Statistics

Refresher, and Python Practice

Calculating mean using the NumPy package

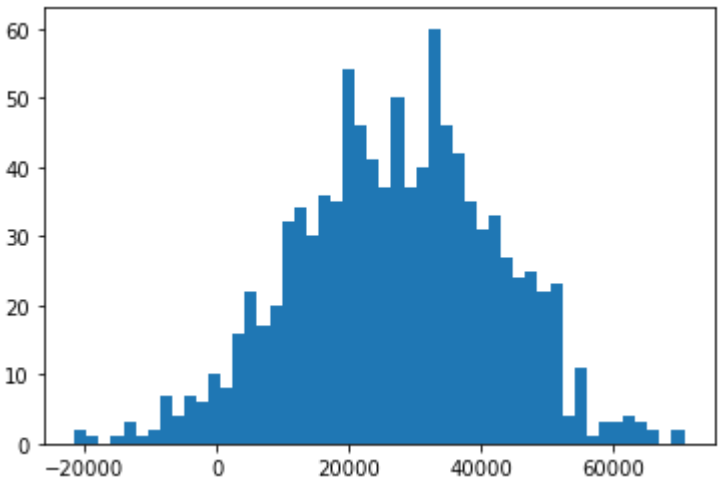
In [2]:
```
import numpy as np
incomes=np.random.normal(27000,15000,1000)
```

In [3]:
```
np.mean(incomes)
```

Out[3]:
```
27106.24031545101
```

Visualizing data using matplotlib

In [10]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.hist(incomes,50)
plt.show()
```



numpy.ptp

numpy.ptp(a, axis=None, out=None, keepdims=)[source] Range of values (maximum - minimum) along an axis.

The name of the function comes from the acronym for ‘peak to peak

In [11]:
```
x = np.array([[4, 9, 2, 10],
              [6, 9, 7, 12]])
```

In [13]:
```
np.ptp(x,axis=1)
```

Out[13]:
```
array([8, 6])
```

In [14]:
```
np.ptp(x,axis=0)
```

Out[14]:
```
array([2, 0, 5, 2])
```

In [15]:
```
np.ptp(x)
```

Out[15]:
```
10
```

This example shows that a negative value can be returned when the input is an array of signed integers.

In [16]:
```
y = np.array([[1, 127],
              [0, 127],
              [-1, 127],
              [-2, 127]], dtype=np.int8)
np.ptp(y, axis=1)
```

Out[16]:
```
array([ 126,  127, -128, -127], dtype=int8)
```

A work-around is to use the view() method to view the result as unsigned integers with the same bit width:

In [18]:
```
np.ptp(y, axis=1).view(np.uint8)
```

Out[18]:
```
array([126, 127, 128, 129], dtype=uint8)
```

numpy.percentile

numpy.percentile(a, q, axis=None, out=None, overwrite_input=False, method='linear', keepdims=False, *, interpolation=None)

In [54]:
```
a = np.array([[10, 7, 4], [3, 2, 1]])
a
```

Out[54]:
```
array([[10,  7,  4],
       [ 3,  2,  1]])
```

In [55]:
```
np.percentile(a, 50)
```

Out[55]:
```
3.5
```

In [56]:
```
np.percentile(a, 50, axis=0)
```

Out[56]:
```
array([6.5, 4.5, 2.5])
```

In [57]:
```
np.percentile(a, 50, axis=1)
```

Out[57]:
```
array([7., 2.])
```

In [58]:
```
np.percentile(a, 50, axis=1, keepdims=True)
```

Out[58]:
```
array([[7.],
       [2.]])
```

In [59]:
```
m = np.percentile(a, 50, axis=0)
out = np.zeros_like(m)
np.percentile(a, 50, axis=0, out=out)
```

Out[59]:
```
array([6.5, 4.5, 2.5])
```

In [60]:
```
b = a.copy()
np.percentile(b, 50, axis=1, overwrite_input=True)
```

Out[60]:
```
array([7., 2.])
```

The different methods can be visualized graphically

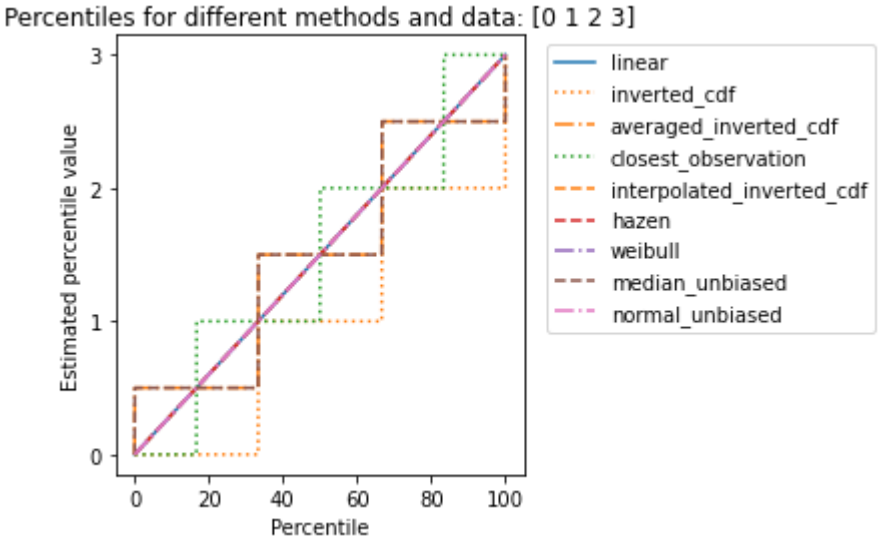
In [79]:
```
import numpy as np
import matplotlib.pyplot as plt

a = np.arange(4)
p = np.linspace(0, 100, 6001)
ax = plt.gca()
lines = [
    ('linear', '-', 'C0'),
    ('inverted_cdf', ':', 'C1'),
    # Almost the same as `inverted_cdf':
    ('averaged_inverted_cdf', '-.', 'C1'),
    ('closest_observation', ':', 'C2'),
    ('interpolated_inverted_cdf', '--', 'C1'),
    ('hazen', '--', 'C3'),
    ('weibull', '-.', 'C4'),
    ('median_unbiased', '--', 'C5'),
    ('normal_unbiased', '-.', 'C6'),
]

for method, style, color in lines:
    if method == 'linear':
        percentiles = np.percentile(a, p)  # Default method is linear
    elif method == 'inverted_cdf':
        percentiles = np.percentile(a, p, interpolation='lower')
    elif method == 'averaged_inverted_cdf':
        percentiles = np.percentile(a, p, interpolation='midpoint')
    elif method == 'closest_observation':
        percentiles = np.percentile(a, p, interpolation='nearest')
    elif method == 'interpolated_inverted_cdf':
        percentiles = np.percentile(a, p, interpolation='linear')
    elif method == 'hazen':
        # Implement your custom method for 'hazen' here
        # You can replace this line with your custom calculation
        percentiles = np.percentile(a, p)
    elif method == 'weibull':
        # Implement your custom method for 'weibull' here
        # You can replace this line with your custom calculation
        percentiles = np.percentile(a, p)
    elif method == 'median_unbiased':
        percentiles = np.percentile(a, p, interpolation='midpoint')
    elif method == 'normal_unbiased':
        percentiles = np.percentile(a, p, interpolation='linear')

    ax.plot(p, percentiles, label=method, linestyle=style, color=color)

ax.set(
    title='Percentiles for different methods and data: ' + str(a),
    xlabel='Percentile',
    ylabel='Estimated percentile value',
    yticks=a)
ax.legend(bbox_to_anchor=(1.03, 1))
plt.tight_layout()
plt.show()
```



In [22]:
```
a=np.array([[10.,7.,4.],[3.,2.,1.]])
a[1][0]=np.nan
a
```

Out[22]:
```
array([[10.,  7.,  4.],
       [nan,  2.,  1.]])
```

In [21]:
```
a = np.array([[10., 7., 4.], [3., 2., 1.]])
a[0][1] = np.nan
a
```

Out[21]:
```
array([[10., nan,  4.],
       [ 3.,  2.,  1.]])
```

In [23]:
```
np.percentile(a,50)
```

Out[23]:
```
nan
```

In [24]:
```
np.nanpercentile(a,50)
```

Out[24]:
```
4.0
```

In [25]:
```
np.nanpercentile(a,50,axis=0)
```

Out[25]:
```
array([10. ,  4.5,  2.5])
```

In [26]:
```
np.nanpercentile(a, 50, axis=1, keepdims=True)
```

Out[26]:
```
array([[7. ],
       [1.5]])
```

In [27]:
```
m = np.nanpercentile(a, 50, axis=0)
```

In [28]:
```
m
```

Out[28]:
```
array([10. ,  4.5,  2.5])
```

In [32]:
```
m = np.nanpercentile(a, 50, axis=0)
out = np.zeros_like(m)
np.nanpercentile(a, 50, axis=0, out=out)
```

Out[32]:
```
array([10. ,  4.5,  2.5])
```

In [33]:
```
b = a.copy()
np.nanpercentile(b, 50, axis=1, overwrite_input=True)
```

Out[33]:
```
array([7. , 1.5])
```

In [34]:
```
assert not np.all(a==b)
```

numpy.quantile

numpy.quantile(a, q, axis=None, out=None, overwrite_input=False, method='linear', keepdims=False, *, interpolation=None)

In [44]:
```
a=np.array([[10.,7.,4.],[3.,2.,1]])
a
```

Out[44]:
```
array([[10.,  7.,  4.],
       [ 3.,  2.,  1.]])
```

In [45]:
```
np.quantile(a, 0.5)
```

Out[45]:
```
3.5
```

In [46]:
```
np.quantile(a, 0.5, axis=0)
```

Out[46]:
```
array([6.5, 4.5, 2.5])
```

In [47]:
```
np.quantile(a, 0.5, axis=1)
```

Out[47]:
```
array([7., 2.])
```

In [49]:
```
np.quantile(a, 0.5, axis=1,keepdims=True)
```

Out[49]:
```
array([[7.],
       [2.]])
```

In [50]:
```
np.quantile(a, 0.5, axis=0)
```

Out[50]:
```
array([6.5, 4.5, 2.5])
```

In [51]:
```
m = np.quantile(a, 0.5, axis=0)
out = np.zeros_like(m)
np.quantile(a, 0.5, axis=0, out=out)
```

Out[51]:
```
array([6.5, 4.5, 2.5])
```

In [52]:
```
b = a.copy()
np.quantile(b, 0.5, axis=1, overwrite_input=True)
```

Out[52]:
```
array([7., 2.])
```

In [53]:
```
m
```

Out[53]:
```
array([6.5, 4.5, 2.5])
```

numpy.nanquantile

numpy.nanquantile(a, q, axis=None, out=None, overwrite_input=False, method='linear', keepdims=, *, interpolation=None)

In [80]:
```
a = np.array([[10., 7., 4.], [3., 2., 1.]])
a[0][1] = np.nan
a
```

Out[80]:
```
array([[10., nan,  4.],
       [ 3.,  2.,  1.]])
```

In [81]:
```
np.quantile(a, 0.5)
```

Out[81]:
```
nan
```

In [82]:
```
np.nanquantile(a, 0.5)
```

Out[82]:
```
3.0
```

In [83]:
```
np.nanquantile(a, 0.5, axis=0)
```

Out[83]:
```
array([6.5, 2. , 2.5])
```

In [84]:
```
np.nanquantile(a, 0.5, axis=1, keepdims=True)
```

Out[84]:
```
array([[7.],
       [2.]])
```

In [85]:
```
m = np.nanquantile(a, 0.5, axis=0)
out = np.zeros_like(m)
np.nanquantile(a, 0.5, axis=0, out=out)
```

Out[85]:
```
array([6.5, 2. , 2.5])
```

In [86]:
```
m
```

Out[86]:
```
array([6.5, 2. , 2.5])
```

In [87]:
```
b = a.copy()
np.nanquantile(b, 0.5, axis=1, overwrite_input=True)
```

Out[87]:
```
array([7., 2.])
```

In [*]:
```
# Import pandas library
import pandas as pd

# Read Data
data = pd.read_csv("StudentsPerformance.csv")
data
```

In [*]:
```
data.info()
```

In [*]:
```
data.describe()
```

In [ ]:
```

```