assignment3

March 1, 2021

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

[1]:
```
NAME = "Cody Lange"
COLLABORATORS = ""
```

1 Presenting Uncertainty

1.1 School of Information, University of Michigan

1.2 Week 3: Assignment Overview

Version 1.1 ### The objectives for this week are for you to: - learn how to construct hypothetical outcome plots (HOPs) and spaghetti plots for a fit line - practice making HOPs and spaghetti plots on Boston Housing Prices dataset

[2]:
```
import time
import altair as alt
import pandas as pd
import ipywidgets as widgets
from ipywidgets import interact
from sklearn import linear_model
from sklearn import gaussian_process
import numpy as np

import operator
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

2 Part 1: Learn to plot HOPs and spaghetti plots for linear regression (12 points)

The following salary dataset describes the relationship between someone's salary and the number of years of experience someone has. In this section, we will construct an animated hypothetical

outcome plot (HOP) and a spaghetti plot of a linear regression fit to this dataset.

[3]:
```
#load dataset
salary_df = pd.read_csv("asset/Salary_Data.csv")
salary_df.head()
```

[3]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

2.1 1.1 Construct the basic building blocks of a HOPs visualization

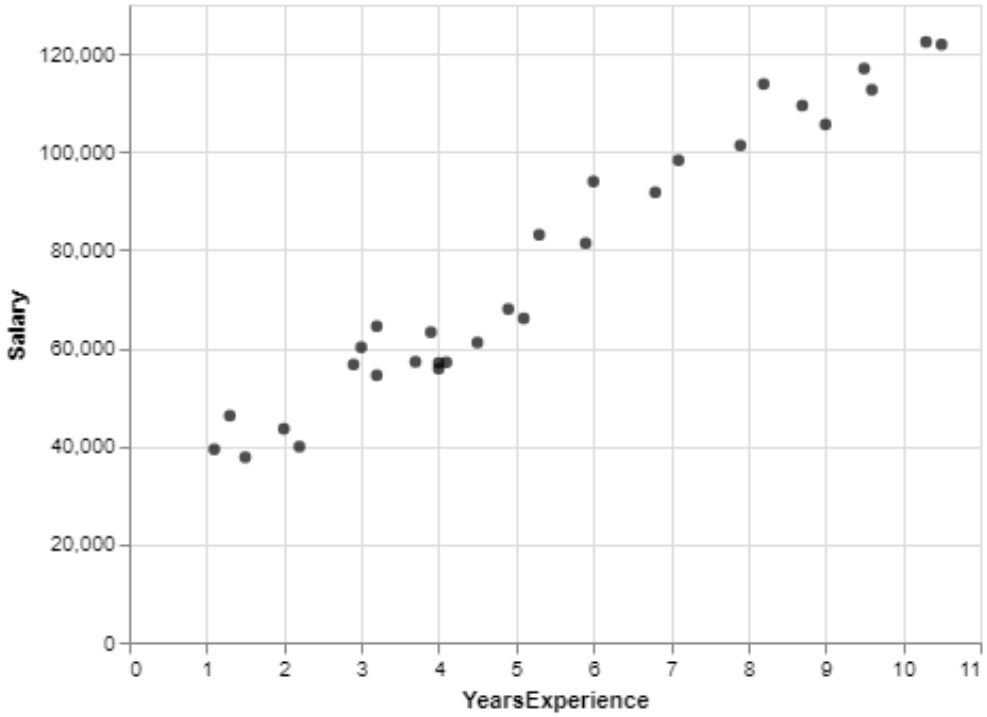
In order to construct a HOPs visualization, we need the following functions:

1. A function to construct an Altair chart of the data: `get_salary_points_chart()`
2. A function to get one bootstrap sample of the linear regression fit: `get_one_bootstrap_salary_fit()`
3. A function to construct an Altair chart of one linear regression fit line: `get_salary_linear_fit_chart()`

Then we will combine all these functions together to make an animation.

2.1.1 Question 1.1.1 Plot the data (5 points)

Construct a function, `get_salary_points_chart()`, which plots the data in `salary_df` as a scatterplot. The output should look like this:



A scatterplot of Years of Experience (x axis) against Salary (y axis)

[4]:
```
def get_salary_points_chart():
    '''
    This function should return an altair plot object that is a scatterplot of
    the salary data, with YearsExperience on the x axis and Salary on the y␣
 ↪axis
    '''
    salary_points = alt.Chart(salary_df).mark_circle(color='black').encode(
    x="YearsExperience",
    y="Salary"
    )

    return salary_points

get_salary_points_chart()
```

[4]:
```
alt.Chart(...)
```

2.1.2 Question 1.1.2 Bootstrap one linear regression fit (2 points)

We will need a function that returns one bootstrap sample of the regression fit. That is, it resamples the dataset with replacement, then fits a linear regression to the data. Fill in the code below to complete the function:

[5]:
```
def get_one_bootstrap_salary_fit():
    '''
```

```
    Returns a sklearn.linear_model.LinearRegression model representing
    a fit to a bootstrap-resampled version of salary_df
    '''

    #resample the data with replacement (replace=True) to a data frame with
    #the same number of data points (frac=1.0)
    resampled_df = salary_df.sample(frac=1.0, replace=True)

    #fit model to resampled data
    X = resampled_df[['YearsExperience']] #[[ ]] subsets so X remains a␣
↪DataFrame
    y = resampled_df['Salary']            #y should be an array, so we use [ ]

    # insert code below using LinearRegression to return a linear regression␣
↪model
    # with predictor X and outcome variable y

    lr = LinearRegression().fit(X,y)

    return lr
```

[6]:
```
get_one_bootstrap_salary_fit()
```

[6]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[7]:
```
np.random.seed(1234)
fit = get_one_bootstrap_salary_fit()
assert np.abs(fit.coef_[0] - 10004) < 0.5, "Bootstrap linear regression: slope␣
 ↪coefficient does not match the expected value"
assert np.abs(fit.intercept_ - 21485) < 0.5, "Bootstrap linear regression:␣
 ↪intercept does not match the expected value"
```

We can use this function to get a single sample from the bootstrap sampling distribution of the fit (e.g., its slope and intercept). Each time you run the following cell you should get slightly different values:

[8]:
```
salary_reg = get_one_bootstrap_salary_fit()
print("Bootstrapped intercept: ", salary_reg.intercept_)
print("Bootstrapped slope:     ", salary_reg.coef_[0])
```

```
Bootstrapped intercept:  23608.862570184552
Bootstrapped slope:      9759.092501075675
```

2.1.3 Question 1.1.3 Construct an Altair chart of one regression fit (5 points)

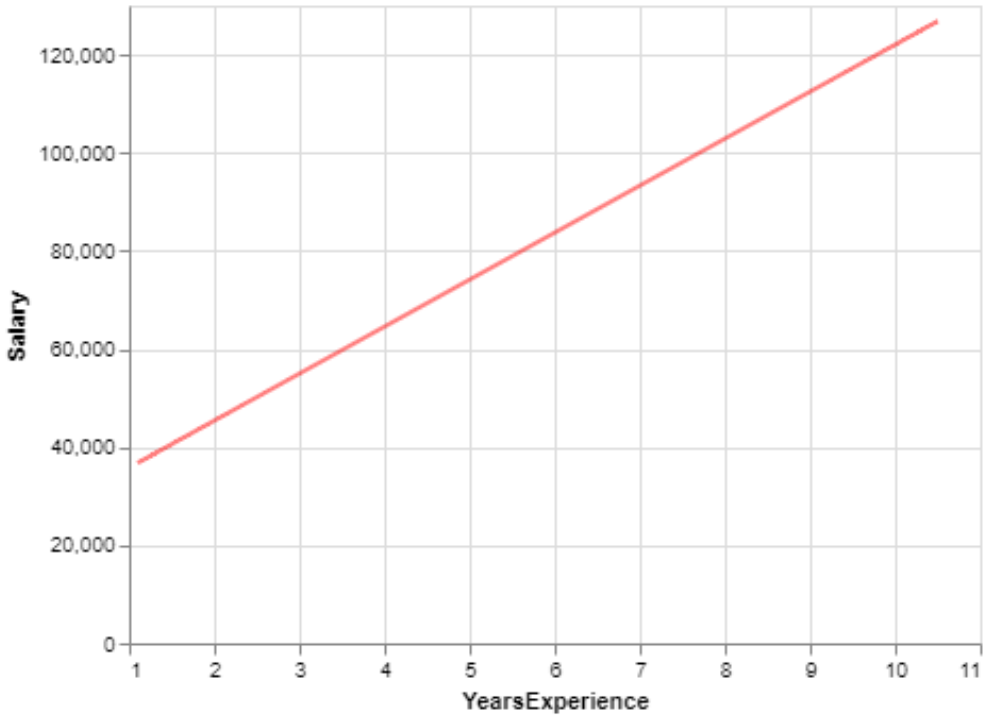
To construct a chart of a fit line or fit curve, we first need a *prediction grid*: a set of x values we want to use to make predictions. This should be in the same form as the input to the regression function (i.e., a DataFrame).

For this example, we will use evenly-spaced values of `"YearsExperience"`, the x value in our charts. Because it is a linear fit, we strictly speaking only need 2 values, but we will use more (101)

because it generalizes better. When you plot non-linear relationships (as we will in Part 2), you need a large number of points in your prediction grid so that the curve is smooth.

[9]:
```
# construct a prediction grid for the salary dataset with 101
# evenly-spaced values from the minimum to maximum number of years of␣
 ↪experience
salary_pred_grid = pd.DataFrame({'YearsExperience': np.linspace(
    salary_df['YearsExperience'].min(),
    salary_df['YearsExperience'].max(),
    num=101
)})
```

Complete the `get_salary_linear_fit_chart()` so that it displays a single fit line from the linear regression fit passed in to it. The chart should look like this:



A line chart of Years of Experience (x axis) against Salary (y axis)

[10]:
```
def get_salary_linear_fit_chart(salary_reg, opacity=0.5):
    '''
    Takes a single linear regression fit (as returned by␣
 ↪`get_one_bootstrap_salary_fit()`) and
    returns an Altair chart plotting the fit line

    Parameters:

    - salary_reg: A regression fit
    - opacity: The opacity of the output line
    '''
```

```
    #use the model to predict the mean Salary at each x position
    pred_df = pd.DataFrame({
        'YearsExperience': salary_pred_grid['YearsExperience'],
        'Salary': salary_reg.predict(salary_pred_grid)
    })

    #insert code to return an Altair chart showing the fit line using `pred_df`
    #remember to set the opacity of the line mark to the `opacity` value
    #passed into this function (e.g. `mark_line(opacity=opacity)`)
    # YOUR CODE HERE
    lr_chart = alt.Chart(pred_df).mark_line(color='red',opacity=opacity).
↪encode(x="YearsExperience", y="Salary")

    return lr_chart

get_salary_linear_fit_chart(salary_reg)
```

[10]: `alt.Chart(...)`

2.2 1.2 Construct HOPs of the salary data

Now that you have all the pieces, you should be able to put them together to construct a HOPs visualization.

First, run the following code chunk a few times: you should notice that the fit line moves each time you run it.

[11]:
```
points_chart = get_salary_points_chart()
salary_reg = get_one_bootstrap_salary_fit()
line_chart = get_salary_linear_fit_chart(salary_reg)
line_chart + points_chart
```

[11]: `alt.LayerChart(...)`

We will use the `interact()` function to run the above code to generate each frame needed in our HOPs. Run the following code, then press the Play button to start the animation:

[12]:
```
def get_one_frame(i):
    '''
    Return one frame in the animation
    '''

    time.sleep(.2)

    # get the point chart
    points_chart = get_salary_points_chart()

    # fit one bootstrap regression
    salary_reg = get_one_bootstrap_salary_fit()

    # get the line chart
```

```
    line_chart = get_salary_linear_fit_chart(salary_reg)

    #return the combined points + lines chart
    return line_chart + points_chart

interact(get_one_frame, i = widgets.Play(
    value=0,
    min=0,
    max=100,
    step=1,
    description="Press play",
    disabled=False))
```

```
interactive(children=(Play(value=0, description='Press play'), Output()), _dom_classes=('widget
```

[12]:
```
<function __main__.get_one_frame(i)>
```

2.3 1.3 Construct a spaghetti plot of the salary data

The same functions we used to make the HOPs chart above can be used to make a spaghetti plot as well. This time, we will combine all the line charts together instead of playing them frame-by-frame. First, we make a list containing all the line charts (in the `line_charts` variable), then we use `alt.layer()` to layer all of the line charts together. Finally, we add on the chart of the points:

[13]:
```
B = 50

# get `B` bootstrapped fit line charts
# Note opacity=0.1 sets the line opacity so it is easier to see the overlapping␣
 ↪lines. Make
# sure your get_salary_linear_fit_chart() function (defined above) properly␣
 ↪uses the opacity argument!
line_charts = [get_salary_linear_fit_chart(get_one_bootstrap_salary_fit(),␣
 ↪opacity=0.1) for _ in range(B)]

#combine all the line charts together and layer on the points chart
alt.layer(*line_charts) + get_salary_points_chart()
```

[13]:
```
alt.LayerChart(...)
```

3 Part 2: Spaghetti plots for Polynomial Regression (5 points)

To demonstrate the difference in how you must modify your code to fit a new model, in this section we show how to create spaghetti plots for a polynomial regression. We will follow the same steps as before:

1. A function to construct an Altair chart of the data: `get_poly_points_chart()`

2. A function to get one bootstrap sample of the linear regression fit: `get_one_bootstrap_poly_fit()`
3. A function to construct an Altair chart of one linear regression fit line: `get_poly_fit_chart()`

We will generate a dataset with two variables (x and y), then draw spaghetti plots of a polynomial fit to the dataset.

3.1 2.1 Generate dataset

First, generate the dataset:

[14]:
```
#prepare dataset
np.random.seed(42)
n = 25

original_x = 5 - 4 * np.random.normal(0, 1, n)
original_y = -2 + 3*original_x - 5*(original_x ** 2) + 7*(original_x ** 3) + np.
 ↪random.normal(0, 1000, n)

poly_df = pd.DataFrame({'x': original_x, 'y': original_y})
```

3.2 2.2 Define helper functions

We'll define the polynomial points chart and draw it:

[15]:
```
def get_poly_points_chart():
    '''
    This function should return an altair plot object that is a scatterplot of
    the salary data, with YearsExperience on the x axis and Salary on the y␣
 ↪axis
    '''
    return alt.Chart(poly_df).mark_circle(color="black").encode(
        x='x',
        y='y'
    )

get_poly_points_chart()
```

[15]:
```
alt.Chart(...)
```

Then we define the `get_one_bootstrap_poly_fit()` and `get_poly_fit_chart()` functions so we can draw a single fit:

[16]:
```
#prediction grid
poly_pred_grid = pd.DataFrame({
    "x": np.linspace(poly_df['x'].min(), poly_df['x'].max(), num=101)
})

def get_one_bootstrap_poly_fit():
    '''Get one bootstrap sampled polynomial regression fit to the data'''
```

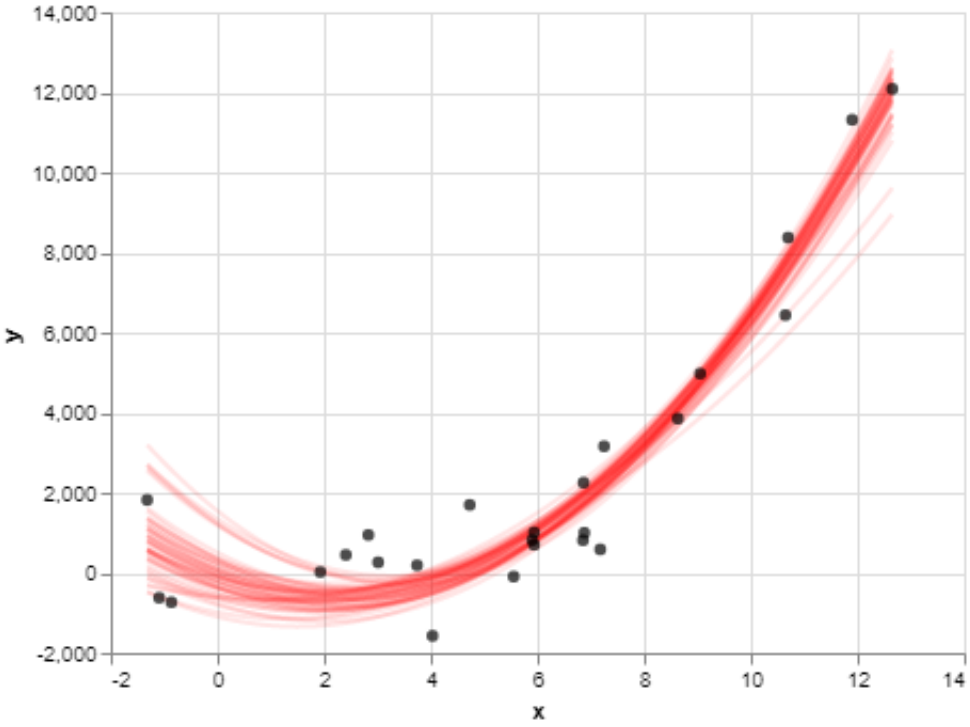
```
    #resample the data with replacement (replace=True) to a data frame with
    #the same number of data points (frac=1.0)
    resampled_df = poly_df.sample(frac=1.0, replace=True)

    #fit model to resampled data
    X = resampled_df[['x']] #[[ ]] subsets so X remains a DataFrame
    y = resampled_df['y']   #y should be an array, so we use [ ]

    #x must be transformed into polynomials (e.g. x, x^2, x^3 ... up to the␣
 ↪value of `degree`)
    polynomial_features = PolynomialFeatures(degree=2)
    X_poly = polynomial_features.fit_transform(X)
    poly_reg = linear_model.LinearRegression()
    poly_reg.fit(X_poly, y)

    return poly_reg

def get_poly_fit_chart(poly_reg, opacity=0.5):
    '''
    Takes a single polynomial regression fit (as returned by␣
 ↪`get_one_bootstrap_poly_fit()`) and
    returns an Altair chart plotting the fit curve

    Parameters:

    - poly_reg: A regression fit
    - opacity: The opacity of the output line
    '''
    #use the model to predict y at each x position
    polynomial_features = PolynomialFeatures(degree=2)
    pred_df = pd.DataFrame({
        'x': poly_pred_grid['x'],
        'y': poly_reg.predict(polynomial_features.fit_transform(poly_pred_grid))
    })

    #return an Altair chart showing the fit line
    return alt.Chart(pred_df).mark_line(
        opacity=opacity,
        color='red'
    ).encode(
        x='x',
        y='y'
    )

poly_reg = get_one_bootstrap_poly_fit()
get_poly_fit_chart(poly_reg)
```

[16]: `alt.Chart(...)`

4 2.3 Draw spaghetti plot for polynomial regression

4.0.1 Question 2.3.1 Draw a spaghetti plot for the above polynomial regression (5 points)

Using the helper functions defined above (`get_poly_points_chart()`, `get_one_bootstrap_poly_fit()`, and `get_poly_fit_chart()`), draw a spaghetti plot for the example polynomial regression data. Your output should look something like this:



Polynomial spaghetti plot fit

[17]:
```
B = 50

# get `B` bootstrapped fit line charts
# Note opacity=0.1 sets the line opacity so it is easier to see the overlapping␣
 ↪lines. Make
# sure your get_salary_linear_fit_chart() function (defined above) properly␣
 ↪uses the opacity argument!
poly_charts = [get_poly_fit_chart(get_one_bootstrap_poly_fit(), opacity=0.1)␣
 ↪for _ in range(B)]

#combine all the line charts together and layer on the points chart
alt.layer(*poly_charts) + get_poly_points_chart()
```

[17]: `alt.LayerChart(...)`

5 Part 3: Boston housing price dataset (23 points)

Apply what you have learned above to construct hypothetical outcome plots and spaghetti plots for the relationship between housing prices and the `LSTAT` variable (average of the proportion of adults without some high school education and the proportion of male workers classified as laborers). It can be seen from the below figure that LSTAT has a slight non-linear variation with the target variable:

[18]:
```
from sklearn.datasets import load_boston
boston = load_boston()

#create a dataframe containing predictors (housing_X) and the response variable␣
 ↪(housing_y)
housing_X = pd.DataFrame(boston.data, columns = boston.feature_names)[['LSTAT']]
housing_y = pd.Series(boston.target, name = "price")

#also create a combined data frame with both predictors and response variables
housing_df = pd.concat([housing_y, housing_X], axis=1)

# show the LSTAT versus price
alt.Chart(housing_df).mark_point().encode(
    x="LSTAT",
    y="price"
)
```

[18]:
```
alt.Chart(...)
```

5.1 3.1 HOPs and spaghetti plots

Use HOPs and spaghetti plots to visualize a regression model predicting `price` using `LSTAT`. You can use any model type you like, including linear regression, polynomial regression, or any other regression model type.

5.1.1 Question 3.1.1 Define helper functions (10 points)

Define the helper functions you will need, including:

1. A function to construct an Altair chart of the data: `get_housing_points_chart()`
2. A function to get one bootstrap sample of the fit: `get_one_bootstrap_housing_fit()`
3. A function to construct an Altair chart of one regression fit curve: `get_housing_fit_chart()`

[19]:
```
# define your helper functions below. Hint: this is also a good place to define␣
 ↪a prediction grid

#prediction grid
poly_pred_grid2 = pd.DataFrame({
    "x": np.linspace(housing_df['LSTAT'].min(), housing_df['price'].max(),␣
 ↪num=101)
```

```
})

def get_housing_points_chart():
    '''
    This function should return an altair plot object that is a scatterplot of
    the salary data, with YearsExperience on the x axis and Salary on the y axis
    '''
    return alt.Chart(housing_df).mark_point().encode(
        x="LSTAT",
        y="price"
    )


def get_one_bootstrap_housing_fit():
    '''Get one bootstrap sampled polynomial regression fit to the data'''
    #resample the data with replacement (replace=True) to a data frame with
    #the same number of data points (frac=1.0)
    resampled_df = housing_df.sample(frac=1.0, replace=True)

    #fit model to resampled data
    X = resampled_df[['LSTAT']] #[[ ]] subsets so X remains a DataFrame
    y = resampled_df['price']   #y should be an array, so we use [ ]

    #x must be transformed into polynomials (e.g. x, x^2, x^3 ... up to the value of `degree`)
    polynomial_features = PolynomialFeatures(degree=2)
    X_poly = polynomial_features.fit_transform(X)
    poly_reg2 = linear_model.LinearRegression()
    poly_reg2.fit(X_poly, y)

    return poly_reg2

def get_housing_fit_chart(poly_reg2, opacity=0.5):
    '''
    Takes a single polynomial regression fit (as returned by `get_one_bootstrap_poly_fit()`) and
    returns an Altair chart plotting the fit curve

    Parameters:

    - poly_reg: A regression fit
    - opacity: The opacity of the output line
    '''
    #use the model to predict y at each x position
    polynomial_features = PolynomialFeatures(degree=2)
    pred_df = pd.DataFrame({
```

```
        'x': poly_pred_grid2['x'],
        'y': poly_reg2.predict(polynomial_features.
↪fit_transform(poly_pred_grid2))
    })

    #return an Altair chart showing the fit line
    return alt.Chart(pred_df).mark_line(
        opacity=opacity,
        color='red'
    ).encode(
        x='x',
        y='y'
    )
```

5.1.2 Question 3.1.2 Create a spaghetti plot for your model (5 points)

Using the helper functions you created above, visualize a spaghetti plot of your model below.

[20]:
```
B = 50

# get `B` bootstrapped fit line charts
# Note opacity=0.1 sets the line opacity so it is easier to see the overlapping␣
↪lines. Make
# sure your get_salary_linear_fit_chart() function (defined above) properly␣
↪uses the opacity argument!
poly_charts2 = [get_housing_fit_chart(get_one_bootstrap_housing_fit(),␣
↪opacity=0.1) for _ in range(B)]

#combine all the line charts together and layer on the points chart
alt.layer(*poly_charts2) + get_housing_points_chart()
```

[20]:
```
alt.LayerChart(...)
```

5.1.3 Question 3.1.3 Create a HOPs chart for your model (5 points)

Using the helper functions you created above, visualize a HOPs chart of your model below.

[21]:
```
def get_one_frame2(i):
    '''
    Return one frame in the animation
    '''

    time.sleep(.2)

    # get the point chart
    points_chart = get_housing_points_chart()

    # fit one bootstrap regression
```

```
    housing_reg = get_one_bootstrap_housing_fit()

    # get the line chart
    housing_chart = get_housing_fit_chart(housing_reg)

    #return the combined points + lines chart
    return housing_chart + points_chart

interact(get_one_frame2, i = widgets.Play(
    value=0,
    min=0,
    max=100,
    step=1,
    description="Press play",
    disabled=False))
```

```
interactive(children=(Play(value=0, description='Press play'), Output()), _dom_classes=('widget
```

[21]:
```
<function __main__.get_one_frame2(i)>
```

5.1.4 Question 3.1.4 Reflect on your model (3 points)

Given the visualizations above, reflect on the model you chose and your uncertainty in the relationship between `LSTAT` and `price` in these data. Discuss both small world and large world uncertainty. This discussion should be one to two paragraphs at most.

Regarding small world uncertainty, spaghetti plots and HOPs allow you to mitigate deterministic construal errors surrounding uncertainty with the line of best fit. They are effective in that they show which lines of best fit are more likely to express the relationship between LSTAT and price by the density of the lines. Regarding large world uncertainty, each regression line is based off of bootstrapped samples, so if the training data is not representative, the vast majority of regression lines may deviate from the true line.

Please remember to submit both the HTML and .ipynb formats of your completed notebook. When generating your HTML, be sure to run your complete code first before downloading as HTML. Please remember to work on your explanations and interpretations!