



# CS109A Introduction to Data Science:

## Homework 3: Polynomial and LASSO Regression

Harvard University

Fall 2021

Instructors: Pavlos Protopapas and Natesh Pillai

---

In [1]:

```
1 # RUN THIS CELL
2 import requests
3 from IPython.core.display import HTML
4 styles = requests.get(
5     "https://raw.githubusercontent.com/Harvard-IACS/2021-CS109A/master/"
6     "themes/static/css/cs109.css"
7 ).text
8 HTML(styles)
```

Out[1]:

## Import Libraries

In [2]:

```
1 from collections import Counter
2 import math
3
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import pandas as pd
7 import seaborn as sns
8 from sklearn.linear_model import Lasso
9 from sklearn.linear_model import LinearRegression
10 from sklearn.metrics import mean_squared_error
11 from sklearn.model_selection import cross_validate
12 from sklearn.model_selection import train_test_split
13 from sklearn.preprocessing import PolynomialFeatures
14 from sklearn.utils import shuffle
15
16 %matplotlib inline
```

---

## INSTRUCTIONS

- **THIS IS AN INDIVIDUAL ASSIGNMENT. Collaboration on this homework is NOT PERMITTED.**

- **THIS IS A TWO-WEEK HOMEWORK.** The material is indeed two-weeks-worth of work, so make sure you start it early!!!
- To submit your assignment follow the instructions given in Canvas.
- Please **restart the kernel and run the entire notebook again before you submit.**
- Running cells out of order is a common pitfall in Jupyter Notebooks. To make sure your code continues to work, restart the kernel and rerun your notebook periodically while working through this assignment.
- We have tried to include all the libraries you may need to do the assignment in the imports cell provided below. **Please use only the libraries provided in those imports.**
- Please use `.head(...)` when viewing data. Do not submit a notebook that is **excessively long**.
- In questions that require code to answer, such as "calculate and report  $R^2$ ", do not just output the value from a cell. Write a `print(...)` function that clearly labels the output, includes a reference to the calculated value, and rounds it to a reasonable number of digits. **Do not hard code values in your printed output.** For example, this is an appropriate print statement:

```
print(f'The  $R^2$  is {R:.4f}')
```

- **Your plots MUST be clearly labeled and easy to read**, including clear labels for the  $x$  and  $y$  axes, a descriptive title ("MSE plot" is NOT a descriptive title; "95% confidence interval of coefficients for degree-5 polynomial model" on the other hand is descriptive), a legend when appropriate, and clearly formatted text and graphics.
  - **Your code may also be evaluated for efficiency and clarity.** As a result, correct output is not always sufficient for full credit.
- 

## Notebook contents

- [Overview and data description](#)
- [Question 1: Data visualization \[5 pts\]](#)
  - [Solutions](#)
- [Your Homework 3 roadmap](#)
- [Question 2: Guesstimate the polynomial relationship \[14 pts\]](#)
  - [Solutions](#)
- [Question 3: Use a validation set to find the most promising polynomial relationship \[12 pts\]](#)
  - [Solutions](#)
- [Question 4: Finding the best model by k-fold cross validation \[14 pts\]](#)
  - [Solutions](#)
- [Question 5: Finding the most consistent model using k-fold cross validation with bootstraps \[16 pts\]](#)
  - [Solutions](#)
- [Question 6: Improving model consistency with LASSO regularization \[33 pts\]](#)
  - [Solutions](#)
- [Question 7: Analyze your best test MSEs for each section of the homework \[6 pts\]](#)
  - [Solutions](#)

# Overview and data description

[Return to contents](#)

## Predicting percentage change in bacteria populations given their spreading factor

In this homework, we will explore polynomial regression for predicting a quantitative variable. Specifically, we will build regression models that can predict the percentage change in bacteria population after 4 hours based on their "spreading factor". These prediction models can be useful in clustering of a novel bacteria to any class.

If the percentage of change in population is positive, this indicates that the size of the bacteria population has grown. One important factor to note is that the percentage change could be negative. This indicates that within the specified time frame, the population of the bacteria has decreased from its original size.

The data set for this problem has already been split for you. You will find the train and test data sets in the files `data/bacteria_train.csv` and `data/bacteria_test.csv`, respectively. The first column in each file gives information about the change in percentage of the population and the second column contains the spreading factor of bacteria populations.

## Problem Description

We will predict the percentage change in population, given the spreading factor. For this exercise, we will consider the `Spreading_factor` to be our predictor variable and `Perc_population` to be our response variable. We will explore several different approaches, with mean squared error ( $MSE$ ) as an evaluative criteria, for finding the best fit and most robust polynomial regression for modeling this relationship.

## Question 1: Data visualization [5 pts]

[Return to contents](#)

**1.1** Generate a scatter plot of the data points in the `bacteria_train.csv` file with the spreading factor on the  $x$ -axis and the percentage change on the  $y$ -axis.

**REMEMBER:** In this course, you will be expected to ALWAYS label your axes, title your graphs, and produce visuals that clearly communicate the data (as described in the [INSTRUCTIONS](#) at the start of this notebook). Visuals should typically be accompanied by text identifying the key point of the visual and defending any choices you make as a data scientist regarding the visual to best communicate your data.

**1.2** Based on the graph, is there any discernable relationship between the spreading factor and percentage change in population?

## Question 1: Solutions

[Return to contents](#)

**1.1** Generate a scatter plot of the data points in the `bacteria_train.csv` file with the spreading factor on the  $x$ -axis and the percentage change on the  $y$ -axis.

**REMEMBER:** In this course, you will be expected to ALWAYS label your axes, title your graphs, and produce visuals that clearly communicate the data (as described in the [INSTRUCTIONS](#) at the start of this notebook). Visuals should typically be accompanied by text identifying the key point of the visual and defending any choices you make as a data scientist regarding the visual to best communicate your data.

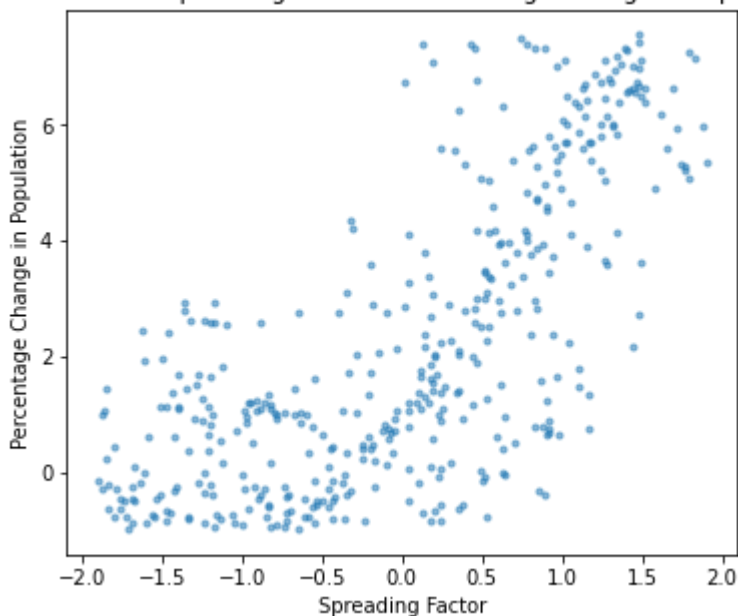
In [3]:

```
1 # your code here
2 bac_tr = pd.read_csv("data/bacteria_train.csv")
3
4 bac_tr.plot.scatter(x='Spreading_factor', y='Perc_population', s=10, alpha=.5, f
5 plt.title('Scatter Plot of Spreading Factor vs. Percentage Change in Population')
6 plt.xlabel('Spreading Factor')
7 plt.ylabel('Percentage Change in Population')
```

Out[3]:

Text(0, 0.5, 'Percentage Change in Population')

Scatter Plot of Spreading Factor vs. Percentage Change in Population



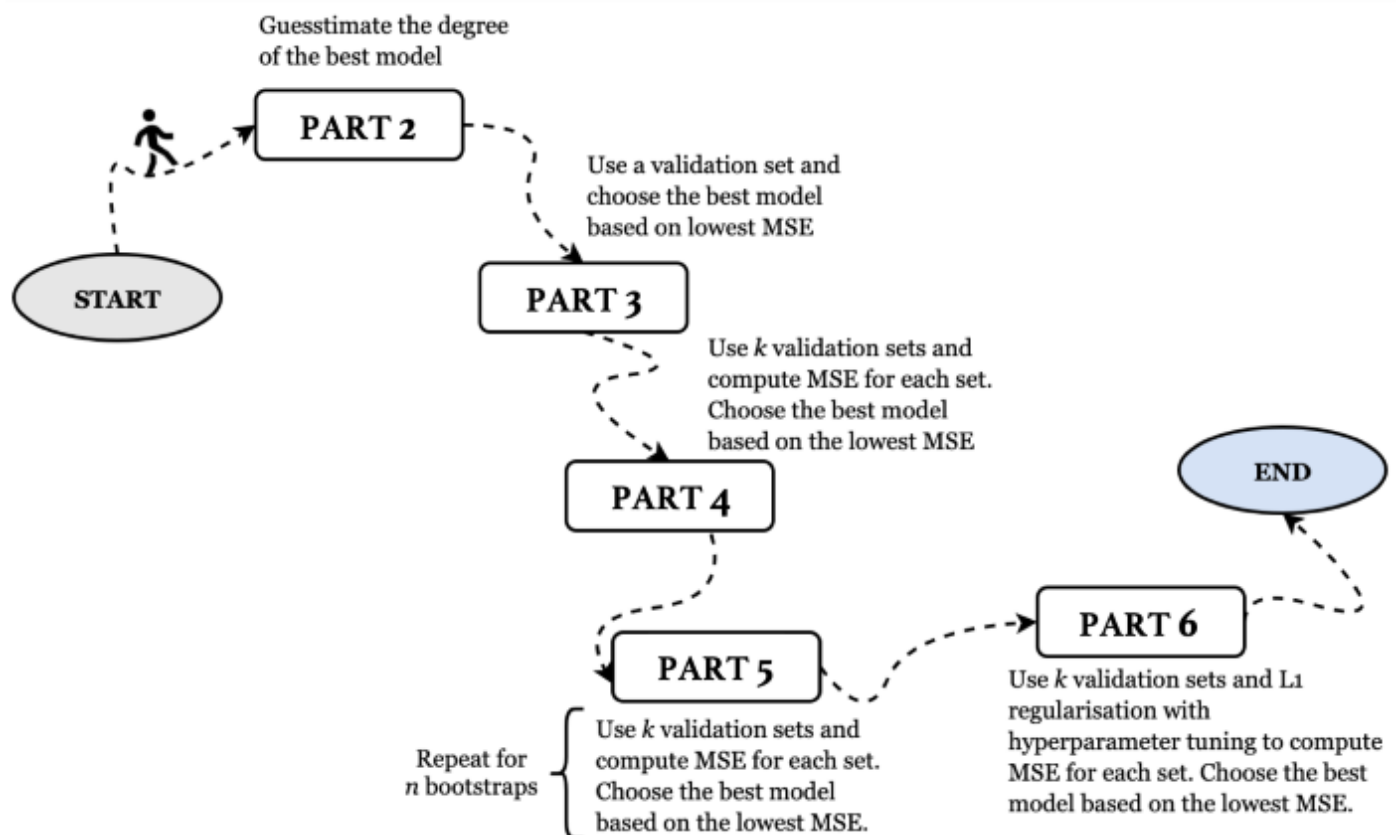
**1.2** Based on the graph, is there any discernable relationship between the spreading factor and percentage change in population?

**Yes, the relationship between spreading factor and percentage change in population is generally positive.**

## Your Homework 3 roadmap

[Return to contents](#)

We will be following this roadmap for Question 2 through Question 6:



## Question 2: Guesstimate the polynomial relationship [14 pts]

[Return to contents](#)

**2.1** Guesstimate the degree of the polynomial regression that may best fit the data given our observation of the data in Question 1, and then fit that model following the requirements outlined below:

- Create a single dictionary called `best_mse_dict` to store the best test  $MSE$ s for each type of model we will be building in this notebook. The keys of this dictionary will be the question number, and the values will be the  $MSE$  values. So, for this question, you will have `best_mse_dict["2.1"] = ...`.
- Generate `PolynomialFeatures(...)` for your guesstimated degree and fit a polynomial regression using scikit-learn's `LinearRegression(...)` on the training set.
- Generate a plot with both the train and test data, as well as your prediction curve.
- Compute and print the  $MSE$  on the test set and save that value to `best_mse_dict`.
- Explain the rationale for your choice of degree (i.e. your "guesstimation") and comment on the fit of your model.

**2.2** Now we will compute confidence intervals for the coefficients generated in our model. For the same degree polynomial regression fit above in 2.1, compute and plot the 95% confidence interval of the beta (i.e.  $\beta$ ) values obtained in that regression model. Be certain to meet these primary requirements:

- Report the coefficient values of the model fit above in 2.1.
- Use bootstrapping to generate your 95% confidence intervals by refitting your model to each bootstrapped sample of your training data (you can limit your number of bootstraps to 100).

- For each bootstrap, store the coefficients of the model, and use those stored coefficients to compute the 2.5 and 97.5 confidence bounds.
- Plot the bootstrapped coefficients as a grid of histograms, one histogram for each  $\beta_i$ , illustrating the distribution of bootstrapped coefficients for each  $\beta_i$ . Each distribution should indicate the regions of 95% confidence. Use `plt.subplots(...)` to arrange your histograms as a grid of plots.
- Comment on your findings.

**REMEMBER:** Bootstrapped samples should always be the same size as the original data and sampled with replacement.

## Question 2: Solutions

[Return to contents](#)

**2.1** Guesstimate the degree of the polynomial regression that may best fit the data given our observation of the data in Question 1, and then fit that model following the requirements outlined below:

- Create a single dictionary called `best_mse_dict` to store the best test *MSE*s for each type of model we will be building in this notebook. The keys of this dictionary will be the question number, and the values will be the *MSE* values. So, for this question, you will have `best_mse_dict["2.1"] = ...`.
- Generate `PolynomialFeatures(...)` for your guesstimated degree and fit a polynomial regression using scikit-learn's `LinearRegression(...)` on the training set.
- Generate a plot with both the train and test data, as well as your prediction curve.
- Compute and print the *MSE* on the test set and save that value to `best_mse_dict`.
- Explain the rationale for your choice of degree (i.e. your "guesstimation") and comment on the fit of your model.

In [4]:

```
1  # your code here
2  bac_ts = pd.read_csv("data/bacteria_test.csv")
3  X_test = bac_ts[['Spreading_factor']]
4  y_test = bac_ts.Perc_population
5
6  best_mse_dict = {}
7  # mse_21 = []
8
9  X = bac_tr[['Spreading_factor']]
10 y = bac_tr[["Perc_population"]]
11
12 # shuffle the data
13 indices= np.random.choice(X.index, replace = False, size = len(X.index))
14 X = X.iloc[indices,:]
15 y = y.iloc[indices,:].Perc_population
16
17 # select degree
18 guess_deg = 3
19 X_poly_train = PolynomialFeatures(degree = guess_deg).fit_transform(X)
20 X_poly_test = PolynomialFeatures(degree = guess_deg).fit_transform(X_test)
21 lreg = LinearRegression(fit_intercept=False)
22 lreg.fit(X_poly_train, y)
23 y_train_pred = lreg.predict(X_poly_train)
24 y_test_pred = lreg.predict(X_poly_test)
25 best_mse_dict['2.1'] = mean_squared_error(y_test, y_test_pred)
26 print(f'The MSE on the test set is {mean_squared_error(y_test, y_test_pred):.4f}')
27
```

The MSE on the test set is 2.6519.

In [5]:

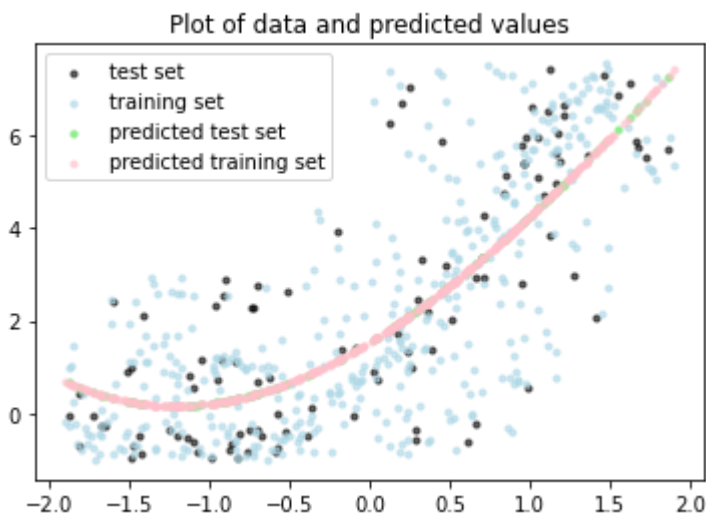
```

1 fig, ax = plt.subplots()
2 plt.scatter(X_test,y_test,color='k', label = 'test set', s=10, alpha=0.6)
3 plt.scatter(X,y,color='lightblue', label = 'training set', s=10,alpha=0.6)
4 plt.scatter(X_test,y_test_pred,color='lightgreen', label = 'predicted test set',
5 plt.scatter(X,y_train_pred,color='pink', label = 'predicted training set', s=10,
6 plt.title("Plot of data and predicted values")
7 ax.legend()

```

Out[5]:

&lt;matplotlib.legend.Legend at 0x7fc5890716d0&gt;

**INTERPRETATION:**

My guesstimated degree is 3, because cubic polynomial is commonly used. With degree of 3, the model has MLE of 2.6519 on the test set. From the plot of data and predicted values shown above, the model fits pretty well.

**2.2** Now we will compute confidence intervals for the coefficients generated in our model. For the same degree polynomial regression fit above in 2.1, compute and plot the 95% confidence interval of the beta (i.e.  $\beta$ ) values obtained in that regression model. Be certain to meet these primary requirements:

- Report the coefficient values of the model fit above in 2.1.
- Use bootstrapping to generate your 95% confidence intervals by refitting your model to each bootstrapped sample of your training data (you can limit your number of bootstraps to 100).
- For each bootstrap, store the coefficients of the model, and use those stored coefficients to compute the 2.5 and 97.5 confidence bounds.
- Plot the bootstrapped coefficients as a grid of histograms, one histogram for each  $\beta_i$ , illustrating the distribution of bootstrapped coefficients for each  $\beta_i$ . Each distribution should indicate the regions of 95% confidence. Use `plt.subplots(...)` to arrange your histograms as a grid of plots.
- Comment on your findings.

**REMEMBER:** Bootstrapped samples should always be the same size as the original data and sampled with replacement.



In [6]:

```

1  # your code here
2
3  print(f"The coefficient values of the model fit above are {[round(i,4) for i in
4
5  # bootstrapping
6  boot_linreg_models = []
7  boot_betas = []
8  numboot = 200
9  # degree=3
10
11  X = bac_tr[['Spreading_factor']]
12  y = bac_tr[["Perc_population"]]
13
14  X_tr_df = X.copy()
15  y_train = y.copy()
16
17  # bootstrap
18  for i in range(numboot):
19
20      # randomly sample indices
21      bootstrap_i_indices= np.random.choice(X_tr_df.index, replace = True, size = 1)
22
23      # get bootstrapped dfs
24      Xtr_boot = X_tr_df.iloc[bootstrap_i_indices,:]
25      ytr_boot = y_train.iloc[bootstrap_i_indices,:].Perc_population
26
27      Xtr_boot_poly = PolynomialFeatures(degree=guess_deg).fit_transform(Xtr_boot)
28      bootstrap_lr = LinearRegression(fit_intercept = False).fit(Xtr_boot_poly, yt
29
30      boot_linreg_models.append(bootstrap_lr)
31      coefs = bootstrap_lr.coef_
32      boot_betas.append(coefs)
33
34  # linreg_models = np.array(linreg_models)
35  boot_betas = pd.DataFrame(boot_betas, columns = ["beta0", "beta1", "beta2", "bet
36  boot_coefs = boot_betas.iloc[:,1:]
37  boot_coefs.head()
38
39  # compute the CIs
40  fig, ax = plt.subplots(1,3, figsize = (15,5))
41  ax = ax.ravel()
42  hist_colors = ["lightblue", "teal", "orange", "purple"]
43  for i in range(3):
44      betavals = boot_coefs.iloc[ : , i]
45      betavals.values.sort()
46
47      x1 = np.percentile(betavals, 2.5)
48      x2 = np.percentile(betavals, 97.5)
49      print(f"The CI of beta{i+1} is ({x1:.4f},{x2:.4f})")
50      x = np.linspace( x1, x2, 500)
51      counts, bins = np.histogram( betavals )
52      y = counts.max()
53      plt.sca( ax[i] )
54      plt.fill_between(x, y+100, color = 'pink',alpha=0.3)
55      plt.hist(betavals,
56              bins = bins,
57              color=hist_colors[i],
58              alpha=1,
59              edgecolor='black',

```

```

60         linewidth=1)
61     plt.ylim(0, y + 4)
62     plt.ylabel(f'Distribution of beta {i+1}', fontsize=18)
63     plt.xlabel(f'Value of beta {i+1}', fontsize=18)
64     plt.axvline(x = np.mean(betavals), color='w', linewidth = 3, linestyle = "--")

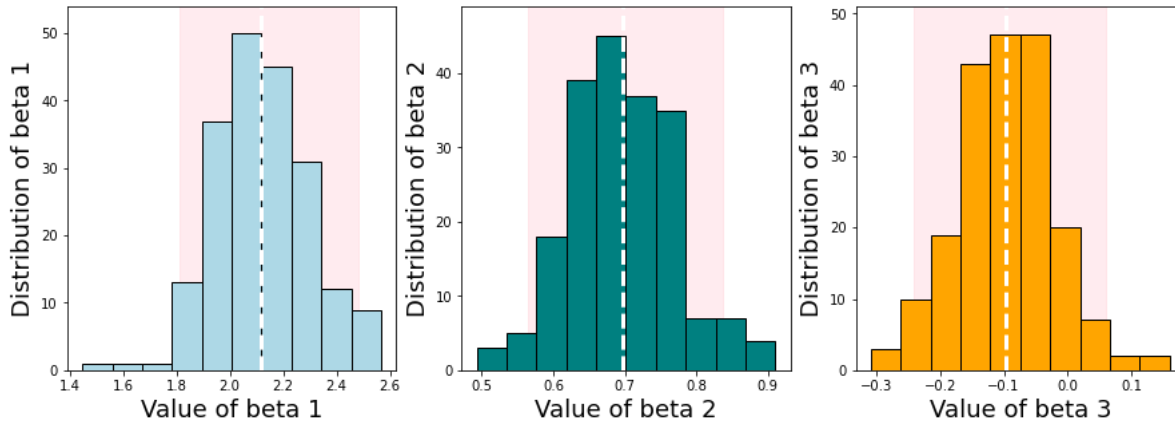
```

The coefficient values of the model fit above are [2.1234, 0.6942, -0.0986].

The CI of beta1 is (1.8115, 2.4796)

The CI of beta2 is (0.5651, 0.8379)

The CI of beta3 is (-0.2405, 0.0593)



## INTERPRETATION:

The plots show the distribution of bootstrapped coefficients for each beta. The distribution is roughly normal with the mean around the coefficient values calculated previously.

## Question 3: Use a validation set to find the most promising polynomial relationship [12 pts]

[Return to contents](#)

**3.1** Find the best degree polynomial relationship using a single validation set. Be certain to meet the requirements outlined below:

- Split your training data such that you separate out a single 20% validation split.
- Fit polynomial regression models up to **degree 30** on the 80% training set (one model for each degree polynomial regression).
- Generate a single plot illustrating the train and validation  $MSE$  values for each fitted degree polynomial regression model.
- Compare the validation  $MSE$  values and select and report the degree for which the validation error is lowest.

**3.2** Now, with the best degree selected, train the polynomial regression on the **complete training set** (including the observations that you had previously removed to make the validation set). Report the train and test  $MSE$  and add the test  $MSE$  to `best_mse_dict`.

**3.3** Generate a plot of the data and your regression curve (similar to [Question 2.1](#)). Comment on how your model fits the data and compare it to the fit of your "guesstimated" model from [Question 2](#).

## Question 3: Solutions

[Return to contents](#)

**3.1** Find the best degree polynomial relationship using a single validation set. Be certain to meet the requirements outlined below:

- Split your training data such that you separate out a single 20% validation split.
- Fit polynomial regression models up to **degree 30** on the 80% training set (one model for each degree polynomial regression).
- Generate a single plot illustrating the train and validation  $MSE$  values for each fitted degree polynomial regression model.
- Compare the validation  $MSE$  values and select and report the degree for which the validation error is lowest.

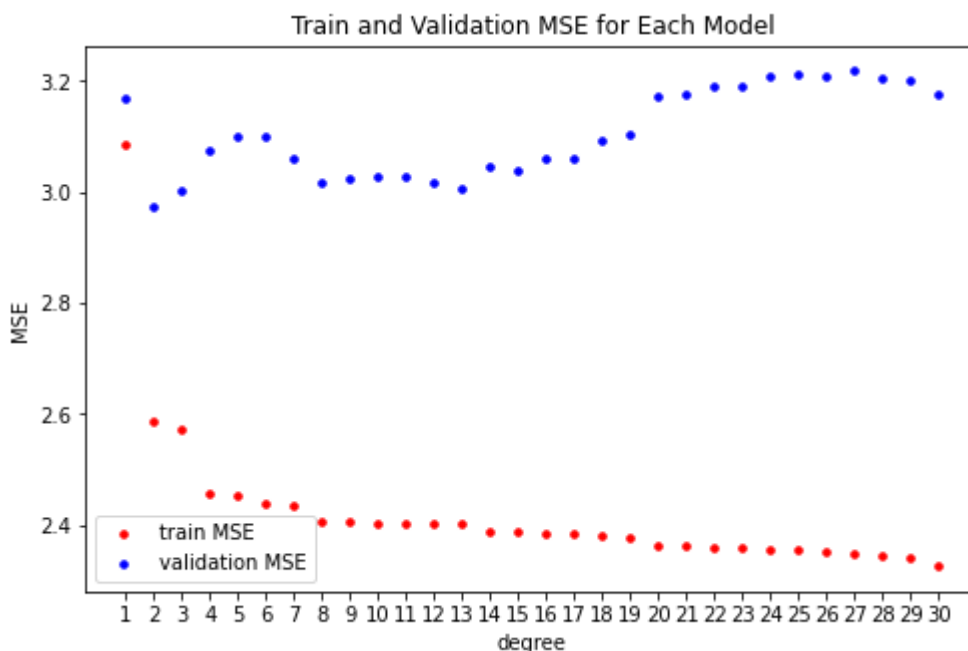
In [7]:

```

1  # your code here
2  X = bac_tr[['Spreading_factor']]
3  y = bac_tr.Perc_population
4  X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=.2, random_st
5  mse_train = []
6  mse_val = []
7  max_deg = 30
8  for deg in range(1,max_deg+1):
9      X_poly_train = PolynomialFeatures(degree = deg).fit_transform(X_train)
10     X_poly_val = PolynomialFeatures(degree = deg).fit_transform(X_val)
11     lreg = LinearRegression(fit_intercept=False)
12     lreg.fit(X_poly_train, y_train)
13     y_train_pred = lreg.predict(X_poly_train)
14     y_val_pred = lreg.predict(X_poly_val)
15     mse_train.append(mean_squared_error(y_train, y_train_pred))
16     mse_val.append(mean_squared_error(y_val, y_val_pred))
17
18 fig, ax = plt.subplots(figsize = (8,5))
19 plt.scatter([i for i in range(1,max_deg+1)],mse_train, color='red', label = 'tra
20 plt.scatter([i for i in range(1,max_deg+1)],mse_val,color='blue', label = 'valid
21 plt.xlabel("degree")
22 plt.ylabel("MSE")
23 plt.xticks([i for i in range(1,max_deg+1)])
24 plt.title("Train and Validation MSE for Each Model")
25 ax.legend()
26
27 # best model
28 degseq = [i for i in range(1,max_deg+1)]
29 bestdegree = degseq[mse_val.index(min(mse_val))]
30 print(f'The best model with lowest validation MSE has {bestdegree} degrees.')

```

The best model with lowest validation MSE has 2 degrees.



**3.2** Now, with the best degree selected, train the polynomial regression on the **complete training set** (including the observations that you had previously removed to make the validation set). Report the train and test *MSE* and add the test *MSE* to `best_mse_dict`.

In [8]:

```

1  # your code here
2  bac_ts = pd.read_csv("data/bacteria_test.csv")
3  X_test = bac_ts[['Spreading_factor']]
4  y_test = bac_ts.Perc_population
5
6  # deg = 2
7  X_train_poly = PolynomialFeatures(degree = bestdegree).fit_transform(X)
8  X_test_poly = PolynomialFeatures(degree = bestdegree).fit_transform(X_test)
9  lreg = LinearRegression(fit_intercept=False)
10 lreg.fit(X_train_poly, y)
11 y_train_pred = lreg.predict(X_train_poly)
12 y_test_pred = lreg.predict(X_test_poly)
13
14 best_mse_dict['3.2'] = mean_squared_error(y_test, y_test_pred)
15 print(f"With polynomial degree of {bestdegree}, the train MSE is {mean_squared_e

```

With polynomial degree of 2, the train MSE is 2.6617, and the test MSE is 2.6749.

**3.3** Generate a plot of the data and your regression curve (similar to [Question 2.1](#)). Comment on how your model fits the data and compare it to the fit of your "guesstimated" model from [Question 2](#).

In [9]:

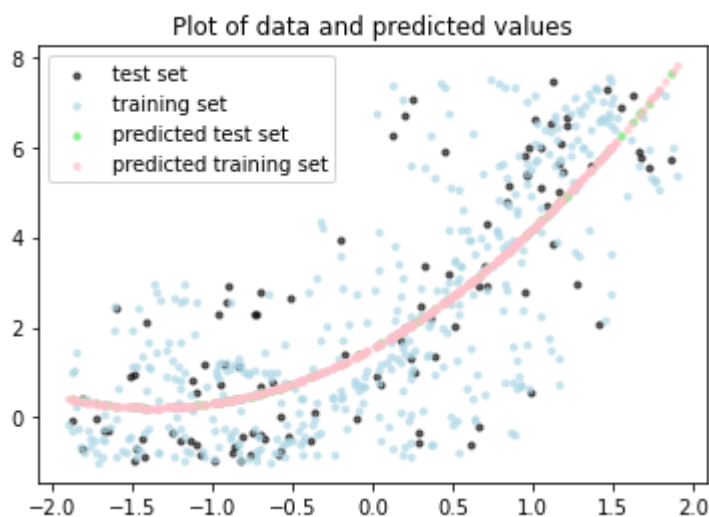
```

1  # your code here
2
3  fig, ax = plt.subplots()
4  plt.scatter(X_test,y_test,color='k', label = 'test set', s=10, alpha=0.6)
5  plt.scatter(X,y,color='lightblue', label = 'training set', s=10,alpha=0.6)
6  plt.scatter(X_test,y_test_pred,color='lightgreen', label = 'predicted test set',
7  plt.scatter(X,y_train_pred,color='pink', label = 'predicted training set', s=10,
8  plt.title("Plot of data and predicted values")
9  ax.legend()

```

Out[9]:

<matplotlib.legend.Legend at 0x7fc548e90880>



**INTERPRETATION:**

The test  $MSE$  is a little bit larger than the test  $MSE$  of my previous "guesstimated" model from Q2. From the plot, in general, this model fits quite well.

## Question 4: Finding the best model by k-fold cross validation [14 pts]

[Return to contents](#)

**4.1** In this part we find the best degree polynomial regression by running the model on a range of degree values and using k-fold cross validation on the `bacteria_train.csv` dataset.

- Use scikit-learn's `cross_validate(...)` to perform cross validation with  $k=10$  for each degree polynomial regression up to **degree 30**.
- Generate a single plot showing the mean train and validation  $MSE$  values (see note below) for each degree polynomial regression, including the  $\pm 1$  standard deviation bounds for the validation  $MSE$  values. Comment on trends and findings illustrated by your plot.
- Print the degree of the best model, identified based on the lowest mean validation  $MSE$ , along with the corresponding mean train and validation  $MSE$  values.

**NOTE:** When we refer to train and validation  $MSE$  values here in Question 4.1 and in future problems where we are performing cross validation, we are referring specifically to the cross-validation generated train and validation  $MSE$  values. Just please remember, when accessing your validation  $MSE$  values while performing cross validation with scikit-learn's `cross_validate(...)`, scikit-learn rather confusingly refers to its validation scores as `test_score` in the dictionary that it returns, even though those values are actually validation scores. Please see the `cross_validate(...)` [documentation \(https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.cross\\_validate.html\)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html) for more info on this function.

**4.2** Fit the best model on the entire training data and report the  $MSE$ .

- For the best degree polynomial regression identified in Question 4.1 above, train on the complete training set from the `bacteria_train.csv` file and predict on the test data.
- Report the coefficients of the model.
- Report both the training and test  $MSE$  values, and save the **test  $MSE$**  to `best_mse_dict`.

**4.3** Generate a plot of the data and your regression curve (similar to [Question 2.1](#)). Comment on how your model fits the data and how your model compares relative to the prior best-fit model you generated using just a single validation set in [Question 3](#).

## Question 4: Solutions

[Return to contents](#)

**4.1** In this part we find the best degree polynomial regression by running the model on a range of degree values and using k-fold cross validation on the `bacteria_train.csv` dataset.

- Use scikit-learn's `cross_validate(...)` to perform cross validation with  $k=10$  for each degree polynomial regression up to **degree 30**.
- Generate a single plot showing the mean train and validation  $MSE$  values (see note below) for each degree polynomial regression, including the  $\pm 1$  standard deviation bounds for the validation  $MSE$  values. Comment on trends and findings illustrated by your plot.
- Print the degree of the best model, identified based on the lowest mean validation  $MSE$ , along with the corresponding mean train and validation  $MSE$  values.

**NOTE:** When we refer to train and validation  $MSE$  values here in Question 4.1 and in future problems where we are performing cross validation, we are referring specifically to the cross-validation generated train and validation  $MSE$  values. Just please remember, when accessing your validation  $MSE$  values while performing cross validation with scikit-learn's `cross_validate(...)`, scikit-learn rather confusingly refers to its validation scores as `test_score` in the dictionary that it returns, even though those values are actually validation scores. Please see the `cross_validate(...)` [documentation \(https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.cross\\_validate.html\)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html) for more info on this function.

In [10]:

```

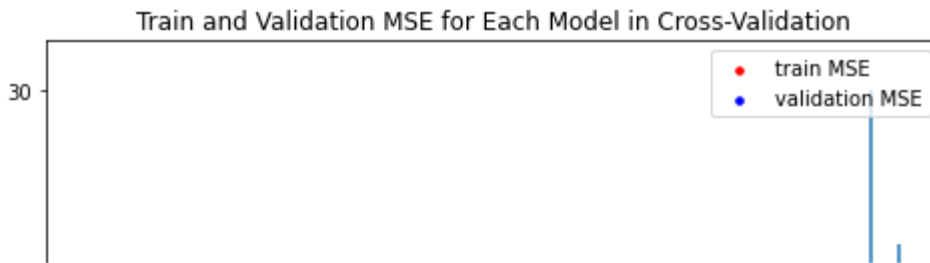
1  # your code here
2
3  X = bac_tr[['Spreading_factor']]
4  y = bac_tr[["Perc_population"]]
5
6  # shuffle the data
7  indices= np.random.choice(X.index, replace = False, size = len(X.index))
8  X = X.iloc[indices,:]
9  y = y.iloc[indices,:].Perc_population
10
11 max_deg = 30
12 cross_validation_error = []
13 cross_train_error = []
14 std_dict = {}
15
16 # cross validation
17 for d in range(1,max_deg+1):
18
19     X_poly = PolynomialFeatures(degree=d).fit_transform(X)
20
21     lreg=LinearRegression(fit_intercept=False)
22     lreg.fit(X_poly,y)
23     validation_results = cross_validate(lreg, X_poly, y,
24                                         scoring = "neg_mean_squared_error",
25                                         cv=10,
26                                         return_train_score = True)
27     mse_validation = validation_results['test_score'].mean()
28     mse_train = validation_results['train_score'].mean()
29     val_scores = validation_results['test_score']
30     std_dict[d] = -val_scores
31     cross_validation_error.append(-mse_validation)
32     cross_train_error.append(-mse_train)
33
34 df = pd.DataFrame(std_dict).melt()
35 df2 = df.groupby("variable").agg('std').reset_index()
36 std_arr = np.array(df2['value'])
37
38 # plot MSE
39 fig, ax = plt.subplots(figsize = (8,10))
40 plt.scatter([i for i in range(1,max_deg+1)],cross_train_error, color='red', label='Train')
41 plt.scatter([i for i in range(1,max_deg+1)],cross_validation_error,color='blue', label='Validation')
42 plt.errorbar([i for i in range(1,max_deg+1)],cross_validation_error, yerr=std_arr,color='blue')
43 plt.xlabel("degree")
44 plt.ylabel("MSE")
45 plt.xticks([i for i in range(1,max_deg+1)])
46 plt.title("Train and Validation MSE for Each Model in Cross-Validation")
47 ax.legend()

```

Out[10]:

&lt;matplotlib.legend.Legend at 0x7fc54a7dfe50&gt;





In [11]:

```
1 # best model
2 degseq = [i for i in range(1,max_deg+1)]
3 bestdegree = degseq[cross_validation_error.index(min(cross_validation_error))]
4 print(f'The best model has {bestdegree} degrees.')
```

The best model has 8 degrees.

## INTERPRETATION:

The plot above shows that as the degree increases, the train MSE increases while the validation MSE decreases and then increases. The standard deviation of the validation MSE starts to increase when the degree is larger than 23. The best model with the lowest validation MSE has 4 degrees.

### 4.2 Fit the best model on the entire training data and report the *MSE*.

- For the best degree polynomial regression identified in Question 4.1 above, train on the complete training set from the `bacteria_train.csv` file and predict on the test data.
- Report the coefficients of the model.
- Report both the training and test *MSE* values, and save the **test *MSE*** to `best_mse_dict`.

In [12]:

```
1 # your code here
2 X_train_poly = PolynomialFeatures(degree = bestdegree).fit_transform(X)
3 X_test_poly = PolynomialFeatures(degree = bestdegree).fit_transform(X_test)
4 lreg = LinearRegression(fit_intercept=False)
5 lreg.fit(X_train_poly, y)
6 y_train_pred = lreg.predict(X_train_poly)
7 y_test_pred = lreg.predict(X_test_poly)
8 print(f"The coefficients are {[round(i,4) for i in lreg.coef_[1:]]}.")
9
10 best_mse_dict['4.2'] = mean_squared_error(y_test, y_test_pred)
11 print(f"With polynomial degree of {bestdegree}, the train MSE is {mean_squared_e
```

The coefficients are [2.597, -1.0949, -1.1891, 3.4831, 0.6529, -1.713, -0.1153, 0.2404].

With polynomial degree of 8, the train MSE is 2.5125, and the test MSE is 2.6130.

### 4.3 Generate a plot of the data and your regression curve (similar to [Question 2.1](#)). Comment on how your model fits the data and how your model compares relative to the prior best-fit model you generated using just a single validation set in [Question 3](#).

In [13]:

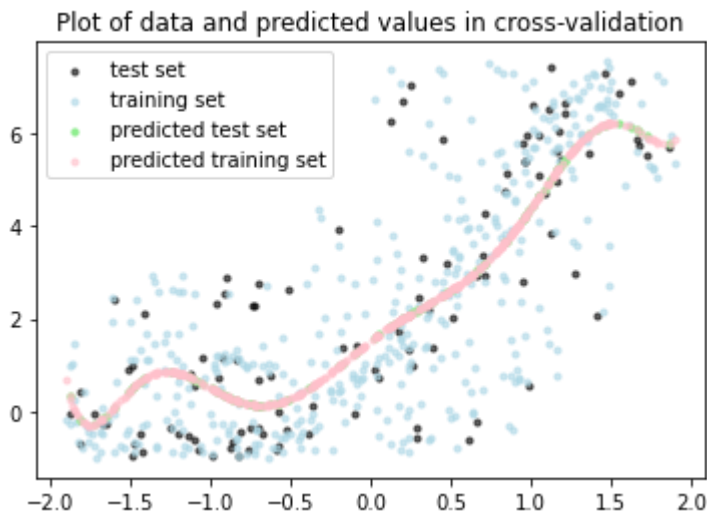
```

1 # your code here
2 fig, ax = plt.subplots()
3 plt.scatter(X_test,y_test,color='k', label = 'test set', s=10, alpha=0.6)
4 plt.scatter(X,y,color='lightblue', label = 'training set', s=10,alpha=0.6)
5 plt.scatter(X_test,y_test_pred,color='lightgreen', label = 'predicted test set',
6 plt.scatter(X,y_train_pred,color='pink', label = 'predicted training set', s=10,
7 plt.title("Plot of data and predicted values in cross-validation")
8 ax.legend()

```

Out[13]:

&lt;matplotlib.legend.Legend at 0x7fc5381fd700&gt;

**INTERPRETATION:**

From the plot, it seems that the model fits the data pretty well. The validation MSE of this model with degree of 4 is about 0.07 lower than the MSE of the previous model with degree of 2.

## Question 5: Finding the most consistent model using k-fold cross validation with bootstraps [16 pts]

[Return to contents](#)

**5.1** In the previous part, we used  $k$ -fold cross validation to find the best model. But how confident are you of your estimated best degree? We have already used bootstraps in [Question 2.2](#) to estimate the confidence intervals of our  $\beta_i$  values. In this segment, we will use bootstrapping to test the robustness of our estimation. As before, we use the `bacteria_train.csv` data to train the model and `bacteria_test.csv` to test the model.

- Similar to Question 2.2, run **at least 100 bootstraps** of your data.
- For each bootstrap:
  - Use scikit-learn's `cross_validate(...)` to perform cross validation with  $k=10$  for each degree polynomial regression up to **degree 30**.

- Select the best cross-validated degree polynomial regression based on lowest mean validation  $MSE$  and store that best degree to a list.
- After completing your bootstraps, you should have a list of "best degree" numbers, one degree for each completed bootstrap.
- Generate a bar plot, with the polynomial degree on the  $x$ -axis and the number of times that degree was deemed best on the  $y$ -axis, using the "best degree" numbers saved during your bootstraps.

**NOTE:** Once complete, your code for this problem will likely take several minutes to execute. This is to be expected.

## 5.2 What are your observations?

- Why do you see so much variation in the "best degree" over the bootstraps?
- Which degree polynomial regression will you choose as your overall best degree based on your bootstrapped results, and why?

**5.3** Now, with the overall best degree identified with your bootstrapping results above, train the polynomial regression model on the **complete training set**. Report both the training and test  $MSE$  values, and save the test  $MSE$  to `best_mse_dict`.

**5.4** Generate a plot of the data and your regression curve (similar to [Question 2.1](#)). Comment on how your model fits the data and compare it to the fit of your cross-validated model from [Question 4](#).

# Question 5: Solutions

[Return to contents](#)

**5.1** In the previous part, we used  $k$ -fold cross validation to find the best model. But how confident are you of your estimated best degree? We have already used bootstraps in [Question 2.2](#) to estimate the confidence intervals of our  $\beta_i$  values. In this segment, we will use bootstrapping to test the robustness of our estimation. As before, we use the `bacteria_train.csv` data to train the model and `bacteria_test.csv` to test the model.

- Similar to Question 2.2, run **at least 100 bootstraps** of your data.
- For each bootstrap:
  - Use scikit-learn's `cross_validate(...)` to perform cross validation with  $k=10$  for each degree polynomial regression up to **degree 30**.
  - Select the best cross-validated degree polynomial regression based on lowest mean validation  $MSE$  and store that best degree to a list.
- After completing your bootstraps, you should have a list of "best degree" numbers, one degree for each completed bootstrap.
- Generate a bar plot, with the polynomial degree on the  $x$ -axis and the number of times that degree was deemed best on the  $y$ -axis, using the "best degree" numbers saved during your bootstraps.

**NOTE:** Once complete, your code for this problem will likely take several minutes to execute. This is to be expected.

In [14]:

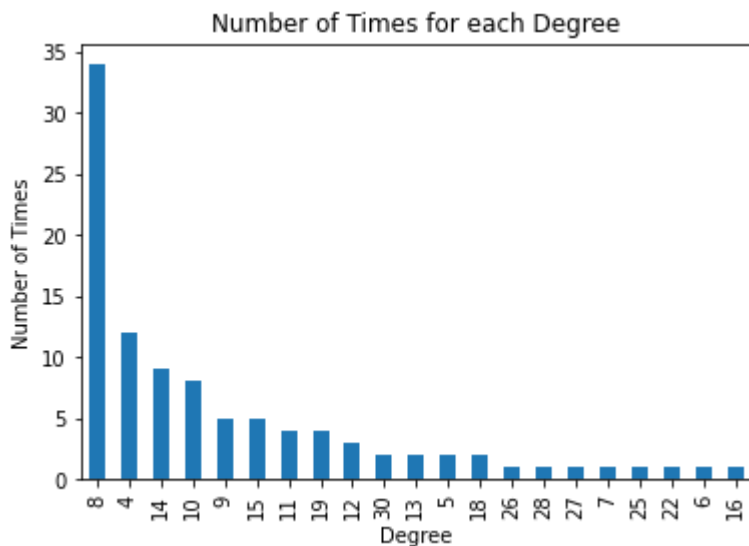
```

1  # your code here
2
3  # bootstrapping
4  boot_linreg_models = []
5  boot_betas = []
6  numboot = 100
7  max_deg=30
8  bestdeg_list = []
9
10 X = bac_tr[['Spreading_factor']]
11 y = bac_tr[['Perc_population']]
12 # X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random
13
14 X_tr_df = X.copy()
15 y_train = y.copy()
16 boot_lreg=LinearRegression(fit_intercept=False)
17
18 all_deg = [i for i in range(1,max_deg+1)]
19
20 # bootstrap
21 for i in range(numboot):
22
23     # randomly sample indices
24     bootstrap_i_indices= np.random.choice(X_tr_df.index, replace = True, size = 1
25
26     # get bootstrapped dfs
27     Xtr_boot = X_tr_df.iloc[bootstrap_i_indices,:]
28     ytr_boot = y_train.iloc[bootstrap_i_indices,:].Perc_population
29
30     # select best
31     cross_validation_error = []
32
33     # cross validation
34     for d in range(1,max_deg+1):
35
36         Xtr_boot_poly = PolynomialFeatures(degree=d).fit_transform(Xtr_boot)
37
38         boot_lreg.fit(Xtr_boot_poly,ytr_boot)
39         mse_validation = cross_validate(boot_lreg, Xtr_boot_poly, ytr_boot,
40                                         scoring = "neg_mean_squared_error",
41                                         cv=10,
42                                         return_train_score = True)['test_score'].mean(
43         cross_validation_error.append(-mse_validation)
44
45
46     bestdeg_list.append(all_deg[cross_validation_error.index(min(cross_validation_error))])
47
48 # Plot
49 fig,ax = plt.subplots()
50 pd.DataFrame(bestdeg_list)[0].value_counts(normalize=False).plot.bar();
51 plt.ylabel('Number of Times')
52 plt.xlabel('Degree')
53 plt.title("Number of Times for each Degree")

```

Out[14]:

Text(0.5, 1.0, 'Number of Times for each Degree')



## 5.2 What are your observations?

- Why do you see so much variation in the "best degree" over the bootstraps?
- Which degree polynomial regression will you choose as your overall best degree based on your bootstrapped results, and why?

From the plot above, I see that most of time in 100 bootstraps the best degree is 8. I see so much variation in the "best degree" over the bootstraps because for each bootstrap the coefficients of the polynomial regression model are different within some confidence intervals. The variance of the models revealed by bootstraps leads to the variance of the best degree. I will choose the model with degree of 8, because in 100 bootstraps, the model with degree of 8 is the best for almost 40 times based on validation MSE from 10 fold cross validation, which is more frequent than others.

**5.3** Now, with the overall best degree identified with your bootstrapping results above, train the polynomial regression model on the **complete training set**. Report both the training and test *MSE* values, and save the **test MSE** to `best_mse_dict`.

In [15]:

```

1  # your code here
2  bestdegree = pd.DataFrame(bestdeg_list).value_counts().to_frame().rename(columns={'Degree': 'count'})
3  X_train_poly = PolynomialFeatures(degree = bestdegree).fit_transform(X)
4  X_test_poly = PolynomialFeatures(degree = bestdegree).fit_transform(X_test)
5  lreg = LinearRegression(fit_intercept=False)
6  lreg.fit(X_train_poly, y)
7  y_train_pred = lreg.predict(X_train_poly)
8  y_test_pred = lreg.predict(X_test_poly)
9  # print(f"The coefficients are {[round(i,4) for i in lreg.coef_[1:]]}.")
10
11 best_mse_dict['5.3'] = mean_squared_error(y_test, y_test_pred)
12 print(f"With polynomial degree of {bestdegree}, the train MSE is {mean_squared_error(y_train_pred, y_train)} and the test MSE is {mean_squared_error(y_test, y_test_pred)}")

```

With polynomial degree of 8, the train MSE is 2.5125, and the test MSE is 2.6130.

**5.4** Generate a plot of the data and your regression curve (similar to [Question 2.1](#)). Comment on how your model fits the data and compare it to the fit of your cross-validated model from [Question 4](#).

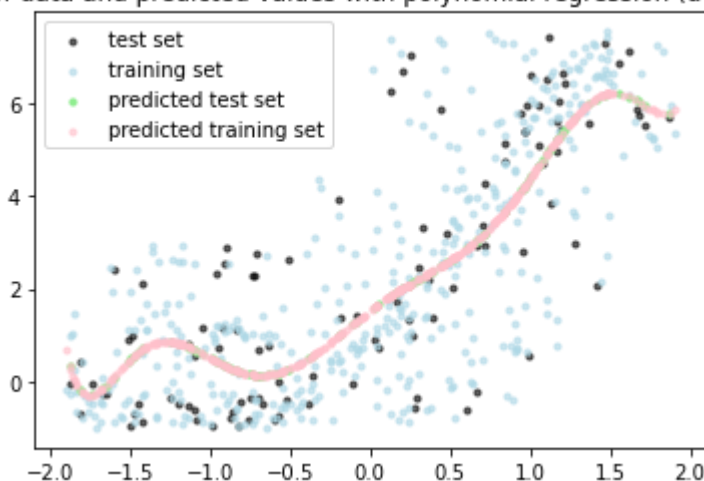
In [16]:

```
1 # your code here
2 fig, ax = plt.subplots()
3 plt.scatter(X_test,y_test,color='k', label = 'test set', s=10, alpha=0.6)
4 plt.scatter(X,y,color='lightblue', label = 'training set', s=10,alpha=0.6)
5 plt.scatter(X_test,y_test_pred,color='lightgreen', label = 'predicted test set',
6 plt.scatter(X,y_train_pred,color='pink', label = 'predicted training set', s=10,
7 plt.title("Plot of data and predicted values with polynomial regression (degree
8 ax.legend()
```

Out[16]:

<matplotlib.legend.Legend at 0x7fc54a85c3d0>

Plot of data and predicted values with polynomial regression (degree 8)



## INTERPRETATION:

The plot above shows that the model (degree 8) fits pretty well. Compared to the model in question 4, this model has 4 more degrees. The model with degree 8 has higher validation MSE than previous model in question 4.

## Question 6: Improving model consistency with LASSO regularization [33 pts]

[Return to contents](#)

In the previous sections, we compared many polynomial models to find the best degree. For each model of degree  $n$ , we considered all polynomial coefficients up to **degree**  $n$  using `PolynomialFeatures(...)`. In this section we will consider polynomial features of  $n = 30$ , and a **best model** that could be chosen from any possible combination of our 30 degrees.

For instance, we could choose a best model with an arbitrary set of polynomial degrees up degree 30, such as  $x^i \in [x^1, x^5, x^{19}, x^{24}]$ , rather than the standard approach of using *all* consecutive degrees up to our maximum degree  $n = 30$  as in  $x^i \in [x^0, x^1, \dots, x^{30}]$ .

However, the total number of such possible models for  $n = 30$  is  $2^{30} \approx 10^9$ , and searching through all of them would be laborious and computationally inefficient. Instead, we can use *LASSO* (i.e. *L1*) regularization, which "switches off" highly unstable degree coefficients by shrinking them to zero (or very close to zero), thus giving us an efficient solution to this particular combinatorial challenge.

### 6.1 Find the best hyper-parameter alpha, $\alpha$

First, we will use cross validation to find the the best regularization parameter  $\alpha$ . As before, we use the `bacteria_train.csv` for cross validation.

- Use polynomial features with **degree 30**.
- For each regularization parameter  $\alpha$  in  $\alpha \in [10^{-3}, 10^{-2}, 10^{-1}, 10^0, 10^1]$ , perform scikit-learn's `Lasso(...)` regression using `cross_validate(...)` with `k=10`.
- Do not worry about normalizing your  $x$  values for any parts of Question 6. It will not meaningfully affect our final results here.
- Generate a single plot showing the mean train and validation *MSE* values for each regularization parameter  $\alpha$ , including the  $\pm 1$  standard deviation bounds for the validation *MSE* values.
- Print the best  $\alpha$  value, identified based on the lowest mean validation *MSE*, along with the corresponding mean train and validation *MSE* values.
- Did your *LASSO* regression model generate any warnings while solving this problem? If so, what was that warning, what did it mean, what steps did you take to address the cause of this warning, and why might it be important to take those steps? (See the [NOTE](#) at the bottom of Question 6 for some hints.)

### 6.2 Find the most significant degrees

From the previous section, you will now have the `best_alpha` hyperparameter. Now, we will use this best  $\alpha$  value to find the "most significant" set of polynomial degrees by using bootstraps.

What we mean by this is, for each bootstrap of the data, you will get some set of degrees that are **significant**, as will be indicated by the value of their coefficients. Typically, while using *LASSO* regularization, we would consider  $|\beta_i| > 0$  as an indication of a particular coefficient's significance. However, for this particular exercise, we are going to use  $|\beta_i| > 0.1$  as our criteria for significance (we have found this 0.1 threshold to work well when designing this particular problem).

Based on this criteria, you will identify which degree coefficients are significant over many bootstrapped iterations with  $n = 30$  polynomial features and your chosen best  $\alpha$  from Question 6.1.

- Use polynomial features with **degree 30**.
- Use a dictionary called `significant_degrees` to store a cumulative count for each degree identified as significant during your bootstraps.
- Run at least 100 bootstraps of your data.
- For each bootstrap:
  - Train a `Lasso(...)` model with the `best_alpha` identified in Question 6.1.
  - Identify the significant polynomial degree coefficients using  $|\beta_i| > 0.1$  as our threshold for significance.
  - For each significant coefficient, update the count of that degree in your `significant_degrees` dictionary.
- Generate a bar plot showing the count for each degree, showing how many times it was identified as significant over your bootstraps.
- Based on this plot, there should likely be a clear maximum degree, above which, few if any iterations have been identified as significant. Interpret your bar plot. choose a maximum degree polynomial based upon it.



and explain your rationale for choosing that degree.

### 6.3 Compute and print the test $MSE$

From our Question 6.2 results, we can see that only some degrees are consistently significant over many bootstraps. You will now retrain the  $L1$  regularized model on the entire **training** data and find the test  $MSE$ .

- Use polynomial features based on the maximum degree you identified using your results from Question 6.2 (for the sake of simplicity, use all degrees up to that maximum degree as is typically done in polynomial regression).
- Train your **LASSO** regularized model using your `best_alpha` hyper-parameter on the entire training set.
- Generate a plot of the data and your regression curve (similar to [Question 2.1](#)).
- Report the polynomial degrees and corresponding coefficients for this fully trained model that have an absolute value greater than 0.1 (i.e.  $|\beta_i| > 0.1$ ).
- Report the train and test  $MSE$  and save the test  $MSE$  to `best_mse_dict`.

### 6.4 What are your observations?

- Compare your `best_degree` from [Question 5](#) with your chosen degree in 6.3 above and comment on the difference.
- Compare your test  $MSE$  in this question with the result from question 5. Which one is smaller, and by how much?
- Which degree coefficients in your best fit model from 6.3 are significant? If not all coefficients are significant, why?
- Would you expect your current model from 6.3 to be more "reliable" or robust than the models in Question 5 and prior? On what basis would you make a claim of its robustness?

#### NOTE:

- Once complete, your code for 6.1 and 6.2 will likely take several minutes to execute. This is to be expected.
- You will also likely receive `ConvergenceWarning` messages as your  $LASSO$  regression models are trained.
  - Try resolving this warning by increasing your  $LASSO$  models' maximum iterations by between 20-times to 100-times greater than scikit-learn's default number of iterations for its `Lasso(...)` implementation.
  - If that fails to resolve this warning, feel free to also uncomment the [IPython cell magic](https://ipython.readthedocs.io/en/stable/interactive/magics.html#cellmagic-capture) (<https://ipython.readthedocs.io/en/stable/interactive/magics.html#cellmagic-capture>) provided at the top the code cell (i.e. `%%capture --no-stdout --no-display`) to silence the warning.
  - Please note that using a `%%capture` cell magic such as this is a good alternative to using Python's native `warnings.filterwarnings(...)` when working in a Jupyter notebook, because it helps to ensure you don't accidentally silence important warnings in other parts of your notebook.

## Question 6: Solutions

[Return to contents](#)

### 6.1 Find the best hyper-parameter $\alpha$ , $\alpha$

First, we will use cross validation to find the the best regularization parameter  $\alpha$ . As before, we use the `bacteria_train.csv` for cross validation.



- Use polynomial features with **degree 30**.
- For each regularization parameter  $\alpha$  in  $\alpha \in [10^{-3}, 10^{-2}, 10^{-1}, 10^0, 10^1]$ , perform scikit-learn's `Lasso(...)` regression using `cross_validate(...)` with `k=10`.
- Do not worry about normalizing your  $x$  values for any parts of Question 6. It will not meaningfully affect our final results here.
- Generate a single plot showing the mean train and validation  $MSE$  values for each regularization parameter  $\alpha$ , including the  $\pm 1$  standard deviation bounds for the validation  $MSE$  values.
- Print the best  $\alpha$  value, identified based on the lowest mean validation  $MSE$ , along with the corresponding mean train and validation  $MSE$  values.
- Did your  $LASSO$  regression model generate any warnings while solving this problem? If so, what was that warning, what did it mean, what steps did you take to address the cause of this warning, and why might it be important to take those steps? (See the [NOTE](#) at the bottom of Question 6 for some hints.)

In [17]:

```

1 %%capture --no-stdout --no-display
2
3 # your code here
4 X = bac_tr[['Spreading_factor']]
5 y = bac_tr[['Perc_population']]
6
7 # shuffle the data
8 indices= np.random.choice(X.index, replace = False, size = len(X.index))
9 X = X.iloc[indices,:]
10 y = y.iloc[indices,:].Perc_population
11
12 max_deg = 30
13 k=10
14 alphas = [1e-3, 1e-2, 1e-1, 1, 10]
15 cross_validation_error = []
16 cross_train_error = []
17 std_dict = {}
18 for a in alphas:
19
20     # cross validation
21     X_poly_train = PolynomialFeatures(degree=max_deg).fit_transform(X)
22     lassoreg = Lasso(alpha = a,max_iter=2000)#.fit(X_poly_train,y)
23
24     validation_result = cross_validate(lassoreg, X_poly_train, y,
25                                       scoring = "neg_mean_squared_error",
26                                       cv=k,
27                                       return_train_score = True)
28     val_scores = validation_result['test_score']
29     mse_validation = val_scores.mean()
30     mse_train = validation_result['train_score'].mean()
31
32     std_dict[a] = -val_scores
33     cross_validation_error.append(-mse_validation)
34     cross_train_error.append(-mse_train)
35
36 best_alpha = alphas[cross_validation_error.index(min(cross_validation_error))]
```

In [18]:

```

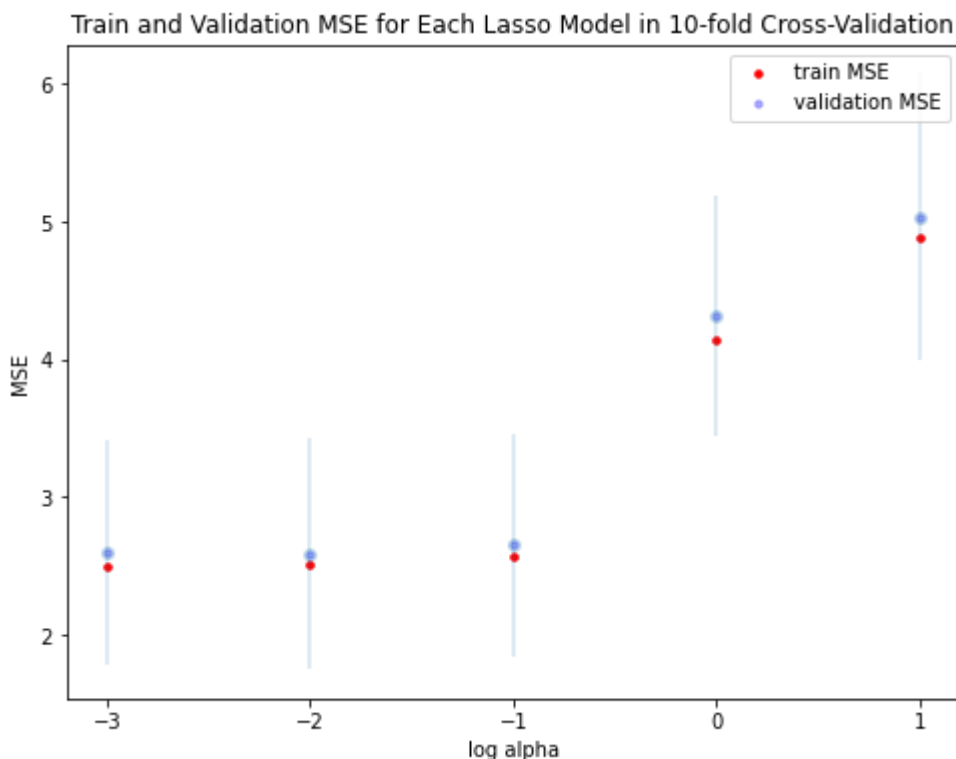
1 print(f"The alpha of the best lasso model with the lowest validation MSE is {bes
2
3 df = pd.DataFrame(std_dict).melt()
4 df2 = df.groupby("variable").agg('std').reset_index()
5 std_arr = np.array(df2['value'])
6
7 # plot MSE
8 fig, ax = plt.subplots(figsize = (8,6))
9 plt.scatter(np.log10(alphas),cross_train_error, color='red', label = 'train MSE'
10 plt.scatter(np.log10(alphas),cross_validation_error,color='blue', label = 'valid
11 plt.errorbar(np.log10(alphas),cross_validation_error, yerr=std_arr, fmt="o", alp
12 plt.xlabel("log alpha")
13 plt.ylabel("MSE")
14 plt.xticks(np.log10(alphas))
15 plt.title("Train and Validation MSE for Each Lasso Model in 10-fold Cross-Valida
16 ax.legend()

```

The alpha of the best lasso model with the lowest validation MSE is 0.01. The lowest validation MSE is 2.5906. The corresponding train MSE is 2.5048.

Out[18]:

&lt;matplotlib.legend.Legend at 0x7fc54a8b8c40&gt;

**INTERPRETATION:**

**Yes there are warnings. It says "ConvergenceWarning: Objective did not converge. You might want to**

increase the number of iterations". It means the model does not converge. To avoid warning, I tried to increase iteration by setting the `max_iter` to be a larger value, but this does not work. The reason why increasing the number of iteration might be important is that the default iteration is 1000 which is too small to make the model reach the convergence. When the iteration number is large, the model is more likely to converge. Then I include `"%%capture --no-stdout --no-display"` in the jupyter block to silence the warning.

## 6.2 Find the most significant degrees

From the previous section, you will now have the `best_alpha` hyperparameter. Now, we will use this best  $\alpha$  value to find the "most significant" set of polynomial degrees by using bootstraps.

What we mean by this is, for each bootstrap of the data, you will get some set of degrees that are **significant**, as will be indicated by the value of their coefficients. Typically, while using *LASSO* regularization, we would consider  $|\beta_i| > 0$  as an indication of a particular coefficient's significance. However, for this particular exercise, we are going to use  $|\beta_i| > 0.1$  as our criteria for significance (we have found this 0.1 threshold to work well when designing this particular problem).

Based on this criteria, you will identify which degree coefficients are significant over many bootstrapped iterations with  $n = 30$  polynomial features and your chosen best  $\alpha$  from Question 6.1.

- Use polynomial features with **degree 30**.
- Use a dictionary called `significant_degrees` to store a cumulative count for each degree identified as significant during your bootstraps.
- Run at least 100 bootstraps of your data.
- For each bootstrap:
  - Train a `Lasso(...)` model with the `best_alpha` identified in Question 6.1.
  - Identify the significant polynomial degree coefficients using  $|\beta_i| > 0.1$  as our threshold for significance.
  - For each significant coefficient, update the count of that degree in your `significant_degrees` dictionary.
- Generate a bar plot showing the count for each degree, showing how many times it was identified as significant over your bootstraps.
- Based on this plot, there should likely be a clear maximum degree, above which, few if any iterations have been identified as significant. Interpret your bar plot, choose a maximum degree polynomial based upon it, and explain your rationale for choosing that degree.

In [19]:

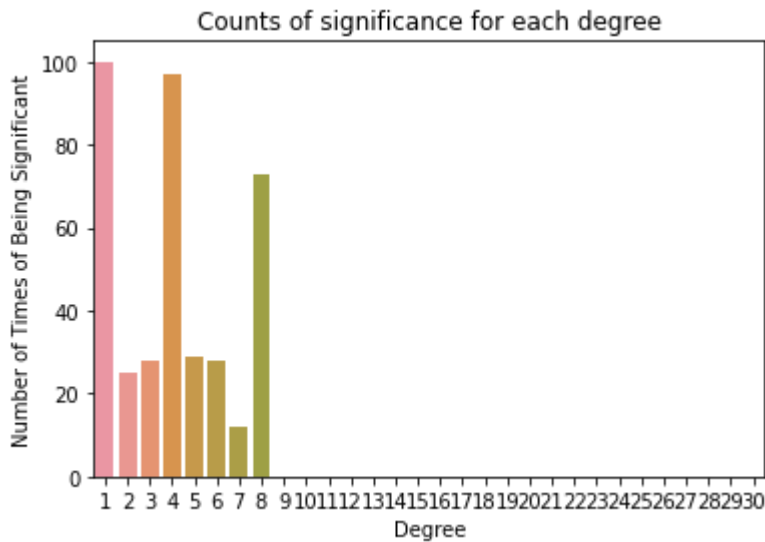
```

1 %%capture --no-stdout --no-display
2
3 # your code here
4
5 boot_linreg_models = []
6 boot_betas = []
7 numboot = 100
8 max_deg=30
9 bestdeg_list = []
10 significant_degrees = {}
11 for j in range(1,max_deg+1):
12     significant_degrees[j] = 0
13
14 X = bac_tr[['Spreading_factor']]
15 y = bac_tr[['Perc_population']]
16 # X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random
17
18 X_tr_df = X.copy()
19 y_train = y.copy()
20
21
22 all_deg = [i for i in range(1,max_deg+1)]
23
24 # bootstrap
25 for i in range(numboot):
26
27     # randomly sample indices
28     bootstrap_i_indices= np.random.choice(X_tr_df.index, replace = True, size = 1
29
30     # get bootstrapped dfs
31     Xtr_boot = X_tr_df.iloc[bootstrap_i_indices,:]
32     ytr_boot = y_train.iloc[bootstrap_i_indices,:].Perc_population
33
34     # select best
35     cross_validation_error = []
36
37     # a lasso model
38     Xtr_boot_poly = PolynomialFeatures(degree=max_deg).fit_transform(Xtr_boot)
39     lassoreg = Lasso(alpha = best_alpha,max_iter=2000)#.fit(X_poly_train,y)
40
41     lassoreg.fit(Xtr_boot_poly,ytr_boot)
42     y_boot_pred = lassoreg.predict(Xtr_boot_poly)
43     mse_validation = mean_squared_error(y_boot_pred, ytr_boot)
44     cross_validation_error.append(-mse_validation)
45
46     for j in range(1,max_deg+1):
47         if (lassoreg.coef_[1:][j-1] > 0.1 or lassoreg.coef_[1:][j-1] < -0.1):
48             value = significant_degrees[j]
49             significant_degrees[j] = value + 1
50
51
52 newdf = pd.DataFrame(significant_degrees.values()).reset_index().rename(columns=
53 newdf['index'] = newdf['index'] + 1
54
55 # Plot
56 sns.barplot(data = newdf, x = 'index', y = 'value')
57 plt.ylabel('Number of Times of Being Significant')
58 plt.xlabel('Degree')
59 plt.title("Counts of significance for each degree")

```

Out[19]:

Text(0.5, 1.0, 'Counts of significance for each degree')



## INTERPRETATION:

Based on this plot, the maximum degree is 8, because the degrees larger than 8 did not be identified as significant. Lasso regression can be seen as a method for feature selection. Based on 100 bootstrapping, the Lasso gives zero coefficients for those with degrees larger than 8. Therefore, we are confident that those degrees larger than 8 are not important.

### 6.3 Compute and print the test $MSE$

From our Question 6.2 results, we can see that only some degrees are consistently significant over many bootstraps. You will now retrain the  $L1$  regularized model on the entire **training** data and find the test  $MSE$ .

- Use polynomial features based on the maximum degree you identified using your results from Question 6.2 (for the sake of simplicity, use all degrees up to that maximum degree as is typically done in polynomial regression).
- Train your **LASSO** regularized model using your `best_alpha` hyper-parameter on the entire training set.
- Generate a plot of the data and your regression curve (similar to [Question 2.1](#)).
- Report the polynomial degrees and corresponding coefficients for this fully trained model that have an absolute value greater than 0.1 (i.e.  $|\beta_i| > 0.1$ ).
- Report the train and test  $MSE$  and save the test  $MSE$  to `best_mse_dict`.

In [20]:

```

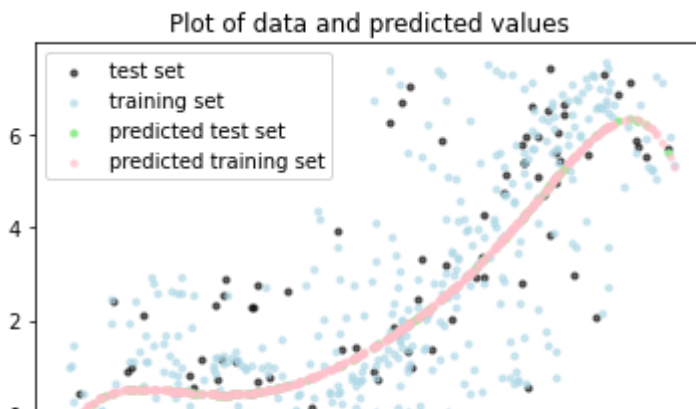
1  # your code here
2  lasso_deg = 8
3  X_train_poly = PolynomialFeatures(degree = lasso_deg).fit_transform(X)
4  X_test_poly = PolynomialFeatures(degree = lasso_deg).fit_transform(X_test)
5  lassoreg = Lasso(alpha = best_alpha).fit(X_train_poly, y)
6  y_train_pred = lassoreg.predict(X_train_poly)
7  y_test_pred = lassoreg.predict(X_test_poly)
8
9  list_coef = [round(i,4) for i in lassoreg.coef_[1:]]
10
11 best_mse_dict['6.3'] = mean_squared_error(y_test, y_test_pred)
12 print(f"With polynomial degree of 8, the corresponding coefficients are {list_coef}")
13
14
15 # plot
16 fig, ax = plt.subplots()
17 plt.scatter(X_test, y_test, color='k', label = 'test set', s=10, alpha=0.6)
18 plt.scatter(X, y, color='lightblue', label = 'training set', s=10, alpha=0.6)
19 plt.scatter(X_test, y_test_pred, color='lightgreen', label = 'predicted test set', s=10, alpha=0.6)
20 plt.scatter(X, y_train_pred, color='pink', label = 'predicted training set', s=10, alpha=0.6)
21 plt.title("Plot of data and predicted values")
22 ax.legend()

```

model\_coordinate\_descent.py:350: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Dual  
ity gap: 14.496300864179375, tolerance: 0.2716198598316785  
model = cd\_fast.enet\_coordinate\_descent(

Out[20]:

<matplotlib.legend.Legend at 0x7fc58921efa0>



## 6.4 What are your observations?

- Compare your `best_degree` from [Question 5](#) with your chosen degree in 6.3 above and comment on the difference.
- Compare your test  $MSE$  in this question with the result from question 5. Which one is smaller, and by how much?
- Which degree coefficients in your best fit model from 6.3 are significant? If not all coefficients are significant, why?
- Would you expect your current model from 6.3 to be more "reliable" or robust than the models in Question 5 and prior? On what basis would you make a claim of its robustness?

In [21]:

```
1 best_mse_dict
```

Out[21]:

```
{'2.1': 2.6519056545149615,  
'3.2': 2.67486257544031,  
'4.2': 2.61301063735301,  
'5.3': 2.6130106373530064,  
'6.3': 2.594991995723685}
```

The best degree from Q5 and my chosen degree in 6.3 are the same. The difference is that by using Lasso I got lower validation MSE by about 0.02. Not all coefficients are significant, only 4 of them have absolute coefficient values larger than 0.1. This is because by using the original training data (different from the sampled data in previous bootstrap), Lasso identified some coefficients as insignificant and assigned their coefficient to a value closer to null. If we resample the data we will possibly get slightly different results. I would expect the current model to be more reliable, more generalizable, less overfitting. Because Lasso added penalty/regularization term to the model which makes the model more tolerant to the error and perform better on the unknown test data. This is why I make a claim of its robustness.

## Question 7: Analyze your best test MSEs for each sub section of the homework [6 pts]

[Return to contents](#)

**7.1** Convert the `best_mse_dict` dictionary used to store the best  $MSE$  values from each section of the homework into a Pandas dataframe. Generate a bar plot illustrating the  $MSE$  values in that dataframe and also display the resulting dataframe.

**7.2** Which model do you think is best, and why? What are some of that model's drawbacks compared to the other models?

## Question 7: Solutions

[Return to contents](#)

**7.1** Convert the `best_mse_dict` dictionary used to store the best  $MSE$  values from each section of the homework into a Pandas dataframe. Generate a bar plot illustrating the  $MSE$  values in that dataframe and also display the resulting dataframe.

In [22]:

```

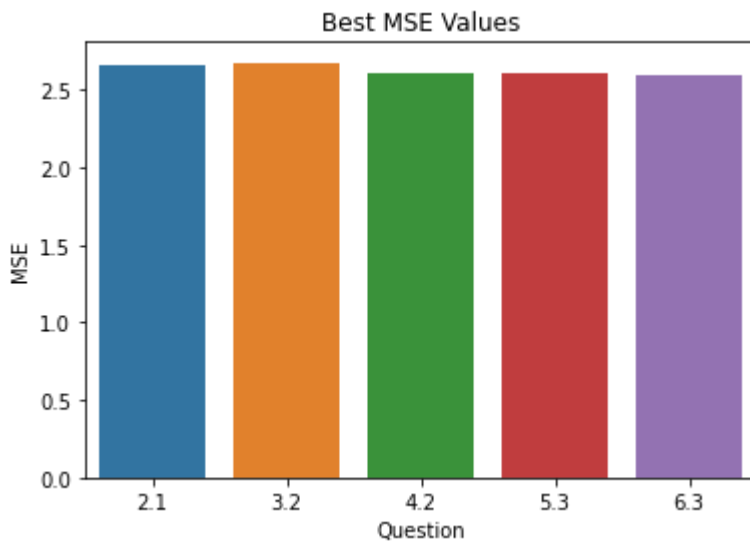
1 # your code here
2 best_mse_df = pd.DataFrame(best_mse_dict.keys(), best_mse_dict.values()).reset_i
3 print(best_mse_df.head())
4 sns.barplot(data = best_mse_df, x = 'Question', y = 'MSE')
5 plt.title('Best MSE Values')
6

```

	MSE	Question
0	2.651906	2.1
1	2.674863	3.2
2	2.613011	4.2
3	2.613011	5.3
4	2.594992	6.3

Out[22]:

Text(0.5, 1.0, 'Best MSE Values')



**7.2** Which model do you think is best, and why? What are some of that model's drawbacks compared to the other models?

The model in 6.3 is the best because it has the lowest MSE score on the test set. Compared to other models, it has less predictors, so it can be underfit. This means it might have lower variance but higher bias.

**THE END**



