_bets = 100

net_gains = make_array()

for i in np.arange(20000):
    spins = red.sample(num_bets)
    new_net_gain = sum(spins.column('Winnings on Red'))
    net_gains = np.append(net_gains, new_net_gain)
```

In [27]:
```
Table().with_columns('Net Gain', net_gains).hist()
plots.xticks(np.arange(-45, 36, 10));
```



In [28]:
```
np.average(net_gains)
```

Out[28]: -5.3489

Central Limit Theorem and Simulating Sample Mean

In [29]:

```python
united = Table.read_table('united_summer2015.csv')
united
```

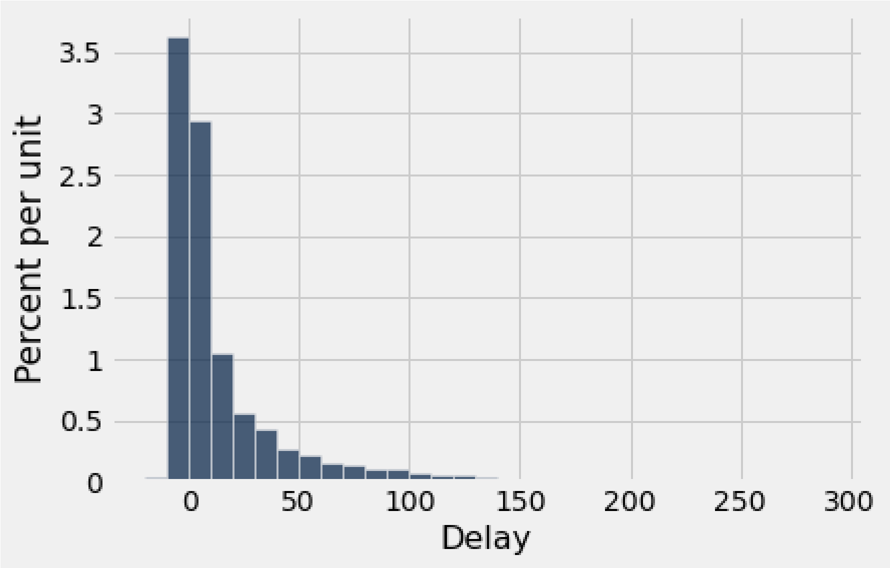
Out[29]:

Date	Flight Number	Destination	Delay
6/1/15	73	HNL	257
6/1/15	217	EWR	28
6/1/15	237	STL	-3
6/1/15	250	SAN	0
6/1/15	267	PHL	64
6/1/15	273	SEA	-6
6/1/15	278	SEA	-8
6/1/15	292	EWR	12
6/1/15	300	HNL	20
6/1/15	317	IND	-10

... (13815 rows omitted)

In [30]:

```python
united.hist('Delay', bins = np.arange(-20, 300, 10))
```



In [31]:

```
delays = united.column('Delay')
mean_delay = np.mean(delays)
sd_delay = np.std(delays)

mean_delay, sd_delay
```

Out[31]: (16.658155515370705, 39.480199851609314)

In [32]:

```
united = united.with_columns(
    'Delay in Standard Units', standard_units(delays)
)
united.sort('Delay', descending=True)
```

Out[32]:

Date	Flight Number	Destination	Delay	Delay in Standard Units
6/21/15	1964	SEA	580	14.269
6/22/15	300	HNL	537	13.1798
6/21/15	1149	IAD	508	12.4453
6/20/15	353	ORD	505	12.3693
8/23/15	1589	ORD	458	11.1788

Date	Flight Number	Destination	Delay	Delay in Standard Units
7/23/15	1960	LAX	438	10.6722
6/23/15	1606	ORD	430	10.4696
6/4/15	1743	LAX	408	9.91236
6/17/15	1122	HNL	405	9.83637
7/27/15	572	ORD	385	9.32979

... (13815 rows omitted)

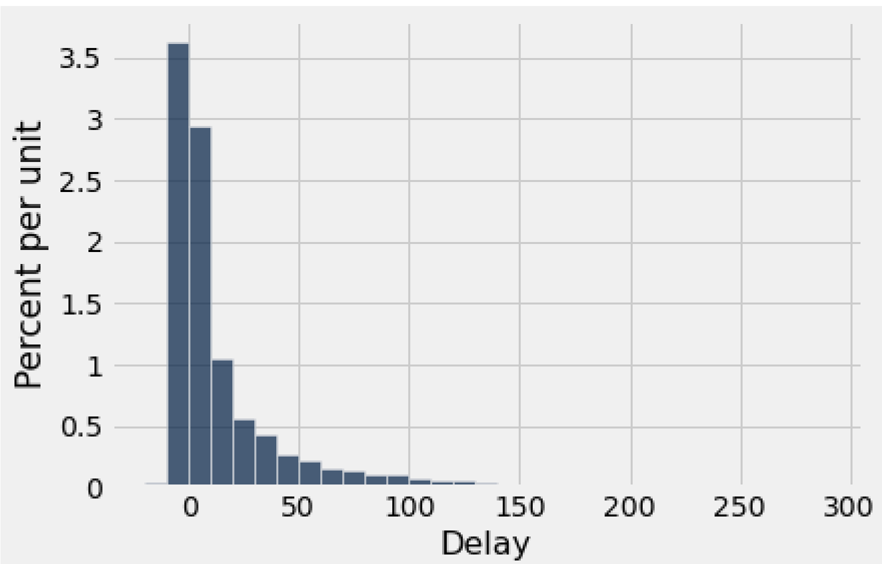
In [33]:

```
chosen = united.where('Delay in Standard Units', are.between(-3, 3))
chosen.num_rows/united.num_rows
```

Out[33]: 0.9790235081374322

In [34]:

```
united.hist('Delay', bins = np.arange(-20, 300, 10))
```



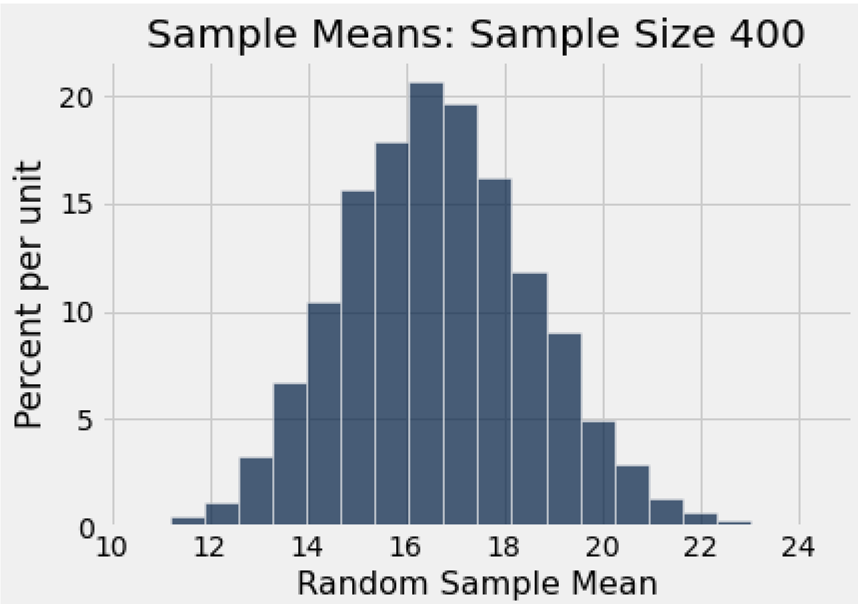
In [35]:

```
sample_size = 400

means = make_array()
```

```
for i in np.arange(10000):
    sampled_flights = united.sample(sample_size)
    sample_mean = np.mean(sampled_flights.column('Delay'))
    means = np.append(means, sample_mean)
```

In [36]:

```
Table().with_columns('Sample Mean', means).hist(bins = 20)
plots.title('Sample Means: Sample Size ' + str(sample_size))
plots.xlabel('Random Sample Mean');
```



In [37]:

```
#report the mean and the standard deviation of the
#sample means of 10000 samples of size 400

mean_of_sample_means = np.mean(means)
sd_of_sample_means = np.std(means)
mean_of_sample_means, sd_of_sample_means
```

Out[37]: (16.650805249999998, 1.950262306639145)

In [38]:

```
#to empirical illustrate/validate the Central Limit Theorem
#compare the mean of the 13,000+ delays
```

```
#with the mean of the 10,000 means of random samples of size 400
mean_delay, mean_of_sample_means
```

Out[38]: (16.658155515370705, 16.650805249999998)

In [39]:

```
#and compare the
#standard deviation of the 13,000+ delays / sqrt(400)
#with the
#standard deviation of the 10,000 means of random samples of size 400

import math
sd_delay/math.sqrt(400), sd_of_sample_means
```

Out[39]: (1.9740099925804657, 1.950262306639145)

In [ ]:

```

```