DS1EDP: Homework 06 – Solutions

1. Plot the Vote

Question 1:

```
def proportions_in_resamples():
    prop_c = make_array()
    for i in np.arange(5000):
        bootstrap = votes.sample()
        single_proportion = np.count_nonzero(bootstrap.column(0) == 'C')
         / bootstrap.num_rows
        prop_c = np.append(prop_c, single_proportion)
    return prop_c
```

Question 2:

```
c_lower_bound = percentile(2.5, sampled_proportions)
c_upper_bound = percentile(97.5, sampled_proportions)
```

Question 3:

```
bins = np.arange(-0.2,0.2,0.01)

def leads_in_resamples():
    leads = make_array()
    for i in np.arange(5000):
        bootstrap = votes.sample()
        c = np.count_nonzero(bootstrap.column(0) == 'C') / bootstrap.num_rows
        t = np.count_nonzero(bootstrap.column(0) == 'T') / bootstrap.num_rows
        leads = np.append(leads, c-t)
    return leads

sampled_leads = leads_in_resamples()
Table().with_column('leads', sampled_leads).hist(bins=bins)

diff_lower_bound = percentile(2.5, sampled_leads)
diff_upper_bound = percentile(97.5, sampled_leads)
```

2. Interpreting Confidence Intervals

Question 1:

```
correct_option = 2
```

Question 2:

```
true_proportion_intervals = 9500
```

Question 4:

```
confidence_interval_80 = interval_2
confidence_interval_90 = interval_1
confidence_interval_99 = interval_3
```

Question 5:

```
candidates_tied = 2
```

Question 6:

```
cutoff_one_percent = 2
```

Question 7:

```
cutoff_ten_percent = 3
```

3. Triple Jump Distances vs. Vertical Jump Heights

Note:

For the following questions we are using the functions **f_standard_units, f_correlation, f_slope** and **f_intercept**. These functions were not given and you had to define them yourself.

```
def f_standard_units(any_numbers):
    return (any_numbers - np.mean(any_numbers))/np.std(any_numbers)

def f_correlation(t, label_x, label_y):
    return np.mean(f_standard_units(t.column(label_x)) *
      f_standard_units(t.column(label_y)))

def f_slope(t, label_x, label_y):
    r = f_correlation(t, label_x, label_y)
    return r*np.std(t.column(label_y))/np.std(t.column(label_x))

def f_intercept(t, label_x, label_y):
    return np.mean(t.column(label_y)) - f_slope(t,label_x,label_y) *
      np.mean(t.column(label_x))
```

Question 1:

```
jumps.scatter("triple")
linear_correlation = True
```

Question 2:

```
r_estimate = 0.5
```

Question 3:

```
def regression_parameters(t):
    r = f_correlation(jumps, 0, 1)
    slope = f_slope(jumps, 0, 1)
    intercept = f_intercept(jumps, 0, 1)
    return make_array(r, slope, intercept)

parameters = regression_parameters(jumps)
print('r:', parameters.item(0), '; slope:', parameters.item(1), '; intercept:',
parameters.item(2))
```

Question 4:

```
triple_record_vert_est = f_slope(jumps, 0, 1) * 1829 + f_intercept(jumps, 0, 1)
print("Predicted vertical jump distance: {:f}
centimeters".format(triple_record_vert_est))
```

Question 5:

```
estimation_accurate = False
```

4. Cryptocurrencies

Question 1:

```
## Using functions introduced in section 3
r = f_correlation(Table().with_columns("btc", btc.column("open"), "eth",
eth.column("open")), 0, 1)

## Without fucntions introduced in section 3
def std_units(arr):
    return (arr - np.mean(arr))/np.std(arr)

standard_btc = std_units(btc.column('open'))
standard_eth = std_units(eth.column('open'))

r = np.mean(standard_btc * standard_eth)
```

Question 2:

```
def eth_predictor(btc_price):
    table = Table().with_columns("btc", btc.column("open"), "eth",
      eth.column("open"))
    parameters = regression_parameters(table)
    slope = parameters.item(1)
    intercept = parameters.item(2)
    return slope * btc_price + intercept
```

Question 3:

```
predictions = eth_predictor(btc.column('open'))
prices_with_predictions = Table().with_columns(
    "btc", btc.column("open"),
    "eth", eth.column("open"),
    "eth_prediction", predictions)
```

Question 6:

```
regression_changes = [False, True, True]
```