Asda

|  |  |  |  |
| --- | --- | --- | --- |
|  | sampling | Creating X&Ys | Plotting |
|  | **np.random.binomial**(4, 0.5, size=10) | ecdf() | In [5]: \_ = plt.plot(x, y, marker='.', linestyle='none') |
|  | **np.random.poisson**(6, size=10000) |  |
|  | **np.random.normal**(mean, std, size=10000) | ecdf() | \_ = plt.plot(x\_theor, y\_theor)  \_ = plt.plot(x, y, marker='.', linestyle='none') |
|  | **np.random.exponential**(mean, size=10000) | ecdf() |
| Linear regression by least squares | slope, intercept = **np.polyfit**(total\_votes, dem\_share, 1) | x = np.array([0, 100])  y = a \* x + b | \_ = plt.plot(x, y) |
| Bootstrap confidence intervals | **np.random.choice**(michelson\_speed\_of\_light, size=100) |  | \_ = plt.hist(bs\_replicates, bins=30, normed=True) |
| Bootstrap confidence interval | In : conf\_int = **np.percentile**(bs\_replicates, [2.5, 97.5])  Out: array([ 299837., 299868.]) |  |  |
| Generating a pairs bootstrap sample | slope, intercept = **np.polyfit** (bs\_total\_votes, bs\_dem\_share, 1)  slope, intercept = np.polyfit(total\_votes, dem\_share, 1) | x = np.array([0, 100])  y = a \* x + b | \_ = plt.plot(x, y) |
| Generating permutation replicates | **np.concatenate**( )  **np.random.permutation** ( )  slicing: [:len(dem\_share\_PA)] / (dem\_share\_PA):] |  |  |
| Probability | prob=np.sum(samples <= 144)/len(samples) |  |  |
|  | def ecdf(data):  """Compute ECDF for a one-dimensional array of measurements."""  # Number of data points: n  n=len(data)  # x-data for the ECDF: x  x=np.sort(data)  # y-data for the ECDF: y  y = np.arange(1, len(x)+1) / n  return x, y |  |  |
|  | def bootstrap\_replicate\_1d(data, func):  ...: """Generate bootstrap replicate of 1D data."""  ...: bs\_sample = np.random.choice(data, len(data))  ...: return func(bs\_sample)  In [1]: bs\_replicates = np.empty(10000)  In [2]: for i in range(10000):  ...: bs\_replicates[i] = bootstrap\_replicate\_1d(  ...: michelson\_speed\_of\_light, np.mean) |  |  |
| Computing a pairs bootstrap replicate | In [1]: inds = np.arange(len(total\_votes))  In [2]: bs\_inds = np.random.choice(inds, len(inds))  In [3]: bs\_total\_votes = total\_votes[bs\_inds]  In [4]: bs\_dem\_share = dem\_share[bs\_inds]  In [1]: bs\_slope, bs\_intercept = np.polyfit(bs\_total\_votes,  ...: bs\_dem\_share, 1)  In [2]: bs\_slope, bs\_intercept  Out[2]: (3.9053605692223672e-05, 40.387910131803025)  In [3]: np.polyfit(total\_votes, dem\_share, 1) # fit of original  Out[3]: array([ 4.03707170e-05, 4.01139120e+01]) |  |  |
| Plotting bootstrap regressions | # Generate array of x-values for bootstrap lines: x  x = np.array([0,100])  for i in range(100):  \_ = plt.plot(x, bs\_slope\_reps[i]\*x + bs\_intercept\_reps[i],  linewidth=0.5, alpha=0.2, color='red') |  |  |
| Generating permutation replicates | def permutation\_sample(data1, data2):  data = np.concatenate((data1, data2))  permuted\_data = np.random.permutation(data)  perm\_sample\_1 = permuted\_data[:len(data1)]  perm\_sample\_2 = permuted\_data[len(data1):]  return perm\_sample\_1, perm\_sample\_2 |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |